**ACKNOWLEDGEMENT**

With sincere appreciation and gratitude, I wish to acknowledge several persons whose valuable support and ideas have contributed much in the conceptualization of this practical journal.

I would like to thank our mam **Prof. Sweta Iyer** for helping in completing the practicals.

I would like to thank our H.O.D for providing us the necessary infrastructure which has helped us in preparing our journal.

I am indebted to all the sources that from where I have collected this information.

Any consecutive suggestions for further improvement would be highly appreciated.

 Vidyavardhini’s

Anna Saheb Vartak College of Arts,

Kedarnath Malhotra College of Commerce,

&

E.S.Andrades College of Science

Vasai Road, Dist. Palghar -401202

**CERTIFICATE**

This is to certify that Mr./Miss.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Exam Seat No.\_\_\_\_\_\_\_\_\_\_\_\_ has satisfactorily completed practical’s of the subject **“Data Science”** as part of the practical fulfillment of MSc.IT Sem I as prescribed by the university of Mumbai for the year 2022-2023.

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Subject Teacher External Examiner H.O.D.

**UNIVERSITY OF MUMBAI**



**A Practical Report on**

**“SOFT COMPUTING”**

**Submitted by**

**Harsh Avinash Save**

MSc (IT) - I Semester - I

Academic Year 2022-2023

**Under The Guidance of**

**Prof. Sweta Iyer**

**Vidyavardhini’s**

**Annasaheb Vartak College of Arts,**

**Kedarnath Malhotra College of Commerce,**

**&**

**E.S.Andrades College of Science**

**Vasai Road,Dist.Palghar -401202**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Prac. No.** | **Details** | **Page No.** |
| **1** | Convert Data into horus format | **5** |
| **2** | Utilities and Auditing | **16** |
| **3** | Retrieve Superstep | **21** |
| **4** | Processing Data | **39** |
| **5** | Organizing Data | **54** |
| **6** | Transform superstep | **73** |
| **7** | Generating Data | **83** |
| **8** | Data Visualization with Power BI | **96** |

**Practical-1**

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

1. **Text delimited CSV to HORUS format.**

**Code:**

import pandas as pd

inputDataFile='./country\_codes.csv'

inputData=pd.read\_csv(inputDataFile)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print('\*\*\*\* Input Data \*\*\*\*')

print(inputData)

processedData=inputData

processedData.drop('ISO-2-CODE',axis=1,inplace=True)

processedData.drop('ISO-3-Code',axis=1,inplace=True)

processedData.rename(columns={'Country':'CountryName'},inplace=True)

processedData.rename(columns={'ISO-M49':'CountryCode'},inplace=True)

processedData.set\_index('CountryCode',inplace=True)

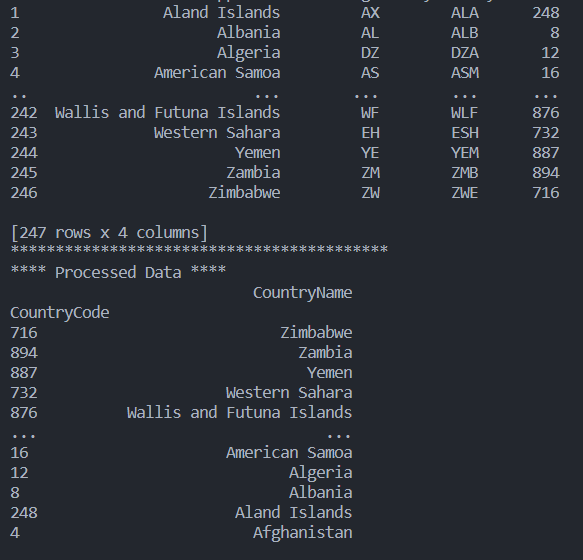
processedData.sort\_values('CountryName',inplace=True,axis=0,ascending=False)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print('\*\*\*\* Processed Data \*\*\*\*')

print(processedData)

**Output:**

****

1. **XML to HORUS Format**

**Code:**

import pandas as pd

import xml.etree.ElementTree as ET

def xmlToDf(xml):

root=ET.XML(xml)

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)

inputFileName='Practical 1\country\_code.xml'

inputData=open(inputFileName).read()

processedData=inputData

processedData=xmlToDf(processedData)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Input Data")

print(processedData)

processedData.drop('ISO-2-CODE',axis=1,inplace=True)

processedData.drop('ISO-3-Code',axis=1,inplace=True)

processedData.rename(columns={'Country':'CountryName'},inplace=True)

processedData.rename(columns={'ISO-M49':'CountryCode'},inplace=True)

processedData.set\_index('CountryCode',inplace=True)

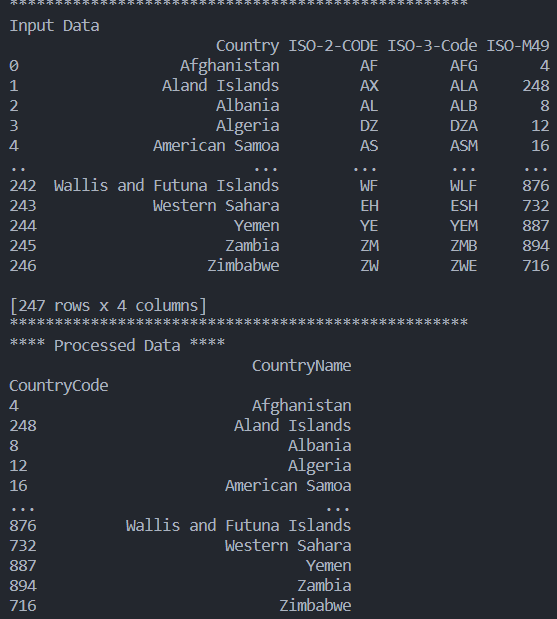
processedData.sort\_values('CountryName',inplace=True,axis=0,ascending=True)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print('\*\*\*\* Processed Data \*\*\*\*')

print(processedData)

**Output:**



1. **JSON to HORUS Format**

**Code:**

import pandas as pd

inputFile='Practical 1\Country\_Code.json'

inputData=pd.read\_json(inputFile,orient='index')

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print('\*\*\* Input Data \*\*\*')

print(inputData)

processedData=inputData

processedData.drop('ISO-2-CODE',axis=1,inplace=True)

processedData.drop('ISO-3-Code',axis=1,inplace=True)

processedData.rename(columns={'ISO-M49':'CountryCode'},inplace=True)

processedData.rename(columns={'Country':'CountryName'},inplace=True)

processedData.set\_index('CountryCode',inplace=True)

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

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print('\*\*\* Processed Data \*\*\*')

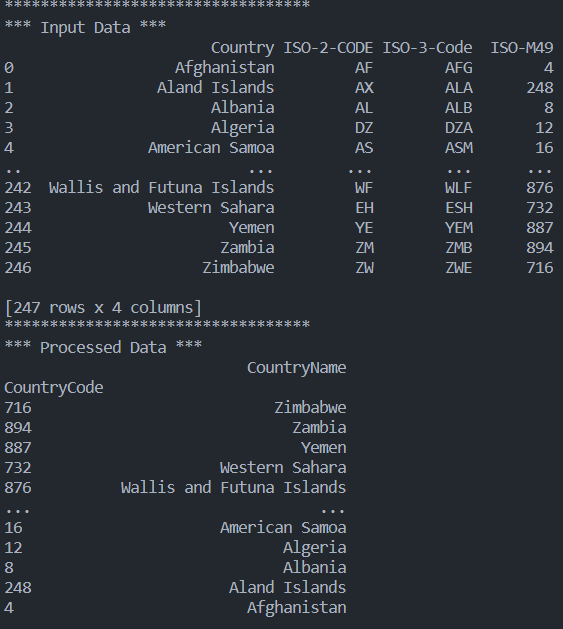
print(processedData)

outputData=processedData

outputFile='./Output/JSONToHorus.csv'

outputData.to\_csv(outputFile)

**Output:**

****

1. **MySql Database to HORUS Format**

**Code:**

import sqlite3 as sq

import pandas as pd

inputFile='utility.db'

inputTable='Country\_Code'

conn=sq.connect(inputFile)

inputQuery='select \* from ' + inputTable+';'

inputData=pd.read\_sql\_query(inputQuery,conn)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print('\*\*\* Input Data \*\*\*')

print(inputData)

processedData=inputData

processedData.drop('ISO-2-CODE',axis=1,inplace=True)

processedData.drop('ISO-3-Code',axis=1,inplace=True)

processedData.rename(columns={'ISO-M49':'CountryCode'},inplace=True)

processedData.rename(columns={'Country':'CountryName'},inplace=True)

processedData.set\_index('CountryCode',inplace=True)

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

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print('\*\*\* Processed Data \*\*\*')

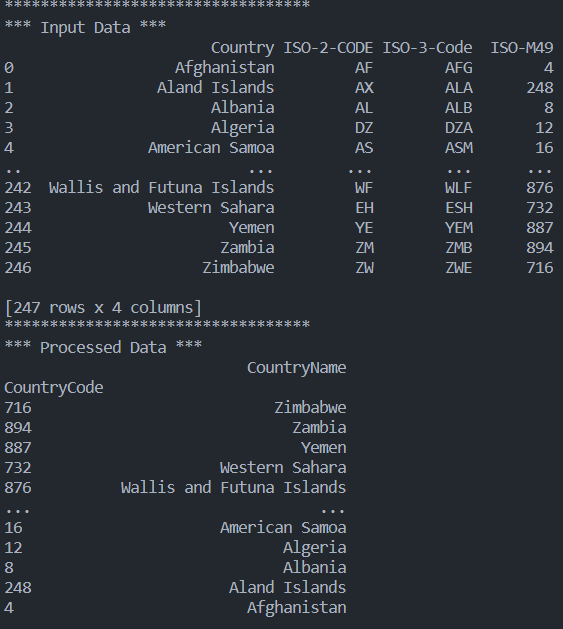
print(processedData)

outputData=processedData

outputFile='./Output/DBToHorus.csv'

outputData.to\_csv(outputFile)

**Output:**

****

1. **Picture (JPEG) to HORUS Format**

**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

inputFileName='./Angus.jpg'

inputData=plt.imread(inputFileName)

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

print("X= ",inputData.shape[0])

print("Y= ",inputData.shape[1])

print("RGBA= ",inputData.shape[2])

processRawData=inputData.flatten()

y=inputData.shape[2]+2

x=int(processRawData.shape[0]/y)

processData=pd.DataFrame(np.reshape(processRawData,(x,y)))

sColumns=['Xaxis','Yaxis','Red','Green','Blue']

processData.columns=sColumns

processData.index.names=['ID']

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

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

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

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

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

plt.imshow(inputData)

plt.show()

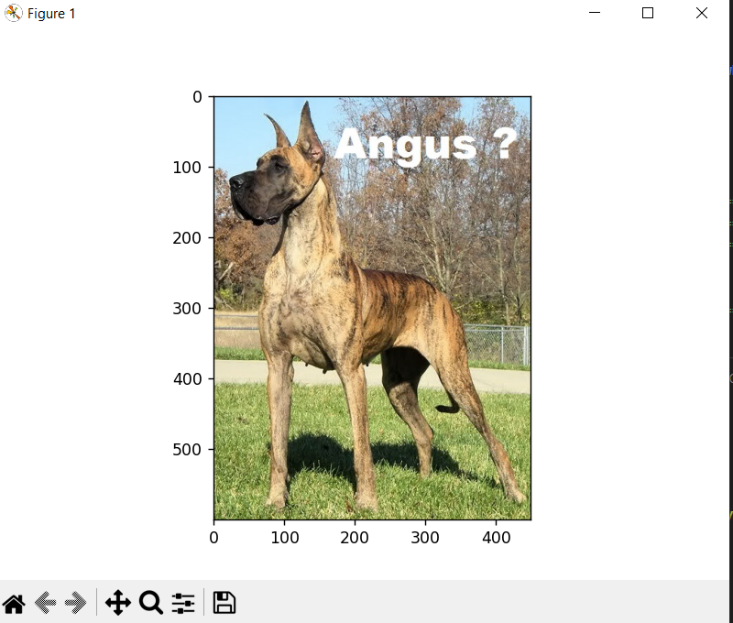
print('=====================================================')

outputData = processData

outputFileName='./Output/horus\_picture.csv'

outputData.to\_csv(outputFileName,index=False)

**Output:**

****

1. **Video to HORUS Format**

**Code:**

import os

import shutil

from sre\_constants import SUCCESS

import cv2

inputFileName ='./Dog.mp4'

baseDir='./temp'

if not os.path.exists(baseDir):

os.mkdir(baseDir)

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

print("Start movie to frames")

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

vidcap=cv2.VideoCapture(inputFileName)

success,image=vidcap.read()

count=0

while success:

success,image=vidcap.read()

sFrame=baseDir+ str('/dog-frame-' + str(format(count, '04d'))+ '.jpg')

print("Extracted ",sFrame)

cv2.imwrite(sFrame,image)

if os.path.getsize(sFrame)==0:

count+=-1

os.remove(sFrame)

print("Removed ",sFrame)

if cv2.waitKey(10)==27:

break

count+=1

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

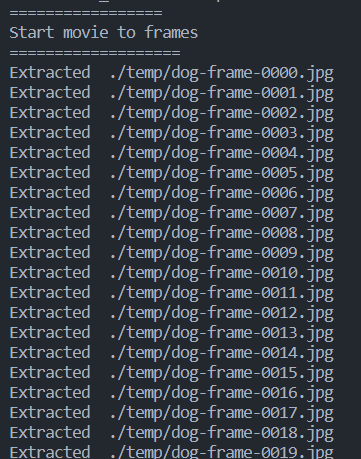
print("Generated frames")

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

print("Generated ",count,'frames')

print("Done")

**Output:**



**Frames to Horus**

**Code:**

from scipy.misc import imread

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import os

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

sDataBaseDir='C:/VKHCG/05-DS/9999-Data/temp'

f=0

for file in os.listdir(sDataBaseDir):

if file.endswith(".jpg"):

f += 1

sInputFileName=os.path.join(sDataBaseDir, file)

print('Process : ', sInputFileName)

InputData = imread(sInputFileName, flatten=False, mode='RGBA')

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

print('X: ',InputData.shape[0])

print('Y: ',InputData.shape[1])

print('RGBA: ', InputData.shape[2])

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

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

ProcessRawData=InputData.flatten()

y=InputData.shape[2] + 2

x=int(ProcessRawData.shape[0]/y)

ProcessFrameData=pd.DataFrame(np.reshape(ProcessRawData, (x, y)))

ProcessFrameData['Frame']=file

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

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

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

plt.imshow(InputData)

plt.show()

if f == 1:

ProcessData=ProcessFrameData

else:

ProcessData=ProcessData.append(ProcessFrameData)

if f > 0:

sColumns= ['XAxis','YAxis','Red', 'Green', 'Blue','Alpha','FrameName']

ProcessData.columns=sColumns

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

ProcessFrameData.index.names =['ID']

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

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

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

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

OutputData=ProcessData

print('Storing File')

sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-Movie-Frame.csv'

