DATA SCIENCE

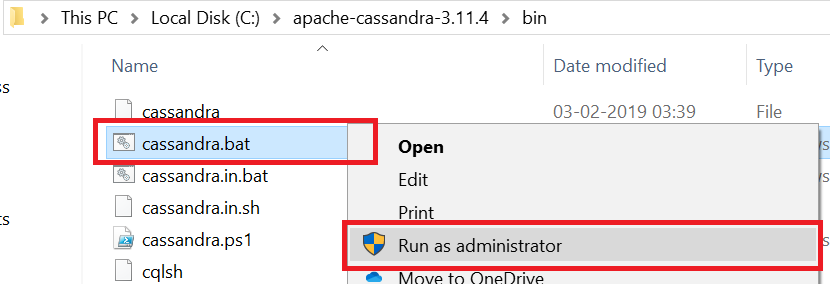
**Table of Contents**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Practical No** | | **Name of the Practical** | **SIGNATURE** |
| **1)** | **---** | | Prerequisites to Data Science Practical. |  |
| **2)** | **1** | | Creating Data Model using Cassandra. |  |
| **3)** | **2** | | **Conversion from different formats to HOURS format.** |  |
| **4)** |  | **A.** | Text delimited csv format. |  |
| **5)** |  | **B.** | XML |  |
| **6)** |  | **C.** | JSON |  |
| **7)** |  | **D.** | MySQL Database |  |
| **8)** |  | **E.** | Picture (JPEG) |  |
| **9)** |  | **F.** | Video |  |
| **10)** |  | **G.** | Audio |  |
| **11)** | **3** | | **Utilities and Auditing** |  |
| **12)** | **4** | | **Retrieving Data** |  |
| **13)** | **5** | | **Assessing Data** |  |
| **14)** | **6** | | **Processing Data** |  |
| **15)** | **7** | | **Transforming Data** |  |
| **16)** | **8** | | **Organizing Data** |  |
| **17)** | **9** | | **Generating Reports** |  |
| **18)** | **10** | | **Data Visualization with Power BI** |  |

Practical 1:

## Creating Data Model using Cassandra.

Go to Cassandra directory

C:\apache-cassandra-3.11.4\bin

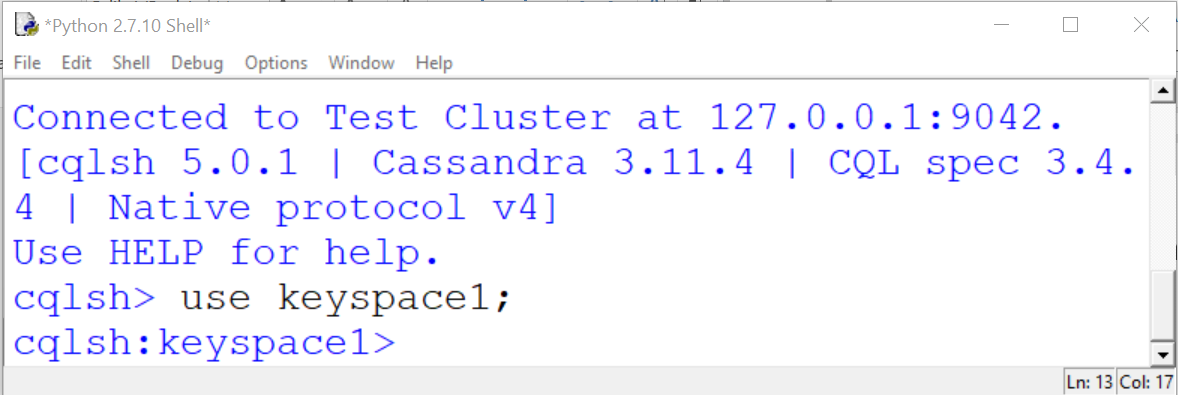
Run Cassandra.bat file

Open C:\apache-cassandra-3.11.4\bin\cqlsh.py