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

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

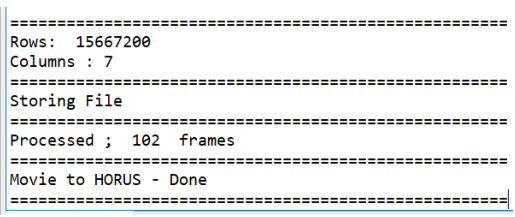
print('Processed ; ', f,' frames')

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

print('Movie to HORUS - Done')

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

**Output:**

****

1. **Audio to HORUS Format**

**Code:**

from scipy.io import wavfile

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

def show\_info(aname,a,r):

print('----------------')

print("Audio: ",aname)

print('----------------')

print("Rate: ",r)

print('----------------')

print("Shape: ",a.shape)

print("dtype: ",a.dtype)

print("Min, Max ",a.min(),a.max())

print('----------------')

plot\_info(aname,a,r)

def plot\_info(aname,a,r):

sTitle='Signal wave'+aname+'at'+str(r)+'hz'

plt.title(sTitle)

sLegend=[]

for c in range(a.shape[1]):

sLabel='Ch'+str(c+1)

sLegend.append(str(c+1))

plt.plot(a[:,c],label=sLabel)

plt.legend(sLegend)

plt.show()

inputFileName='./VKHCG\_05-DS\_9999-Data\_2ch-sound.wav'

inputRate,inputData=wavfile.read(inputFileName)

show\_info("2 Channel",inputData,inputRate)

processData=pd.DataFrame(inputData)

sColumns=['Ch1','Ch2']

processData.columns=sColumns

outputData=processData

outputFileName='./Output/HORUS-Audio-2ch.csv'

outputData.to\_csv(outputFileName,index=False)

inputFileName='./VKHCG\_05-DS\_9999-Data\_4ch-sound.wav'

inputRate,inputData=wavfile.read(inputFileName)

show\_info("4 Channel",inputData,inputRate)

processData=pd.DataFrame(inputData)

sColumns=['Ch1','Ch2','Ch3','Ch4']

processData.columns=sColumns

outputData=processData

outputFileName='./Output/HORUS-Audio-4ch.csv'

outputData.to\_csv(outputFileName,index=False)

inputFileName='./VKHCG\_05-DS\_9999-Data\_6ch-sound.wav'

inputRate,inputData=wavfile.read(inputFileName)

show\_info("6 Channel",inputData,inputRate)

processData=pd.DataFrame(inputData)

sColumns=['Ch1','Ch2','Ch3','Ch4','Ch5','Ch6']

processData.columns=sColumns

outputData=processData

outputFileName='./Output/HORUS-Audio-6ch.csv'

outputData.to\_csv(outputFileName,index=False)

inputFileName='./8Ch-sound.wav'

inputRate,inputData=wavfile.read(inputFileName)

show\_info("8 Channel",inputData,inputRate)

processData=pd.DataFrame(inputData)

sColumns=['Ch1','Ch2','Ch3','Ch4','Ch5','Ch6','Ch7','Ch8']

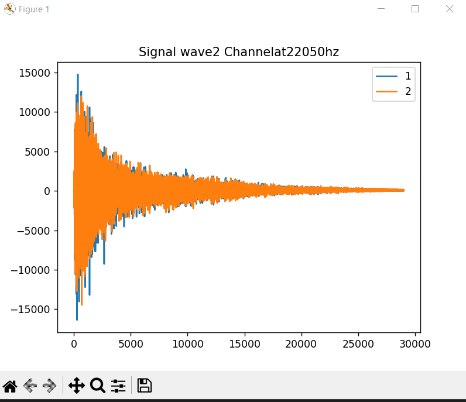
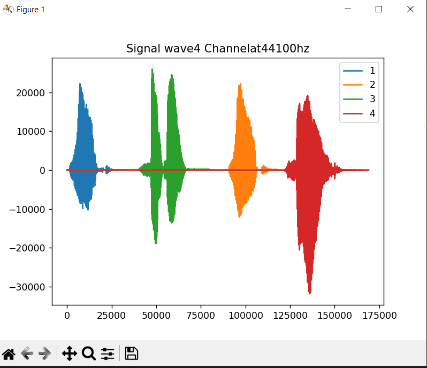
processData.columns=sColumns

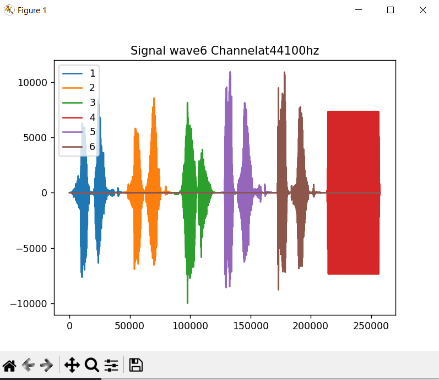
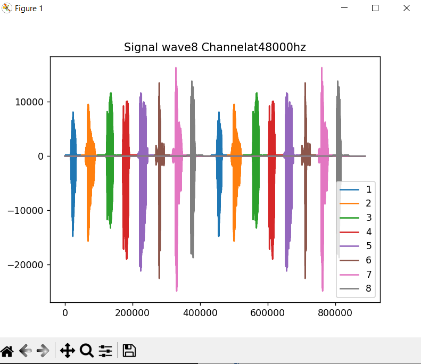
outputData=processData

outputFileName='./Output/HORUS-Audio-8ch.csv'

outputData.to\_csv(outputFileName,index=False)

**Output:**





**Practical-2**

1. **Fixers Utilities**
2. **Removing leading or lagging spaces from a data entry**

print('#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,'<')

1. **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)

1. **Reformatting data entry to match specific formatting criteria.**

print('# 3 Reformatting data entry to match specific formatting criteria.')

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

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

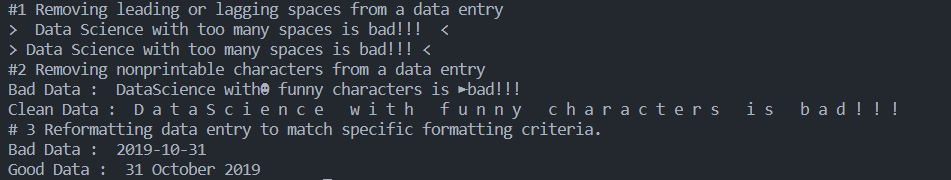
gooddate = dt.datetime.strptime(baddata,'%Y-%m-%d')

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

print('Bad Data : ',baddata)

print('Good Data : ',gooddata)

**Output:**

****

1. **Data Binning or Bucketing**

**Code:**

import numpy as np

import matplotlib.mlab as mlab

import matplotlib.pyplot as plt

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, density=1)

# 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')

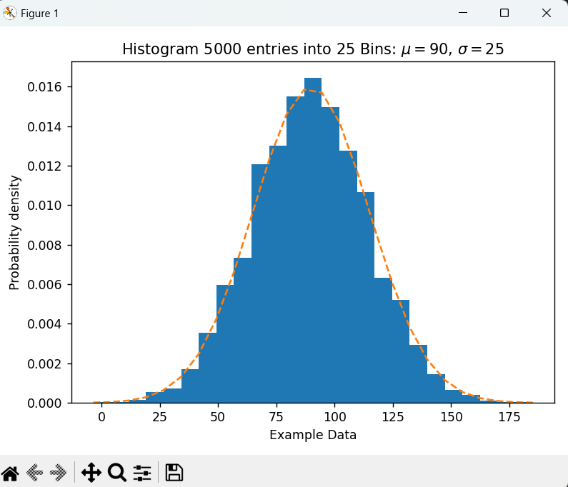
sTitle=r'Histogram ' + str(len(x)) + ' entries into ' + str(num\_bins) + ' Bins: $\mu=' + str(mu) + '$, $\sigma=' + str(sigma) + '$'

ax.set\_title(sTitle)

fig.tight\_layout()

plt.show()

**Output:**

****

1. **Averaging of Data**

**Code:**

import pandas as pd

inputFilePath='./IP\_DATA\_CORE.csv'

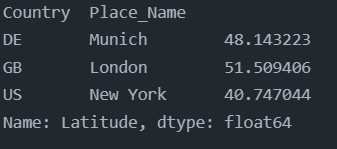
IP\_DATA\_ALL=pd.read\_csv(inputFilePath,usecols=['Country','Place Name','Latitude','Longitude'])

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

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

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

**Output:**

****

1. **Outlier Detection**

**Code:**

import pandas as pd

inputFilePath='./IP\_DATA\_CORE.csv'

IP\_DATA\_ALL=pd.read\_csv(inputFilePath,usecols=['Country','Place Name','Latitude','Longitude'])

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

LondonData=IP\_DATA\_ALL.loc[IP\_DATA\_ALL['Place\_Name']=='London']

AllData=LondonData[['Country','Place\_Name','Latitude']]

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

StdData=AllData.groupby(['Country','Place\_Name'])['Latitude'].std()

upperBound=float(MeanData+StdData)

lowerBound=float(MeanData-StdData)

print('Higher Outliers')

outlierHigher=AllData[AllData.Latitude>upperBound]

print(outlierHigher)

print('Lower Outliers')

outlierLower=AllData[AllData.Latitude<lowerBound]

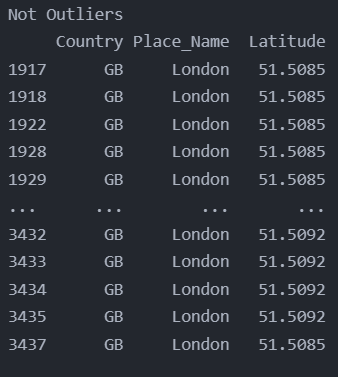
print(outlierLower)

print('Not Outliers')

notOutliers=AllData[(AllData.Latitude=>lowerBound)&(AllData.Latitude<=upperBound)]

print(notOutliers)

**Output:**

****

1. **Logging**

**Code:**

import sys

import os

import logging

import uuid

import shutil

import time

Base='C:/VKHCG'

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

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

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

sLevels=['debug','info','warning','error']for sCompany in sCompanies:

sFileDir=Base + '/' + sCompany

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

for sLayer in sLayers:

log = logging.getLogger() # root logger

for hdlr in log.handlers[:]: # remove all old handlers

log.removeHandler(hdlr)

#----------------------------------------------------------------------------------

sFileDir=Base + '/' + sCompany + '/' + sLayer + '/Logging'

if os.path.exists(sFileDir):

shutil.rmtree(sFileDir)

time.sleep(2)

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

skey=str(uuid.uuid4())

sLogFile=Base + '/' + sCompany + '/' + sLayer + '/Logging/Logging\_'+skey+'.log'

print('Set up:',sLogFile)

# set up logging to file - see previous section for more details

logging.basicConfig(level=logging.DEBUG,

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

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

filename=sLogFile,

filemode='w')

# define a Handler which writes INFO messages or higher to the sys.stderr

console = logging.StreamHandler()

console.setLevel(logging.INFO)

# set a format which is simpler for console use

formatter = logging.Formatter('%(name)-12s: %(levelname)-8s %(message)s')

# tell the handler to use this format

console.setFormatter(formatter)

# add the handler to the root logger

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

# Now, we can log to the root logger, or any other logger. First the root...

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

for sLevel in sLevels:

sApp='Apllication-'+ sCompany + '-' + sLayer + '-' + sLevel

logger = logging.getLogger(sApp)

if sLevel == 'debug':

logger.debug('Practical Data Science logged a debugging message.')

if sLevel == 'info':

logger.info('Practical Data Science logged information message.')

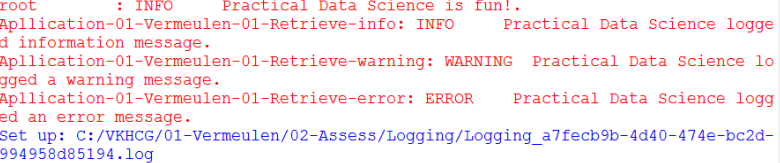
if sLevel == 'warning':

logger.warning('Practical Data Science logged a warning message.')

if sLevel == 'error':

logger.error('Practical Data Science logged an error message.')

**Output:**

****

**Practical-3**

1. **Program to retrieve different attributes of data.**

**Code:**

import sys

import os

import pandas as pd

Base='E:/MSC Practicals/Data science/SUPERSTEPS/'

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('Fixed Data Set with ID')

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

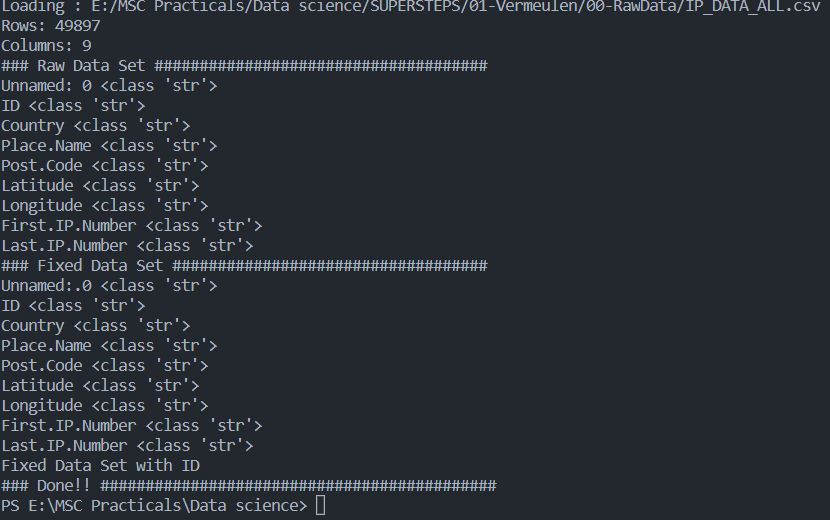
IP\_DATA\_ALL\_with\_ID.index.names = ['RowID']

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

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

print('### Done!! ############################################')

**Output:**

****

**Vermeulen PLC**

**Code:**

import sys

import os

import pandas as pd

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

def haversine(lat1,lon1,lat2,lon2,stype):

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

dlon=lon2-lon1

dlat=lat2-lat1

a=sin(dlat/2)\*\*2+cos(lat1)\*cos(lat2)\*sin(dlon/2)\*\*2

c=2\*asin(sqrt(a))

if stype =='km':

r=6371

else:

r=3956

d=round(c\*r,3)

return d

Base='E:/MSC Practicals/Data science/SUPERSTEPS/'

sFileName=Base+'01-Vermeulen/00-RawData/IP\_DATA\_CORE.csv'

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

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

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

IP\_DATA = IP\_DATA\_ALL.drop\_duplicates(subset=None, keep='first', inplace=False)

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

IP\_DATA1 = IP\_DATA

IP\_DATA1.insert(0, 'K', 1)

IP\_DATA2 = IP\_DATA1

print(IP\_DATA1.shape)

IP\_CROSS=pd.merge(right=IP\_DATA1,left=IP\_DATA2,on='K')

IP\_CROSS.drop('K', axis=1, inplace=True)

IP\_CROSS.rename(columns={'Longitude\_x': 'Longitude\_from', 'Longitude\_y': 'Longitude\_to'},

inplace=True)

IP\_CROSS.rename(columns={'Latitude\_x': 'Latitude\_from', 'Latitude\_y': 'Latitude\_to'},

inplace=True)

IP\_CROSS.rename(columns={'Place\_Name\_x': 'Place\_Name\_from', 'Place\_Name\_y':

'Place\_Name\_to'}, inplace=True)

IP\_CROSS.rename(columns={'Country\_x': 'Country\_from', 'Country\_y': 'Country\_to'},

inplace=True)

IP\_CROSS['DistanceBetweenKilometers'] = IP\_CROSS.apply(lambda row:

haversine(

row['Longitude\_from'],

row['Latitude\_from'],

row['Longitude\_to'],

row['Latitude\_to'],

'km')

,axis=1)

IP\_CROSS['DistanceBetweenMiles'] = IP\_CROSS.apply(lambda row:

haversine(

row['Longitude\_from'],

row['Latitude\_from'],

row['Longitude\_to'],

row['Latitude\_to'],

'miles')

,axis=1)

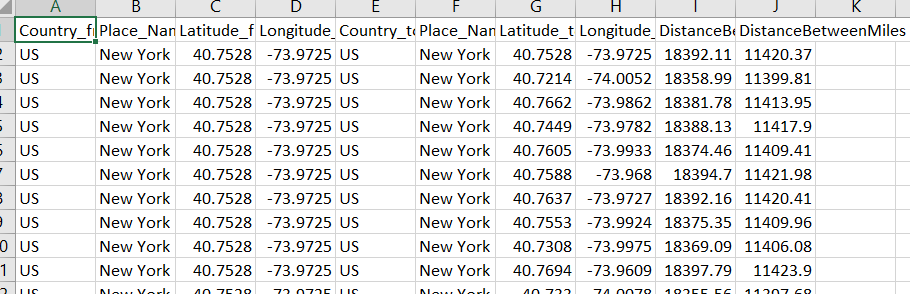
print(IP\_CROSS.shape)

sFileName2=sFileDir + '/Retrieve\_IP\_Routing.csv'

IP\_CROSS.to\_csv(sFileName2, index = False, encoding="latin-1")

print('### Done!! ############################################')

**Output:**

****

**Building a Diagram for the Scheduling of Jobs**

**Code:**

import sys

import os

import pandas as pd

InputFileName='IP\_DATA\_CORE.csv'

OutputFileName='Retrieve\_Router\_Location.csv'

Base='E:/MSC Practicals/Data science/SUPERSTEPS/'

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)

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

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

ROUTERLOC = IP\_DATA\_ALL.drop\_duplicates(subset=None, keep='first', inplace=False)

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

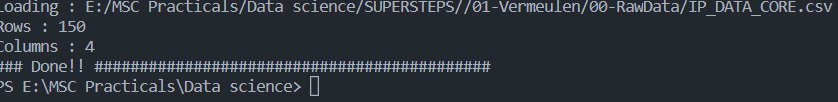
print('Columns :',ROUTERLOC.shape[1])

sFileName2=sFileDir + '/' + OutputFileName

ROUTERLOC.to\_csv(sFileName2, index = False, encoding="latin-1")

print('### Done!! ############################################')

**Output:**

****

**Krennwallner AG**

**Picking Content for Billboards**

**Code:**

import sys

import os

import pandas as pd

InputFileName='DE\_Billboard\_Locations.csv'

OutputFileName='Retrieve\_DE\_Billboard\_Locations.csv'

Company='02-Krennwallner'

Base='E:/MSC Practicals/Data science/SUPERSTEPS/'

sFileName=Base + '/' + Company + '/00-RawData/' + InputFileName

print('Loading :',sFileName)

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

usecols=['Country','PlaceName','Latitude','Longitude'])

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

sFileDir=Base + '/' + Company + '/01-Retrieve/01-EDS/02-Python'

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

ROUTERLOC = IP\_DATA\_ALL.drop\_duplicates(subset=None, keep='first', inplace=False)

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

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

sFileName2=sFileDir + '/' + OutputFileName

ROUTERLOC.to\_csv(sFileName2, index = False)

**Output:**

****

**Understanding Your Online Visitor Data**

**Code:**

import sys

import os

import pandas as pd

import gzip as gz

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

InputFileName='IP\_DATA\_ALL.csv'

OutputFileName='Retrieve\_Online\_Visitor'

CompanyIn= '01-Vermeulen'

CompanyOut= '02-Krennwallner'

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

Base='C:/VKHCG'

print('################################')

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

print('################################')

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

Base='C:/VKHCG'

sFileName=Base + '/' + CompanyIn + '/00-RawData/' + InputFileName

print('Loading :',sFileName)

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

usecols=['Country','Place Name','Latitude','Longitude','First IP Number','Last IP Number'])

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

IP\_DATA\_ALL.rename(columns={'First IP Number': 'First\_IP\_Number'}, inplace=True)

IP\_DATA\_ALL.rename(columns={'Last IP Number': 'Last\_IP\_Number'}, inplace=True)

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

sFileDir=Base + '/' + CompanyOut + '/01-Retrieve/01-EDS/02-Python'

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

visitordata = IP\_DATA\_ALL.drop\_duplicates(subset=None, keep='first', inplace=False)

visitordata10=visitordata.head(10)

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

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

print('Export CSV')

sFileName2=sFileDir + '/' + OutputFileName + '.csv'

visitordata.to\_csv(sFileName2, index = False)

print('Store All:',sFileName2)

sFileName3=sFileDir + '/' + OutputFileName + '\_10.csv'

visitordata10.to\_csv(sFileName3, index = False)

print('Store 10:',sFileName3)

for z in ['gzip', 'bz2', 'xz']:

if z == 'gzip':

sFileName4=sFileName2 + '.gz'

else:

sFileName4=sFileName2 + '.' + z

visitordata.to\_csv(sFileName4, index = False, compression=z)

print('Store :',sFileName4)

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

print('Export JSON')

for sOrient in ['split','records','index', 'columns','values','table']:

sFileName2=sFileDir + '/' + OutputFileName + '\_' + sOrient + '.json'

visitordata.to\_json(sFileName2,orient=sOrient,force\_ascii=True)

print('Store All:',sFileName2)

sFileName3=sFileDir + '/' + OutputFileName + '\_10\_' + sOrient + '.json'

visitordata10.to\_json(sFileName3,orient=sOrient,force\_ascii=True)

print('Store 10:',sFileName3)

sFileName4=sFileName2 + '.gz'

file\_in = open(sFileName2, 'rb')

file\_out = gz.open(sFileName4, 'wb')

file\_out.writelines(file\_in)

file\_in.close()

file\_out.close()

print('Store GZIP All:',sFileName4)

sFileName5=sFileDir + '/' + OutputFileName + '\_' + sOrient + '\_UnGZip.json'

file\_in = gz.open(sFileName4, 'rb')

file\_out = open(sFileName5, 'wb')

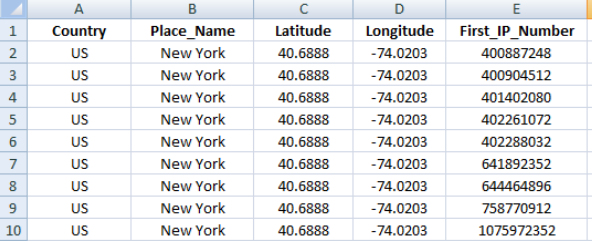
file\_out.writelines(file\_in)

file\_in.close()

file\_out.close()

print('Store UnGZIP All:',sFileName5)

**Output:**



**XML processing**

**Code:**

import sys

import os

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.fromstring(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)

InputFileName='IP\_DATA\_ALL.csv'

OutputFileName='Retrieve\_Online\_Visitor.xml'

CompanyIn= '01-Vermeulen'

CompanyOut= '02-Krennwallner'

if sys.platform == 'linux':

Base=os.path.expanduser('~') + '/VKHCG'

else:

Base='E:/MSC Practicals/Data science/SUPERSTEPS/'

sFileName=Base + '/' + CompanyIn + '/00-RawData/' + InputFileName

print('Loading :',sFileName)

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

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

IP\_DATA\_ALL.rename(columns={'First IP Number': 'First\_IP\_Number'}, inplace=True)

IP\_DATA\_ALL.rename(columns={'Last IP Number': 'Last\_IP\_Number'}, inplace=True)

IP\_DATA\_ALL.rename(columns={'Post Code': 'Post\_Code'}, inplace=True)

sFileDir=Base + '/' + CompanyOut + '/01-Retrieve/01-EDS/02-Python'

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

visitordata = IP\_DATA\_ALL.head(10000)

print('Original Subset Data Frame')

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

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

print(visitordata)

print('Export XML')

sXML=df2xml(visitordata)

sFileName=sFileDir + '/' + OutputFileName

file\_out = open(sFileName, 'wb')

file\_out.write(sXML)

file\_out.close()

print('Store XML:',sFileName)

xml\_data = open(sFileName).read()

unxmlrawdata=xml2df(xml\_data)

print('Raw XML Data Frame')

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

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

print(unxmlrawdata)

unxmldata = unxmlrawdata.drop\_duplicates(subset=None, keep='first', inplace=False)

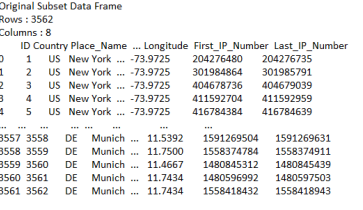
print('Deduplicated XML Data Frame')

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

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

print(unxmldata)

**Output:**

****

**Hillman Ltd**

**Code:**

import pandas as pd

inputFileName='Incoter\_2010.csv'

outputFileName='./Output/hillman\_retreive.csv'

IncoTerm='EXW'

IncotermGrid=pd.read\_csv(inputFileName,header=0,low\_memory=False)

IncotermRule=IncotermGrid[IncotermGrid.Shipping\_Term == IncoTerm]

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

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

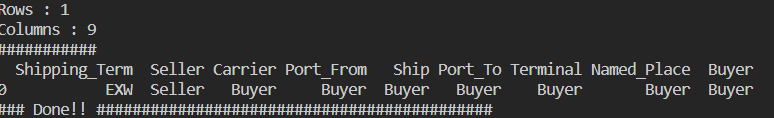
print('###########')

print(IncotermRule)

IncotermRule.to\_csv(outputFileName, index = False)

print('### Done!! ############################################')

**Output:**



**FCA—Free Carrier (Named Place of Delivery)**

**Code:**

import pandas as pd

inputFileName='Incoter\_2010.csv'

outputFileName='./Output/hillman\_retreive\_2.csv'

IncoTerm='FCA'

IncotermGrid=pd.read\_csv(inputFileName,header=0,low\_memory=False)

IncotermRule=IncotermGrid[IncotermGrid.Shipping\_Term == IncoTerm]

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

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

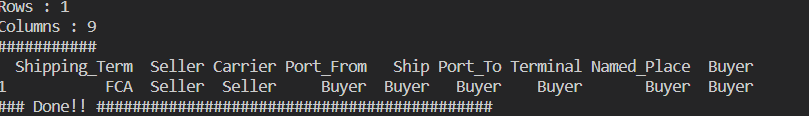
print('###########')

print(IncotermRule)

IncotermRule.to\_csv(outputFileName, index = False)

print('### Done!! ############################################')

**Output:**



**Possible Shipping Routes**

**Code:**

import sys

import os

import pandas as pd

ContainerFileName='Retrieve\_Container.csv'

BoxFileName='Retrieve\_Box.csv'

ProductFileName='Retrieve\_Product.csv'

Company='03-Hillman'

print('################################')

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

print('################################')

sFileDir=Base + '/' + Company + '/01-Retrieve/01-EDS/02-Python'

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)containerLength=range(1,21)

containerWidth=range(1,10)

containerHeigth=range(1,6)

containerStep=1

c=0

for l in containerLength:

for w in containerWidth:

for h in containerHeigth:

containerVolume=(l/containerStep)\*(w/containerStep)\*(h/containerStep)

c=c+1

ContainerLine=[('ShipType', ['Container']),

('UnitNumber', ('C'+format(c,"06d"))),

('Length',(format(round(l,3),".4f"))),

('Width',(format(round(w,3),".4f"))),

('Height',(format(round(h,3),".4f"))),

('ContainerVolume',(format(round(containerVolume,6),".6f")))]

if c==1:

ContainerFrame = pd.DataFrame.from\_dict(ContainerLine)

else:

ContainerRow = pd.DataFrame.from\_dict(ContainerLine)

ContainerFrame = ContainerFrame.append(ContainerRow)

ContainerFrame.index.name = 'IDNumber'print('################')

print('## Container')

print('################')

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

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

print('################')

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

sFileContainerName=sFileDir + '/' + ContainerFileName

ContainerFrame.to\_csv(sFileContainerName, index = False)

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

## Create valid Boxes with packing foam

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

boxLength=range(1,21)

boxWidth=range(1,21)

boxHeigth=range(1,21)

packThick=range(0,6)

boxStep=10

b=0

for l in boxLength:

for w in boxWidth:

for h in boxHeigth:

for t in packThick:

boxVolume=round((l/boxStep)\*(w/boxStep)\*(h/boxStep),6)

productVolume=round(((l-t)/boxStep)\*((w-t)/boxStep)\*((h-t)/boxStep),6)

if productVolume > 0:

b=b+1

BoxLine=[('ShipType', ['Box']),

('UnitNumber', ('B'+format(b,"06d"))),

('Length',(format(round(l/10,6),".6f"))),

('Width',(format(round(w/10,6),".6f"))),

('Height',(format(round(h/10,6),".6f"))),

('Thickness',(format(round(t/5,6),".6f"))),

('BoxVolume',(format(round(boxVolume,9),".9f"))),

('ProductVolume',(format(round(productVolume,9),".9f")))]

if b==1:

BoxFrame = pd.DataFrame.from\_dict(BoxLine)

else:

BoxRow = pd.DataFrame.from\_dict(BoxLine)

BoxFrame = BoxFrame.append(BoxRow)

BoxFrame.index.name = 'IDNumber'

print('#################')

print('## Box')

print('#################')

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

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

print('#################')

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

sFileBoxName=sFileDir + '/' + BoxFileName

BoxFrame.to\_csv(sFileBoxName, index = False)

productLength=range(1,21)

productWidth=range(1,21)

productHeigth=range(1,21)

productStep=10

p=0

for l in productLength:

for w in productWidth:

for h in productHeigth:

productVolume=round((l/productStep)\*(w/productStep)\*(h/productStep),6)

if productVolume > 0:

p=p+1

ProductLine=[('ShipType', ['Product']),

('UnitNumber', ('P'+format(p,"06d"))),

('Length',(format(round(l/10,6),".6f"))),

('Width',(format(round(w/10,6),".6f"))),

('Height',(format(round(h/10,6),".6f"))),

('ProductVolume',(format(round(productVolume,9),".9f")))]

if p==1:

ProductFrame = pd.DataFrame.from\_dict(ProductLine)

else:

ProductRow = pd.DataFrame.from\_dict(ProductLine)

ProductFrame = ProductFrame.append(ProductRow)

BoxFrame.index.name = 'IDNumber'

print('#################')

print('## Product')

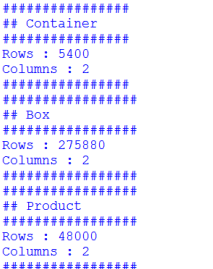
print('#################')

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

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

print('#################')

**Output:**

****

**Create a Delivery Route**

**Code:**

import os

import sys

import pandas as pd

from geopy.distance import vincenty

InputFileName='GB\_Postcode\_Warehouse.csv'

OutputFileName='Retrieve\_GB\_Warehouse.csv'

Company='03-Hillman'

sFileDir=Base + '/' + Company + '/01-Retrieve/01-EDS/02-Python'

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)sFileName=Base + '/' + Company + '/00-RawData/' + InputFileName

print('###########')

print('Loading :',sFileName)

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

WarehouseClean=Warehouse[Warehouse.latitude != 0]

WarehouseGood=WarehouseClean[WarehouseClean.longitude != 0]

WarehouseGood.drop\_duplicates(subset='postcode', keep='first', inplace=True)

WarehouseGood.sort\_values(by='postcode', ascending=1)

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

sFileName=sFileDir + '/' + OutputFileName

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

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

WarehouseLoop = WarehouseGood.head(20)

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

print('Run :',i,' =======>>>>>>>>>>',WarehouseLoop['postcode'][i])

WarehouseHold = WarehouseGood.head(10000)

WarehouseHold['Transaction']=WarehouseHold.apply(lambda row:

'WH-to-WH'

,axis=1)

OutputLoopName='Retrieve\_Route\_' + 'WH-' + WarehouseLoop['postcode'][i] + '\_Route.csv'

WarehouseHold['Seller']=WarehouseHold.apply(lambda row:

'WH-' + WarehouseLoop['postcode'][i]

,axis=1)

WarehouseHold['Seller\_Latitude']=WarehouseHold.apply(lambda row:

WarehouseHold['latitude'][i],axis=1)

WarehouseHold['Seller\_Longitude']=WarehouseHold.apply(lambda row:

WarehouseLoop['longitude'][i],axis=1)

WarehouseHold['Buyer']=WarehouseHold.apply(lambda row:

'WH-' + row['postcode'],axis=1)

WarehouseHold['Buyer\_Latitude']=WarehouseHold.apply(lambda row:

row['latitude'],axis=1)

WarehouseHold['Buyer\_Longitude']=WarehouseHold.apply(lambda row:

row['longitude'],axis=1)

WarehouseHold['Distance']=WarehouseHold.apply(lambda row: round(

vincenty((WarehouseLoop['latitude'][i],WarehouseLoop['longitude'][i]),

(row['latitude'],row['longitude'])).miles,6),axis=1)

WarehouseHold.drop('id', axis=1, inplace=True)

WarehouseHold.drop('postcode', axis=1, inplace=True)

WarehouseHold.drop('latitude', axis=1, inplace=True)

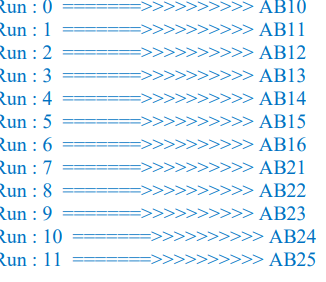
WarehouseHold.drop('longitude', axis=1, inplace=True)

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

sFileLoopName=sFileDir + '/' + OutputLoopName

WarehouseHold.to\_csv(sFileLoopName, index = False)

**Output:**

****

**Connecting to other Data Sources**

**SQLite**

**Code:**

import sqlite3 as sq

import pandas as pd

Base='E:/MSC Practicals/Data science/SUPERSTEPS'

sDatabaseName=Base + '/01-Vermeulen/00-RawData/SQLite/vermeulen.db'

conn = sq.connect(sDatabaseName)

sFileName=Base+'/01-Vermeulen/01-Retrieve/01-EDS/02-Python/Retrieve\_IP\_DATA.csv'

print('Loading :',sFileName)

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

IP\_DATA\_ALL\_FIX.index.names = ['RowIDCSV']

sTable='IP\_DATA\_ALL'

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

IP\_DATA\_ALL\_FIX.to\_sql(sTable, conn, if\_exists="replace")

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

TestData=pd.read\_sql\_query("select \* from IP\_DATA\_ALL;", conn)

print('################')

print('## Data Values')

print('################')

print(TestData)

print('################')

print('## Data Profile')

print('################')

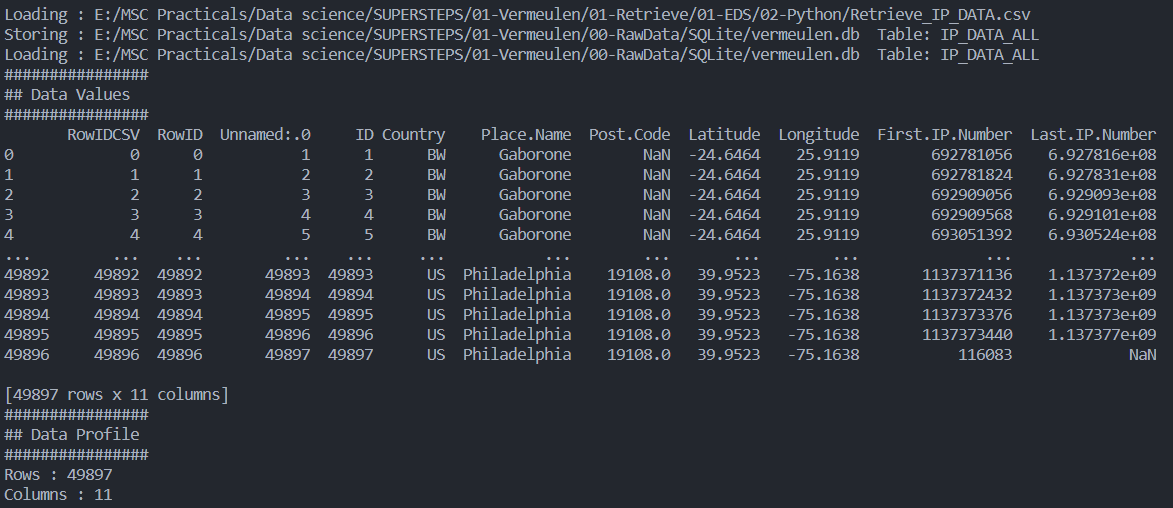
print('Rows :',TestData.shape[0])

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

print('################')

print('### Done!! ############################################')

**Output:**

****

**Microsoft Excel**

**Code:**

import pandas as pd

currencyRawData=pd.read\_excel('./Country\_Currency.xlsx')

sColumns=['Country or territory','Currency','ISO-4217']

currencyData=currencyRawData[sColumns]

currencyData.rename(columns={'Country or territory':'Country','ISO-4217':'CurrencyCode'},inplace=True)

currencyData.dropna(subset=['Currency'],inplace=True)

currencyData['Country']=currencyData['Country'].map(lambda x:x.strip())

currencyData['Currency']=currencyData['Currency'].map(lambda x:x.strip())

currencyData['CurrencyCode']=currencyData['CurrencyCode'].map(lambda x:x.strip())

print(currencyData)

print("Data retrieve from excel file")

outputFileName='./Output/Retrieve-currency.csv'

currencyData.to\_csv(outputFileName,index=False)

**Output:**

****

**Practical-4**

1. **Build the time hub, links, and satellites.**

**Code:**

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

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

if sys.platform == 'linux':

Base=os.path.expanduser('~') + '/VKHCG'

else:

Base='C:/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\_items(TimeLine)

else:

TimeRow = pd.DataFrame.from\_items(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\_items(TimeZoneLine)

else:

TimeZoneRow = pd.DataFrame.from\_items(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('################')

**Golden Nominal**

Code:

import sys

import os

import sqlite3 as sq

import pandas as pd

from pandas.io import sql

from datetime import datetime, timedelta

from pytz import timezone, all\_timezones

from random import randint

import uuid

if sys.platform == 'linux':

Base=os.path.expanduser('~') + '/VKHCG'

else:

Base='C:/VKHCG'

print('################################')

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

print('################################')

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

Company='04-Clark'

sInputFileName='02-Assess/01-EDS/02-Python/Assess\_People.csv'

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

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

if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir)

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

sDatabaseName=sDataBaseDir + '/clark.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)

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

### Import Female Data

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

sFileName=Base + '/' + Company + '/' + sInputFileName

print('################################')

print('Loading :',sFileName)

print('################################')

print(sFileName)

RawData=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1")

RawData.drop\_duplicates(subset=None, keep='first', inplace=True)

start\_date = datetime(1900,1,1,0,0,0)

start\_date\_utc=start\_date.replace(tzinfo=timezone('UTC'))

HoursBirth=100\*365\*24

RawData['BirthDateUTC']=RawData.apply(lambda row:

(start\_date\_utc + timedelta(hours=randint(0, HoursBirth)))

,axis=1)

zonemax=len(all\_timezones)-1

RawData['TimeZone']=RawData.apply(lambda row:

(all\_timezones[randint(0, zonemax)])

,axis=1)

RawData['BirthDateISO']=RawData.apply(lambda row:

row["BirthDateUTC"].astimezone(timezone(row['TimeZone']))

,axis=1)

RawData['BirthDateKey']=RawData.apply(lambda row:

row["BirthDateUTC"].strftime("%Y-%m-%d %H:%M:%S")

,axis=1)

RawData['BirthDate']=RawData.apply(lambda row:

row["BirthDateISO"].strftime("%Y-%m-%d %H:%M:%S")

,axis=1)

RawData['PersonID']=RawData.apply(lambda row:

str(uuid.uuid4())

,axis=1)Data=RawData.copy()

Data.drop('BirthDateUTC', axis=1,inplace=True)

Data.drop('BirthDateISO', axis=1,inplace=True)

indexed\_data = Data.set\_index(['PersonID'])

print('################################')

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

print('################')

sTable='Process\_Person'

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

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

print('################')

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

PersonHubRaw=Data[['PersonID','FirstName','SecondName','LastName','BirthDateKey']]

PersonHubRaw['PersonHubID']=RawData.apply(lambda row:

str(uuid.uuid4())

,axis=1)

PersonHub=PersonHubRaw.drop\_duplicates(subset=None, \

keep='first',\

inplace=False)

indexed\_PersonHub = PersonHub.set\_index(['PersonHubID'])

sTable = 'Hub-Person'

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

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

PersonSatelliteGenderRaw=Data[['PersonID','FirstName','SecondName','LastName'\

,'BirthDateKey','Gender']]

PersonSatelliteGenderRaw['PersonSatelliteID']=RawData.apply(lambda row:

str(uuid.uuid4())

,axis=1)

PersonSatelliteGender=PersonSatelliteGenderRaw.drop\_duplicates(subset=None, \

keep='first', \

inplace=False)

indexed\_PersonSatelliteGender = PersonSatelliteGender.set\_index(['PersonSatelliteID'])

sTable = 'Satellite-Person-Gender'

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

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

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

PersonSatelliteBirthdayRaw=Data[['PersonID','FirstName','SecondName','LastName',\

'BirthDateKey','TimeZone','BirthDate']]

PersonSatelliteBirthdayRaw['PersonSatelliteID']=RawData.apply(lambda row:

str(uuid.uuid4())

,axis=1)

PersonSatelliteBirthday=PersonSatelliteBirthdayRaw.drop\_duplicates(subset=None, \

keep='first',\

inplace=False)

indexed\_PersonSatelliteBirthday = PersonSatelliteBirthday.set\_index(['PersonSatelliteID'])

sTable = 'Satellite-Person-Names'

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

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

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

sFileDir=Base + '/' + Company + '/03-Process/01-EDS/02-Python'

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

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

sOutputFileName = sTable + '.csv'

sFileName=sFileDir + '/' + sOutputFileName

print('################################')

print('Storing :', sFileName)

print('################################')

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

print('################################')

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

print('################')

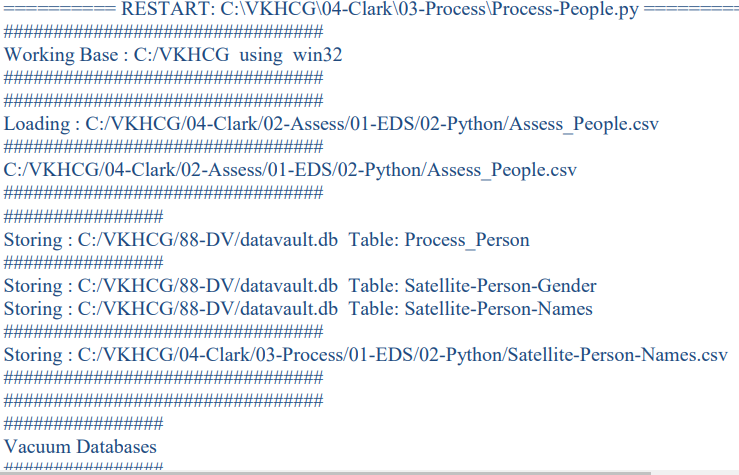
print('Vacuum Databases')

sSQL="VACUUM;"

sql.execute(sSQL,conn1)

sql.execute(sSQL,conn2)

**Output:**

****

**Vehicles**

**Code:**

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

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

if sys.platform == 'linux':

Base=os.path.expanduser('~') + '/VKHCG'

else:

Base='C:/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)

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")

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")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)

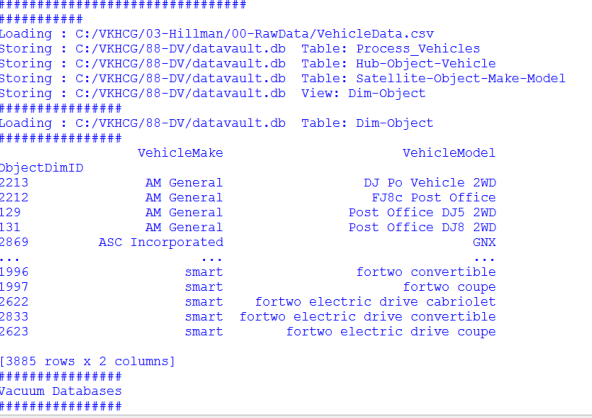
print('################')

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

conn1.close()

conn2.close()

**Output:**

****

**Human-Environment InteractionCode:**

import sys

import os

import pandas as pd

import sqlite3 as sq

from pandas.io import sql

import uuid

Base='C:/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\_items(LocationLine)

else:

LocationRow = pd.DataFrame.from\_items(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('################')

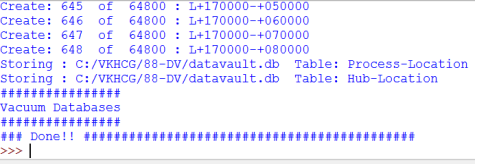
print('Vacuum Databases')

sSQL="VACUUM;"

sql.execute(sSQL,conn1)

sql.execute(sSQL,conn2)

Output:



**Forecasting**

**Code:**

import sys

import os

import sqlite3 as sq

import quandl

import pandas as pd

Base='C:/VKHCG'

print('################################')

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

print('################################')

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

Company='04-Clark'

sInputFileName='00-RawData/VKHCG\_Shares.csv'

sOutputFileName='Shares.csv'

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

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

if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir)

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

sFileDir1=Base + '/' + Company + '/01-Retrieve/01-EDS/02-Python'

if not os.path.exists(sFileDir1):

os.makedirs(sFileDir1)

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

sFileDir2=Base + '/' + Company + '/02-Assess/01-EDS/02-Python'

if not os.path.exists(sFileDir2):

os.makedirs(sFileDir2)

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

sFileDir3=Base + '/' + Company + '/03-Process/01-EDS/02-Python'

if not os.path.exists(sFileDir3):

os.makedirs(sFileDir3)

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

sDatabaseName=sDataBaseDir + '/clark.db'

conn = sq.connect(sDatabaseName)

sFileName=Base + '/' + Company + '/' + sInputFileName

print('################################')

print('Loading :',sFileName)

print('################################')

RawData=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1")

RawData.drop\_duplicates(subset=None, keep='first', inplace=True)

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

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

print('################')

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

sFileName=sFileDir1 + '/Retrieve\_' + sOutputFileName

print('################################')

print('Storing :', sFileName)

print('################################')

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

print('################################')

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

sFileName=sFileDir2 + '/Assess\_' + sOutputFileName

print('################################')

print('Storing :', sFileName)

print('################################')

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

print('################################')

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

sFileName=sFileDir3 + '/Process\_' + sOutputFileName

print('################################')

print('Storing :', sFileName)

print('################################')

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

print('################################')

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

### Import Shares Data Details

nShares=RawData.shape[0]

#nShares=6

for sShare in range(nShares):

sShareName=str(RawData['Shares'][sShare])

ShareData = quandl.get(sShareName)

UnitsOwn=RawData['Units'][sShare]

ShareData['UnitsOwn']=ShareData.apply(lambda row:(UnitsOwn),axis=1)

ShareData['ShareCode']=ShareData.apply(lambda row:(sShareName),axis=1)

print('################')

print('Share :',sShareName)

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

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

print('################')

sTable=str(RawData['sTable'][sShare])

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

ShareData.to\_sql(sTable, conn, if\_exists="replace")

print('################')

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

sOutputFileName = sTable.replace("/","-") + '.csv'

sFileName=sFileDir1 + '/Retrieve\_' + sOutputFileName

print('################################')

print('Storing :', sFileName)

print('################################')

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

print('################################')

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

sOutputFileName = sTable.replace("/","-") + '.csv'

sFileName=sFileDir2 + '/Assess\_' + sOutputFileName

print('################################')

print('Storing :', sFileName)

print('################################')

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

print('################################')

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

sOutputFileName = sTable.replace("/","-") + '.csv'

sFileName=sFileDir3 + '/Process\_' + sOutputFileName

print('################################')

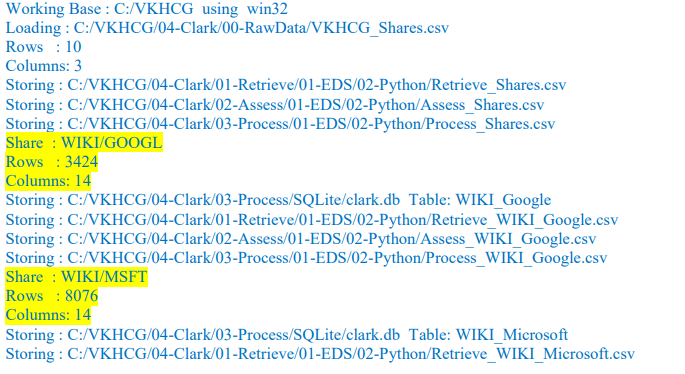
print('Storing :', sFileName)

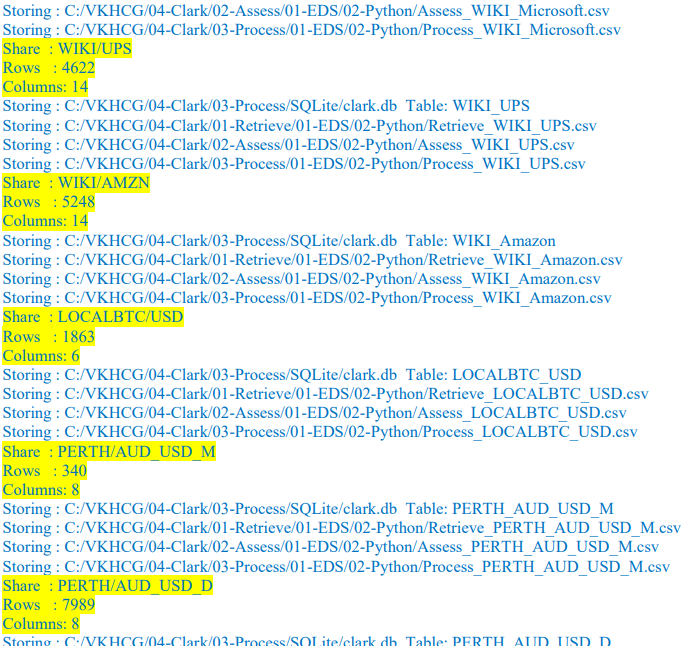
print('################################')

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

print('################################')

Output:





**Practical-5**

**Horizontal Style**

**Code:**

import sys

import os

import pandas as pd

import sqlite3 as sqBase='C:/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)

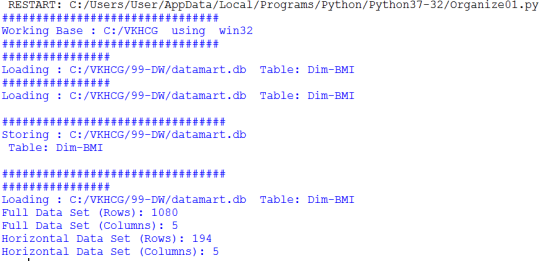
print('Full Data Set (Rows):', PersonFrame0.shape[0])

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:**



**Vertical Style**

**Code:**

import sys

import os

import pandas as pd

import sqlite3 as sq

Base='C:/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)

print('################################')

sSQL="SELECT \

Height,\

Weight,\

Indicator\

FROM [Dim-BMI];"

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

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

DimPerson=PersonFrame1

DimPersonIndex=DimPerson.set\_index(['Indicator'],inplace=False)

sTable = 'Dim-BMI-Vertical'

print('\n#################################')

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

print('\n#################################')

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

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

print('################')

sTable = 'Dim-BMI-Vertical'

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

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

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

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

print('################################')

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

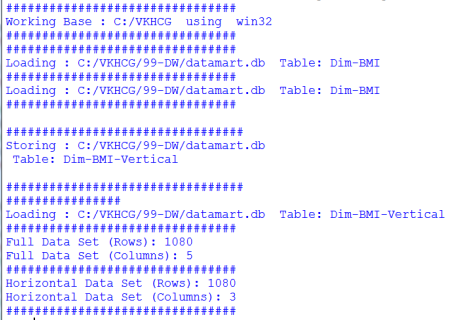
print('Full Data Set (Columns):', PersonFrame0.shape[1])

print('################################')

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

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

**Output:**

****

**Island style**

**Code:**

import sys

import os

import pandas as pd

import sqlite3 as sqBase='C:/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 \

Height,\

Weight,\

Indicator\

FROM [Dim-BMI]\

WHERE Indicator > 2\

ORDER BY \

Height,\

Weight;"

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

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

DimPerson=PersonFrame1

DimPersonIndex=DimPerson.set\_index(['Indicator'],inplace=False)

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

sTable = 'Dim-BMI-Vertical'

print('\n#################################')

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

print('\n#################################')

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

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

print('################################')

sTable = 'Dim-BMI-Vertical'

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

print('################################')

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

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

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

print('################################')

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

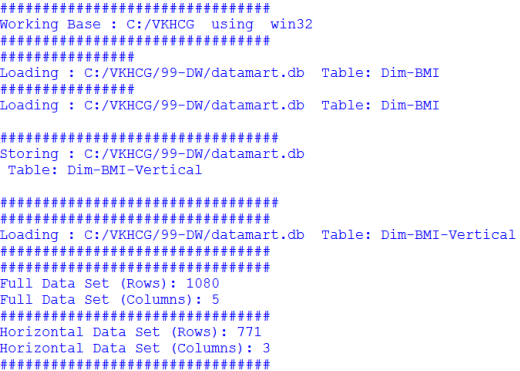
print('Full Data Set (Columns):', PersonFrame0.shape[1])

print('################################')

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

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

**Output:**

****

**Secure Vault Style**

**Code:**

import sys

import os

import pandas as pd

import sqlite3 as sq

Base='C:/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 \

Height,\

Weight,\

Indicator,\

CASE Indicator\

WHEN 1 THEN 'Pip'\

WHEN 2 THEN 'Norman'\

WHEN 3 THEN 'Grant'\

ELSE 'Sam'\

END AS Name\

FROM [Dim-BMI]\

WHERE Indicator > 2\

ORDER BY \

Height,\

Weight;"

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

DimPerson=PersonFrame1

DimPersonIndex=DimPerson.set\_index(['Indicator'],inplace=False)

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

sTable = 'Dim-BMI-Secure'

print('\n#################################')

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

print('\n#################################')

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

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

print('################################')

sTable = 'Dim-BMI-Secure'

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

print('################################')

sSQL="SELECT \* FROM [Dim-BMI-Secure] WHERE Name = 'Sam';"

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

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

print('################################')

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

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

print('################################')

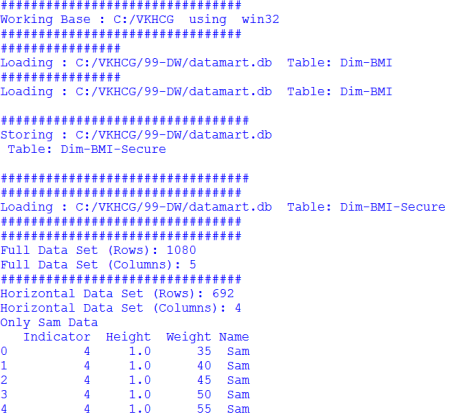
print('Horizontal Data Set (Rows):', PersonFrame2.shape[0])

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

print('Only Sam Data')

print(PersonFrame2.head())

**Output:**

****

**Association Rule Mining**

**Code:**

import sys

import os

import pandas as pd

from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules

Base='C:/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)

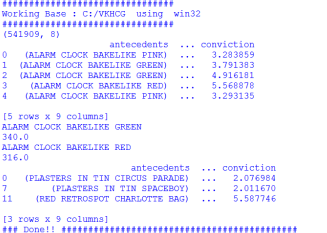
frequent\_itemsets2 = apriori(basket\_sets2, min\_support=0.05, use\_colnames=True)

rules2 = association\_rules(frequent\_itemsets2, metric="lift", min\_threshold=1)

print(rules2[ (rules2['lift'] >= 4) &

(rules2['confidence'] >= 0.5)])

**Output:**



**Create a Network Routing Diagram**

**Code:**

import sys

import os

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt

pd.options.mode.chained\_assignment = None

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

Base='C:/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()

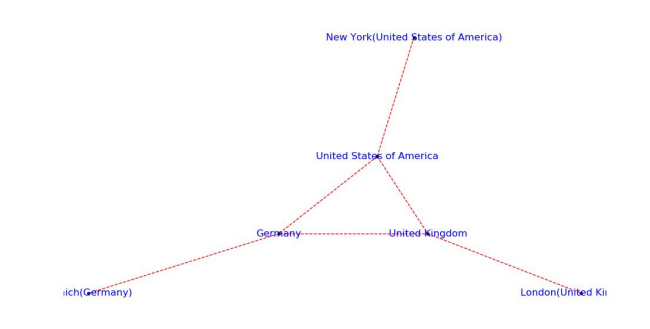
################################################################

print('################################')

print('### Done!! #####################')

print('################################'

**Output:**

****

**Picking Contents for Billboards**

**Code:**

import sys

import os

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt

import numpy as np

pd.options.mode.chained\_assignment = None

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

Base='C:/VKHCG'

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

print('################################')

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

print('################################')

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

sInputFileName='02-Assess/01-EDS/02-Python/Assess-DE-Billboard-Visitor.csv'

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

sOutputFileName1='05-Organise/01-EDS/02-Python/Organise-Billboards.gml'

sOutputFileName2='05-Organise/01-EDS/02-Python/Organise-Billboards.png'

Company='02-Krennwallner'

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

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

### Import Company Data

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

sFileName=Base + '/' + Company + '/' + sInputFileName

print('################################')

print('Loading :',sFileName)

print('################################')

BillboardDataRaw=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1")

print('################################')

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

print(BillboardDataRaw.head())

print(BillboardDataRaw.shape)

BillboardData=BillboardDataRaw

sSample=list(np.random.choice(BillboardData.shape[0],20))

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

G=nx.Graph()

for i in sSample:

for j in sSample:

Node0=BillboardData['BillboardPlaceName'][i] + '('+ BillboardData['BillboardCountry'][i] + ')'

Node1=BillboardData['BillboardPlaceName'][j] + '('+ BillboardData['BillboardCountry'][i] + ')'

if Node0 != Node1:

G.add\_edge(Node0,Node1)

for i in sSample:

Node0=BillboardData['BillboardPlaceName'][i] + '('+ BillboardData['VisitorPlaceName'][i] + ')'

Node1=BillboardData['BillboardPlaceName'][i] + '('+ BillboardData['VisitorCountry'][i] + ')'

if Node0 != Node1:

G.add\_edge(Node0,Node1)

print('Nodes:', G.number\_of\_nodes())

print('Edges:', G.number\_of\_edges())

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

sFileName=Base + '/02-Krennwallner/' + sOutputFileName1

print('################################')

print('Storing :',sFileName)

print('################################')

nx.write\_gml(G, sFileName)

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

sFileName=Base + '/02-Krennwallner/' + sOutputFileName2

print('################################')

print('Storing Graph Image:',sFileName)

print('################################')

plt.figure(figsize=(15, 15))

pos=nx.circular\_layout(G,dim=2)

nx.draw\_networkx\_nodes(G,pos, node\_color='k', node\_size=150, alpha=0.8)

nx.draw\_networkx\_edges(G, pos,edge\_color='r', arrows=False, style='solid')

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()

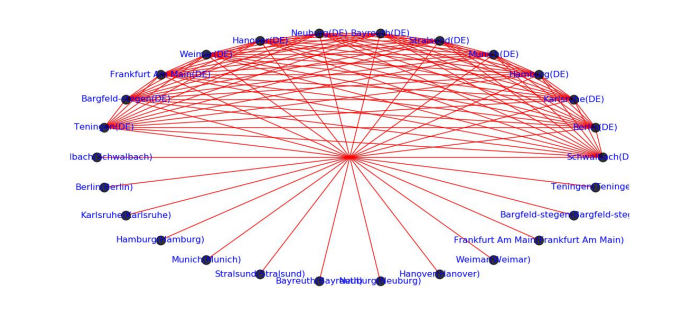
################################################################

print('################################')

print('### Done!! #####################')

print('################################')

**Output:**

****

**Create a Delivery RouteCode:**

import sys

import os

import pandas as pd

Base='C:/VKHCG'

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

print('################################')

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

print('################################')

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

sInputFileName='02-Assess/01-EDS/02-Python/Assess\_Shipping\_Routes.txt'

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

sOutputFileName='05-Organise/01-EDS/02-Python/Organise-Routes.csv'

Company='03-Hillman

sFileName=Base + '/' + Company + '/' + sInputFileName

print('################################')

print('Loading :',sFileName)

print('################################')

RouteDataRaw=pd.read\_csv(sFileName,header=0,low\_memory=False, sep='|', encoding="latin-1")

print('################################')

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

RouteStart=RouteDataRaw[RouteDataRaw['StartAt']=='WH-KA13']

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

RouteDistance=RouteStart[RouteStart['Cost']=='DistanceMiles']

RouteDistance=RouteDistance.sort\_values(by=['Measure'], ascending=False)

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

RouteMax=RouteStart["Measure"].max()

RouteMaxCost=round((((RouteMax/1000)\*1.5\*2)),2)

print('################################')

print('Maximum (£) per day:')

print(RouteMaxCost)

print('################################')

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

RouteMean=RouteStart["Measure"].mean()

RouteMeanMonth=round((((RouteMean/1000)\*2\*30)),6)

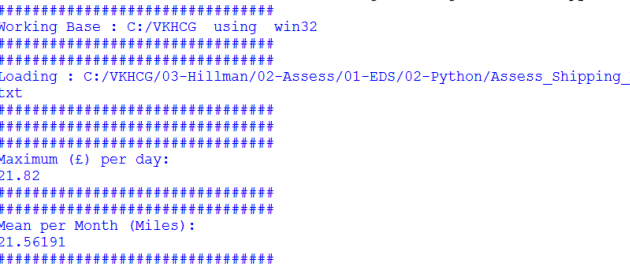
print('################################')

print('Mean per Month (Miles):')

print(RouteMeanMonth)

print('################################')

**Output:**

****

**Clark Ltd**

**Code:**

import sys

import os

import pandas as pd

import sqlite3 as sq

import re

Base='C:/VKHCG'

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

print('################################')

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

print('################################')

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

sInputFileName='03-Process/01-EDS/02-Python/Process\_ExchangeRates.csv'

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

sOutputFileName='05-Organise/01-EDS/02-Python/Organise-Forex.csv'

Company='04-Clark'

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

sDatabaseName=Base + '/' + Company + '/05-Organise/SQLite/clark.db'

conn = sq.connect(sDatabaseName)

#conn = sq.connect(':memory:')

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

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

### Import Forex Data

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

sFileName=Base + '/' + Company + '/' + sInputFileName

print('################################')

print('Loading :',sFileName)

print('################################')

ForexDataRaw=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1")

print('################################')

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

ForexDataRaw.index.names = ['RowID']

sTable='Forex\_All'

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

ForexDataRaw.to\_sql(sTable, conn, if\_exists="replace")

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

sSQL="SELECT 1 as Bag\

, CAST(min(Date) AS VARCHAR(10)) as Date \

,CAST(1000000.0000000 as NUMERIC(12,4)) as Money \

,'USD' as Currency \

FROM Forex\_All \

;"

sSQL=re.sub("\s\s+", " ", sSQL)

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

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

nMoney.index.names = ['RowID']

sTable='MoneyData'

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

nMoney.to\_sql(sTable, conn, if\_exists="replace")

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

sTable='TransactionData'

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

nMoney.to\_sql(sTable, conn, if\_exists="replace")

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

ForexDay=pd.read\_sql\_query("SELECT Date FROM Forex\_All GROUP BY Date;", conn)

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

t=0

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

sDay1=ForexDay['Date'][i]

sDay=str(sDay1)

sSQL='\

SELECT M.Bag as Bag, \

F.Date as Date, \

round(M.Money \* F.Rate,6) AS Money, \

F.CodeIn AS PCurrency, \

F.CodeOut AS Currency \

FROM MoneyData AS M \

JOIN \

( \

SELECT \

CodeIn, CodeOut, Date, Rate \

FROM \

Forex\_All \

WHERE\

CodeIn = "USD" AND CodeOut = "GBP" \

UNION \

SELECT \

CodeOut AS CodeIn, CodeIn AS CodeOut, Date, (1/Rate) AS Rate \

FROM \

Forex\_All \

WHERE\

CodeIn = "USD" AND CodeOut = "GBP" \

) AS F \

ON \

M.Currency=F.CodeIn \

AND \

F.Date ="' +sDay + '";'

sSQL=re.sub("\s\s+", " ", sSQL)

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

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

sBag=str(ForexDayRate['Bag'][j])

nMoney=str(round(ForexDayRate['Money'][j],2))

sCodeIn=ForexDayRate['PCurrency'][j]

sCodeOut=ForexDayRate['Currency'][j]

sSQL='UPDATE MoneyData SET Date= "' + sDay + '", '

sSQL= sSQL + ' Money = ' + nMoney + ', Currency="' + sCodeOut + '"'

sSQL= sSQL + ' WHERE Bag=' + sBag + ' AND Currency="' + sCodeIn + '";'

sSQL=re.sub("\s\s+", " ", sSQL)

cur = conn.cursor()

cur.execute(sSQL)

conn.commit()

t+=1

print('Trade :', t, sDay, sCodeOut, nMoney)

sSQL=' \

INSERT INTO TransactionData ( \

RowID, \

Bag, \

Date, \

Money, \

Currency \

) \

SELECT ' + str(t) + ' AS RowID, \

Bag, \

Date, \

Money, \

Currency \

FROM MoneyData \

;'

sSQL=re.sub("\s\s+", " ", sSQL)

cur = conn.cursor()

cur.execute(sSQL)

conn.commit()

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

sSQL="SELECT RowID, Bag, Date, Money, Currency FROM TransactionData ORDER BY

RowID;"

sSQL=re.sub("\s\s+", " ", sSQL)

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

OutputFile=Base + '/' + Company + '/' + sOutputFileName

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

**Output:**

Save the Assess-Forex.py file, then compile and execute with your Python compiler.

This will produce a set of demonstrated values onscreen.

**Practical-6**

**Transform Superstep**

**Code:**

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='C:/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\_items(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\_items(PersonLine)

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

TimeHub=PersonFrame

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

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

sTable = 'Hub-Person-Gunnarsson'

print('\n#################################')

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

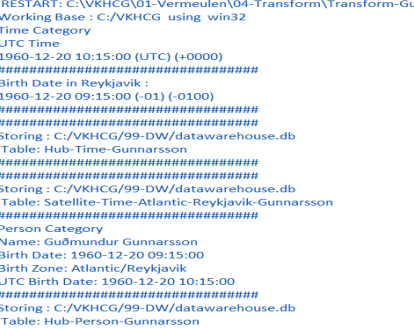
print('\n#################################')

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

sTable = 'Dim-Person-Gunnarsson'

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

**Output:**

****

**Building a Data Warehouse**

**Code:**

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

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

if sys.platform == 'linux':

Base=os.path.expanduser('~') + '/VKHCG'

else:

Base='C:/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)

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

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)

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

sSQL=" SELECT DateTimeValue FROM [Hub-Time];"

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

DateData=DateDataRaw.head(1000)

print(DateData)

print('\n#################################')

print('Time Dimension')

print('\n#################################')

t=0

mt=DateData.shape[0]

for i in range(mt):

BirthZone = ('Atlantic/Reykjavik','Europe/London','UCT')

for j in range(len(BirthZone)):

t+=1

print(t,mt\*3)

BirthDateUTC = datetime.strptime(DateData['DateTimeValue'][i],"%Y-%m-%d %H:%M:%S")

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[j]))

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', [str(IDTimeNumber)]),

('UTCDate', [str(BirthDateZoneStr)]),

('LocalTime', [str(BirthDateLocal)]),

('TimeZone', [str(BirthZone)])]

if t==1:

TimeFrame = pd.DataFrame.from\_items(TimeLine)

else:

TimeRow = pd.DataFrame.from\_items(TimeLine)

TimeFrame=TimeFrame.append(TimeRow)

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

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, conn3, if\_exists="replace")

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

sSQL=" SELECT " + \

" FirstName," + \

" SecondName," + \

" LastName," + \

" BirthDateKey " + \

" FROM [Hub-Person];"

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

PersonData=PersonDataRaw.head(1000)

print('\n#################################')

print('Dimension Person')

print('\n#################################')

t=0

mt=DateData.shape[0]

for i in range(mt):

t+=1

print(t,mt)

FirstName = str(PersonData["FirstName"])

SecondName = str(PersonData["SecondName"])

if len(SecondName) > 0:

SecondName=""

LastName = str(PersonData["LastName"])

BirthDateKey = str(PersonData["BirthDateKey"])

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

IDPersonNumber=str(uuid.uuid4())

PersonLine=[('PersonID', [str(IDPersonNumber)]),

('FirstName', [FirstName]),

('SecondName', [SecondName]),

('LastName', [LastName]),

('Zone', [str('UTC')]),

('BirthDate', [BirthDateKey])]

if t==1:

PersonFrame = pd.DataFrame.from\_items(PersonLine)

else:

PersonRow = pd.DataFrame.from\_items(PersonLine)

PersonFrame = PersonFrame.append(PersonRow)

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

DimPerson=PersonFrame

print(DimPerson)

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, conn3, if\_exists="replace")

**Output:**

You have successfully performed data vault to data warehouse transformation.

**Simple Linear Regression**

**Code:**

import sys

import os

import pandas as pd

import sqlite3 as sq

import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear\_model

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

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

Base='C:/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)

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

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)

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

t=0

tMax=((300-100)/10)\*((300-30)/5)

for heightSelect in range(100,300,10):

for weightSelect in range(30,300,5):

height = round(heightSelect/100,3)

weight = int(weightSelect)

bmi = weight/(height\*height)

if bmi <= 18.5:

BMI\_Result=1

elif bmi > 18.5 and bmi < 25:

BMI\_Result=2

elif bmi > 25 and bmi < 30:

BMI\_Result=3

elif bmi > 30:

BMI\_Result=4

else:

BMI\_Result=0

PersonLine=[('PersonID', [str(t)]),

('Height', [height]),

('Weight', [weight]),

('bmi', [bmi]),

('Indicator', [BMI\_Result])]

t+=1

print('Row:',t,'of',tMax)

if t==1:

PersonFrame = pd.DataFrame.from\_items(PersonLine)

else:

PersonRow = pd.DataFrame.from\_items(PersonLine)

PersonFrame = PersonFrame.append(PersonRow)

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

DimPerson=PersonFrame

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

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

sTable = 'Transform-BMI'

print('\n#################################')

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

print('\n#################################')

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

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

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

sTable = 'Person-Satellite-BMI'

print('\n#################################')

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

print('\n#################################')

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

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

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

sTable = 'Dim-BMI'

print('\n#################################')

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

print('\n#################################')

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

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

fig = plt.figure()

PlotPerson=DimPerson[DimPerson['Indicator']==1]

x=PlotPerson['Height']

y=PlotPerson['Weight']

plt.plot(x, y, ".")

PlotPerson=DimPerson[DimPerson['Indicator']==2]

x=PlotPerson['Height']

y=PlotPerson['Weight']

plt.plot(x, y, "o")

PlotPerson=DimPerson[DimPerson['Indicator']==3]

x=PlotPerson['Height']

y=PlotPerson['Weight']

plt.plot(x, y, "+")

PlotPerson=DimPerson[DimPerson['Indicator']==4]

x=PlotPerson['Height']

y=PlotPerson['Weight']

plt.plot(x, y, "^")

plt.axis('tight')

plt.title("BMI Curve")

plt.xlabel("Height(meters)")

plt.ylabel("Weight(kg)")

plt.plot()

diabetes = datasets.load\_diabetes()

# Use only one feature

diabetes\_X = diabetes.data[:, np.newaxis, 2]

diabetes\_X\_train = diabetes\_X[:-30]

diabetes\_X\_test = diabetes\_X[-50:]

diabetes\_y\_train = diabetes.target[:-30]

diabetes\_y\_test = diabetes.target[-50:]

regr = linear\_model.LinearRegression()

regr.fit(diabetes\_X\_train, diabetes\_y\_train)

diabetes\_y\_pred = regr.predict(diabetes\_X\_test)

print('Coefficients: \n', regr.coef\_)

print("Mean squared error: %.2f"

% mean\_squared\_error(diabetes\_y\_test, diabetes\_y\_pred))

print('Variance score: %.2f' % r2\_score(diabetes\_y\_test, diabetes\_y\_pred))

plt.scatter(diabetes\_X\_test, diabetes\_y\_test, color='black')

plt.plot(diabetes\_X\_test, diabetes\_y\_pred, color='blue', linewidth=3)

plt.xticks(())

plt.yticks(())

plt.axis('tight')

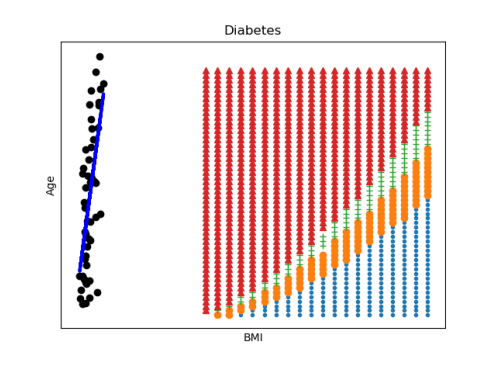
plt.title("Diabetes")

plt.xlabel("BMI")

plt.ylabel("Age")

plt.show()

**Output:**

****

**Practical-7**

**Report Superstep**

**Vermeulen PLC**

**Code:**

import sys

import os

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt

pd.options.mode.chained\_assignment = None

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

if sys.platform == 'linux':

Base=os.path.expanduser('~') + 'VKHCG'

else:

Base='C:/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'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())

print(CustomerData.shape)

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

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)

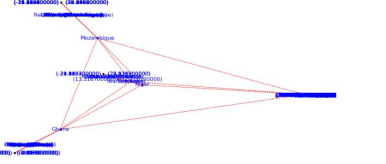
plt.show()

print('################################')

print('### Done!! #####################')

print('################################')

**Output:**

****

**Krennwallner AG**

**Code:**

import sys

import os

import pandas as pd

from folium.plugins import FastMarkerCluster, HeatMap

from folium import Marker, Map

import webbrowser

Base='C:/VKHCG'

print('################################')

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

print('################################')

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

sFileName=Base+'/02-Krennwallner/01-Retrieve/01-EDS/02-Python/Retrieve\_DE\_Billboard\_Locations.csv'

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

df.fillna(value=0, inplace=True)

print(df.shape)

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

t=0

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

try:

sLongitude=df["Longitude"][i]

sLongitude=float(sLongitude)

except Exception:

sLongitude=float(0.0)

try:

sLatitude=df["Latitude"][i]

sLatitude=float(sLatitude)

except Exception:

sLatitude=float(0.0)

try:

sDescription=df["Place\_Name"][i] + ' (' + df["Country"][i]+')'

except Exception:

sDescription='VKHCG'

if sLongitude != 0.0 and sLatitude != 0.0:

DataClusterList=list([sLatitude, sLongitude])

DataPointList=list([sLatitude, sLongitude, sDescription])

t+=1

if t==1:

DataCluster=[DataClusterList]

DataPoint=[DataPointList]

else:

DataCluster.append(DataClusterList)

DataPoint.append(DataPointList)

data=DataCluster

pins=pd.DataFrame(DataPoint)

pins.columns = [ 'Latitude','Longitude','Description']

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

stops\_map1 = Map(location=[48.1459806, 11.4985484], zoom\_start=5)

marker\_cluster = FastMarkerCluster(data).add\_to(stops\_map1)

sFileNameHtml=Base+'/02-Krennwallner/06-Report/01-EDS/02-Python/Billboard1.html'

stops\_map1.save(sFileNameHtml)

webbrowser.open('file://' + os.path.realpath(sFileNameHtml))

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

stops\_map2 = Map(location=[48.1459806, 11.4985484], zoom\_start=5)

for name, row in pins.iloc[:100].iterrows():

Marker([row["Latitude"],row["Longitude"]], popup=row["Description"]).add\_to(stops\_map2)

sFileNameHtml=Base+'/02-Krennwallner/06-Report/01-EDS/02-Python/Billboard2.html'

stops\_map2.save(sFileNameHtml)

webbrowser.open('file://' + os.path.realpath(sFileNameHtml))

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

stops\_heatmap = Map(location=[48.1459806, 11.4985484], zoom\_start=5)

stops\_heatmap.add\_child(HeatMap([[row["Latitude"], row["Longitude"]] for name, row in

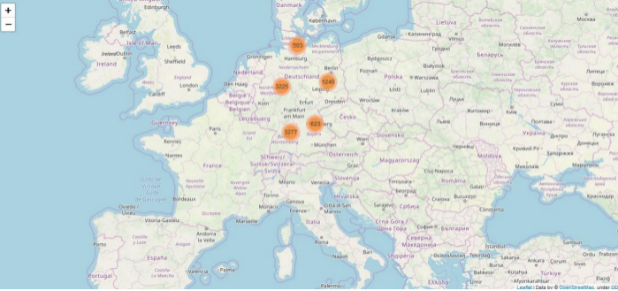
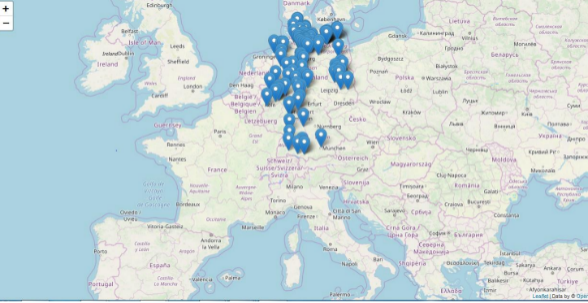
pins.iloc[:100].iterrows()]))

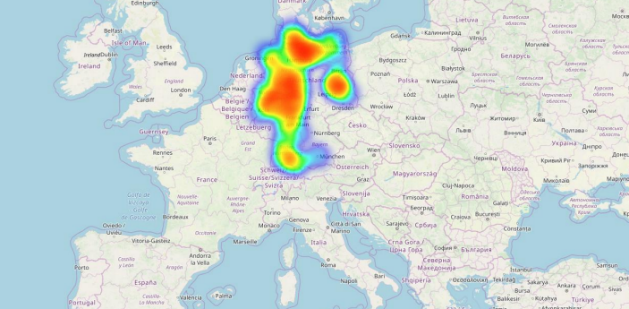
sFileNameHtml=Base+'/02-Krennwallner/06-Report/01-EDS/02-Python/Billboard\_heatmap.html'

stops\_heatmap.save(sFileNameHtml)

webbrowser.open('file://' + os.path.realpath(sFileNameHtml))

**Output:**

** **

****

**Hillman Ltd**

**Code:**

from time import time

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import offsetbox

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

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

X = digits.data

y = digits.target

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.),

fontdict={'weight': 'bold', 'size': 9})

if hasattr(offsetbox, 'AnnotationBbox'):

# only print thumbnails with matplotlib > 1.0

shown\_images = np.array([[1., 1.]]) # 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\_transform(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)

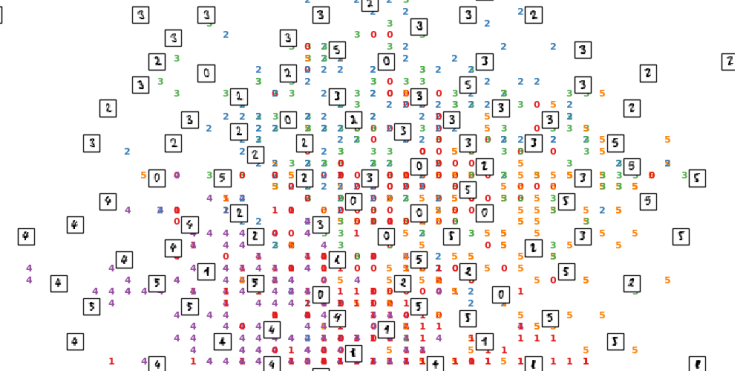
t0 = time()

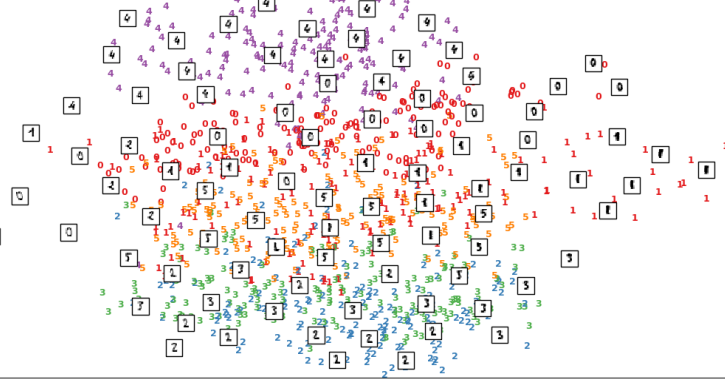
X\_tsne = tsne.fit\_transform(X)

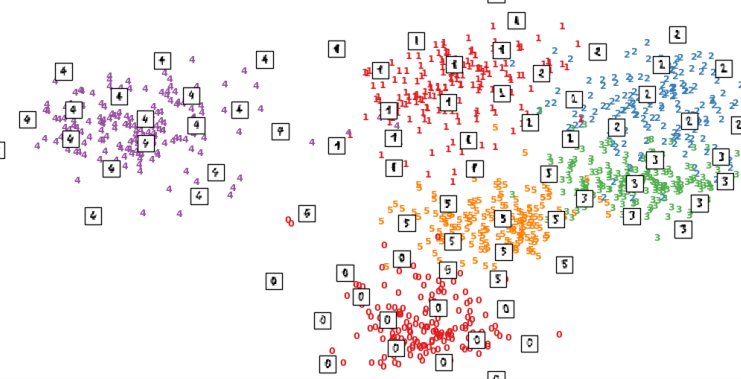
plot\_embedding(X\_tsne,"t-SNE embedding of the digits (time %.2fs)" %(time() - t0))

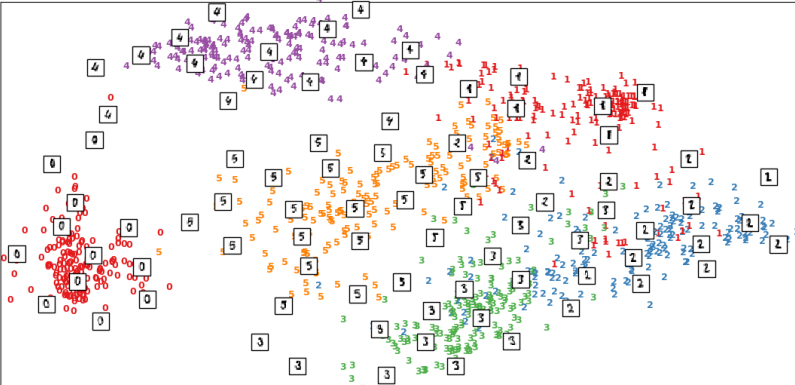
plt.show()

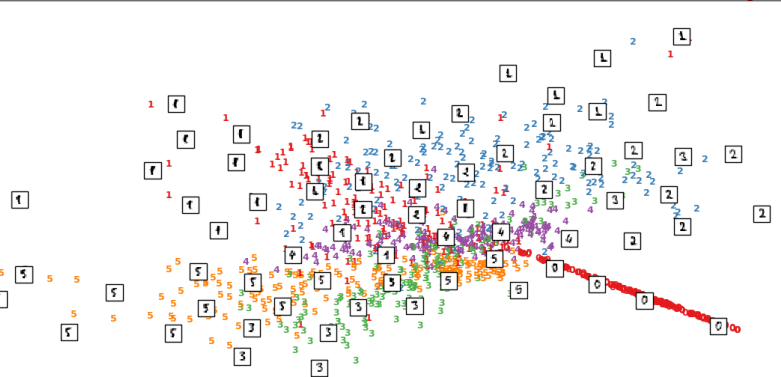
**Output:**

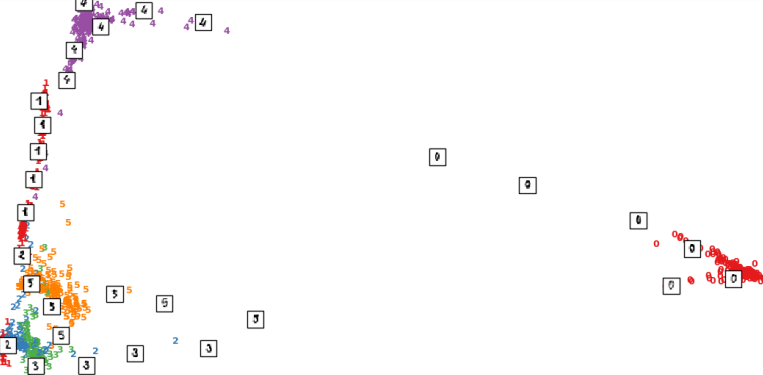
****

****

****

****

****

****

**Clark Ltd**

**Code:**

import sys

import os

import pandas as pd

import sqlite3 as sq

import re

from openpyxl import load\_workbook

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

Base='C:/VKHCG'

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

print('################################')

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

print('################################')

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

sInputTemplateName='00-RawData/Balance-Sheet-Template.xlsx'

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

sOutputFileName='06-Report/01-EDS/02-Python/Report-Balance-Sheet'

Company='04-Clark'

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

sDatabaseName=Base + '/' + Company + '/06-Report/SQLite/clark.db'

conn = sq.connect(sDatabaseName)

#conn = sq.connect(':memory:')

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

### Import Balance Sheet Data

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

for y in range(1,13):

sInputFileName='00-RawData/BalanceSheets' + str(y).zfill(2) + '.csv'

sFileName=Base + '/' + Company + '/' + sInputFileName

print('################################')

print('Loading :',sFileName)

print('################################')

ForexDataRaw=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1")

print('################################')

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

ForexDataRaw.index.names = ['RowID']

sTable='BalanceSheets'

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

if y == 1:

print('Load Data')

ForexDataRaw.to\_sql(sTable, conn, if\_exists="replace")

else:

print('Append Data')

ForexDataRaw.to\_sql(sTable, conn, if\_exists="append")

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

sSQL="SELECT \

Year, \

Quarter, \

Country, \

Company, \

CAST(Year AS INT) || 'Q' || CAST(Quarter AS INT) AS sDate, \

Company || ' (' || Country || ')' AS sCompanyName , \

CAST(Year AS INT) || 'Q' || CAST(Quarter AS INT) || '-' ||\

Company || '-' || Country AS sCompanyFile \

FROM BalanceSheets \

GROUP BY \

Year, \

Quarter, \

Country, \

Company \

HAVING Year is not null \

;"

sSQL=re.sub("\s\s+", " ", sSQL)

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

print(sDatesRaw.shape)

sDates=sDatesRaw.head(5)

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

## Loop Dates

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

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

sFileName=Base + '/' + Company + '/' + sInputTemplateName

wb = load\_workbook(sFileName)

ws=wb.get\_sheet\_by\_name("Balance-Sheet")

sYear=sDates['sDate'][i]

sCompany=sDates['sCompanyName'][i]

sCompanyFile=sDates['sCompanyFile'][i]

sCompanyFile=re.sub("\s+", "", sCompanyFile)

ws['D3'] = sYear

ws['D5'] = sCompany

sFields = pd.DataFrame(

[

['Cash','D16', 1],

['Accounts\_Receivable','D17', 1],

['Doubtful\_Accounts','D18', 1],

['Inventory','D19', 1],

['Temporary\_Investment','D20', 1],

['Prepaid\_Expenses','D21', 1],

['Long\_Term\_Investments','D24', 1],

['Land','D25', 1],

['Buildings','D26', 1],

['Depreciation\_Buildings','D27', -1],

['Plant\_Equipment','D28', 1],

['Depreciation\_Plant\_Equipment','D29', -1],

['Furniture\_Fixtures','D30', 1],

['Depreciation\_Furniture\_Fixtures','D31', -1],

['Accounts\_Payable','H16', 1],

['Short\_Term\_Notes','H17', 1],

['Current\_Long\_Term\_Notes','H18', 1],

['Interest\_Payable','H19', 1],

['Taxes\_Payable','H20', 1],

['Accrued\_Payroll','H21', 1],

['Mortgage','H24', 1],

['Other\_Long\_Term\_Liabilities','H25', 1],

['Capital\_Stock','H30', 1]

]

)

nYear=str(int(sDates['Year'][i]))

nQuarter=str(int(sDates['Quarter'][i]))

sCountry=str(sDates['Country'][i])

sCompany=str(sDates['Company'][i])

sFileName=Base + '/' + Company + '/' + sOutputFileName + \

'-' + sCompanyFile + '.xlsx'

print(sFileName)

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

sSumField=sFields[0][j]

sCellField=sFields[1][j]

nSumSign=sFields[2][j]

sSQL="SELECT \

Year, \

Quarter, \

Country, \

Company, \

SUM(" + sSumField + ") AS nSumTotal \

FROM BalanceSheets \

GROUP BY \

Year, \

Quarter, \

Country, \

Company \

HAVING \

Year=" + nYear + " \

AND \

Quarter=" + nQuarter + " \

AND \

Country='" + sCountry + "' \

AND \

Company='" + sCompany + "' \

;"

sSQL=re.sub("\s\s+", " ", sSQL)

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

ws[sCellField] = sSumRaw["nSumTotal"][0] \* nSumSign

print('Set cell',sCellField,' to ', sSumField,'Total')

wb.save(sFileName)

**Output:**

You now have all the reports you need.

Check the Following files for generated reports in C:/VKHCG/04-Clark/06-Report/01-EDS/02-Python/

1. Report-Balance-Sheet-2000Q1-Clark-Afghanistan.xlsx

2. Report-Balance-Sheet-2000Q1-Hillman-Afghanistan.xlsx

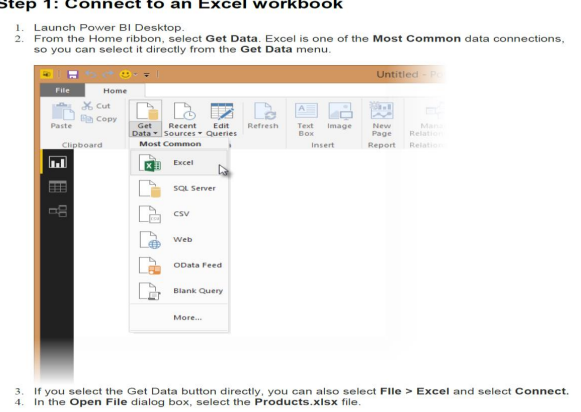
3. Report-Balance-Sheet-2000Q1-Krennwallner-Afghanistan.xlsx

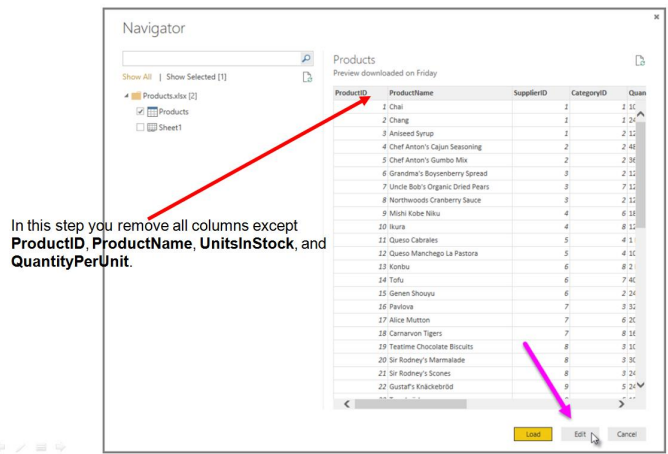
4. Report-Balance-Sheet-2000Q1-Vermeulen-Afghanistan.xlsx

5. Report-Balance-Sheet-2000Q1-Clark-AlandIslands.xlsx

**Practical-8**

**Case Study: Sales Data**

****

****

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

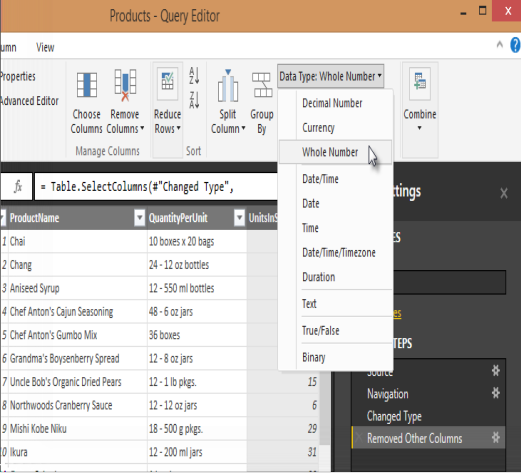
(use Ctrl+Click to select more than one column, or Shift+Click to select columns that are

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





Step 3: Change the data type of the UnitsInStock column

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. 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).

**Task 2: Import order data from an OData feed**

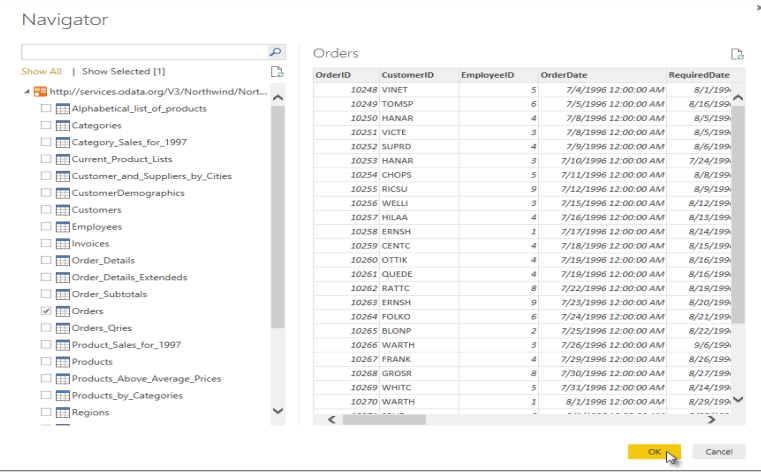
Step 1: Connect to an OData feed

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

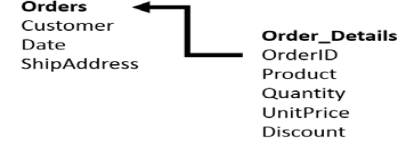
2. Browse to the OData Feed data source.

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

4. Select OK.



Step 2: Expand the Order\_Details table



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.

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.

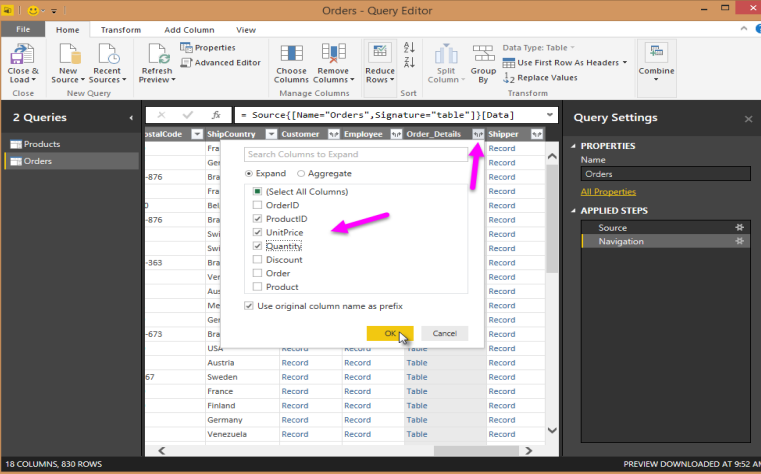
1. In the Query View, scroll to the Order\_Details column.

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

3. In the Expand drop-down: a. Select (Select All Columns) to clear all columns.

Select ProductID, UnitPrice, and Quantity.

click OK.



Step 3: Remove other columns to only display columns of interest

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

Order\_Details.ProductID, Order\_Details.UnitPrice, and Order\_Details.Quantity columns. In the

previous task, you used Remove Other Columns. For this task, you remove selected columns.

In the Query View, select all columns by completing a.

a. Click the first column (OrderID).

b. Shift+Click the last column (Shipper).

c. Now that all columns are selected, use Ctrl+Click to unselect the following columns:

OrderDate, ShipCity, ShipCountry, Order\_Details.ProductID, Order\_Details.UnitPrice, and

Order\_Details.Quantity.

Now that only the columns we want to remove are selected, right-click on any selected column

header and click Remove Columns.

Step 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.

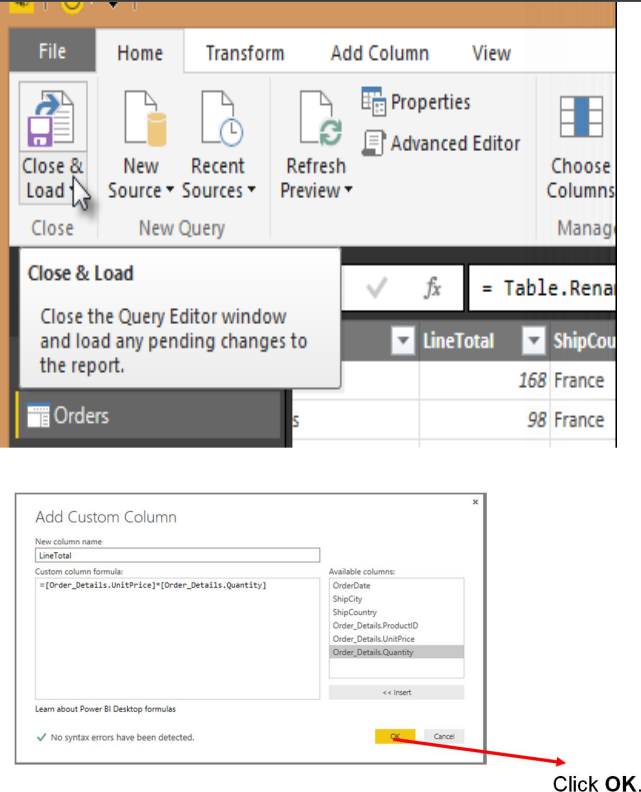
Calculate the line total for each Order\_Details row:

1. In the Add Column ribbon tab, click Add Custom Column.

2. In the Add Custom Column dialog box, in the Custom Column Formula textbox, enter

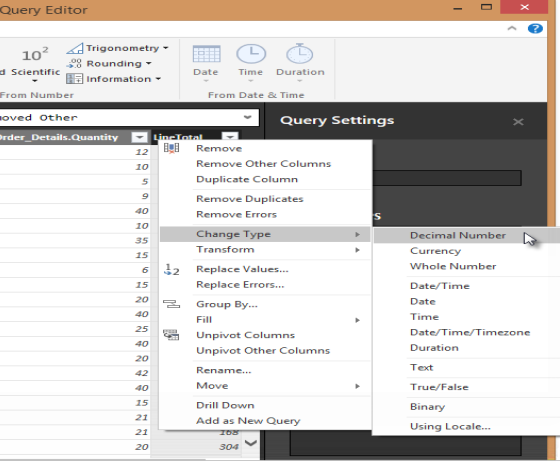
[Order\_Details.UnitPrice] \* [Order\_Details.Quantity].

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



Step 5: Set the datatype of the LineTotal field

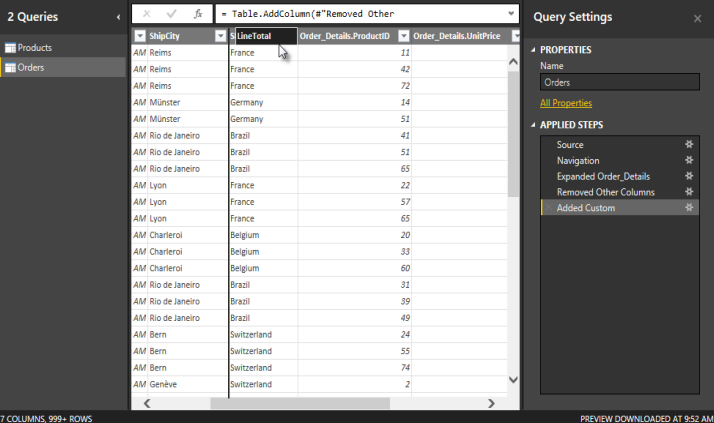
1. Right click the LineTotal column.
2. Select Change Type and choose Decimal Number.



Step 6: Rename and reorder columns in the query

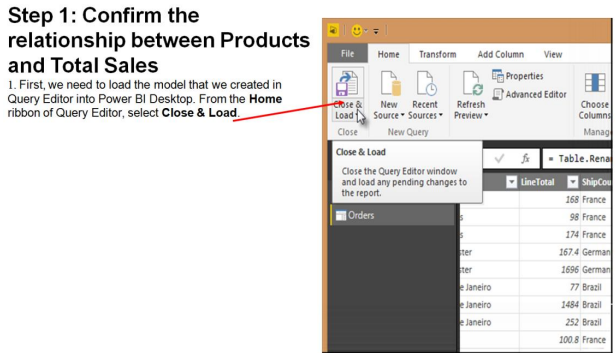
1. In Query Editor, drag the LineTotal column to the left, after ShipCountry.

2. Remove



1. 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.

**Task 3: Combine the Products and Total Sales queries**

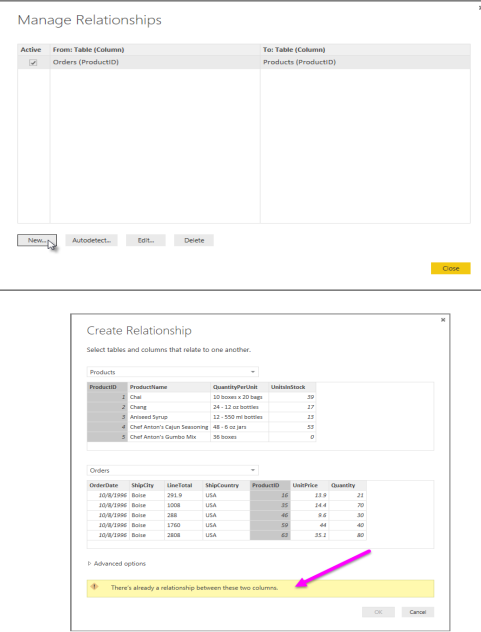
****

2. Power BI Desktop loads the data from the two queries

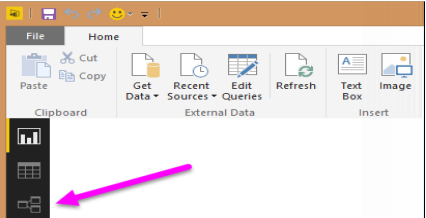
3. Once the data is loaded, select the Manage Relationships button Home ribbon

4. Select the New… button

5. 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.

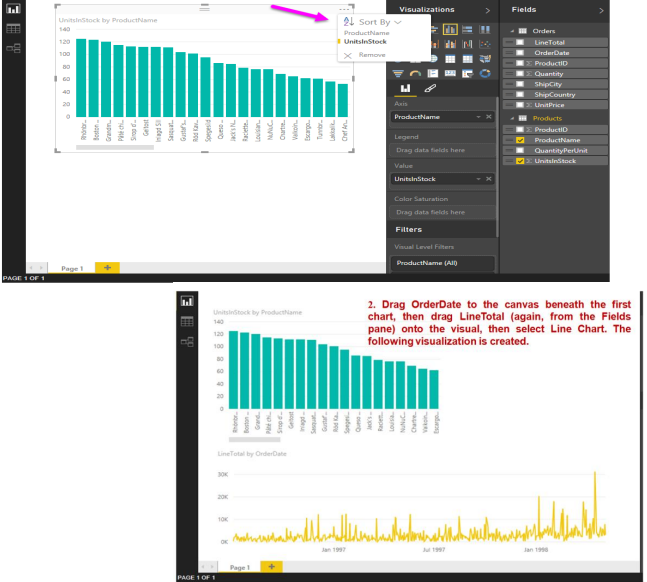


1. Select Cancel, and then select Relationship view in Power BI Desktop

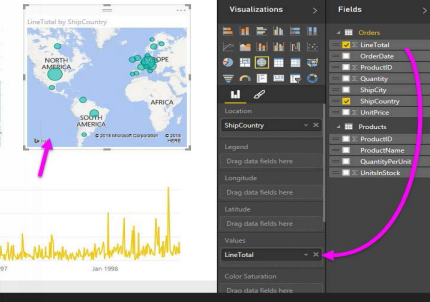


**Task 4: Build visuals using your data**

Step 1: Create charts showing Units in Stock by Product and Total Sales by Year



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 visuals to analze further

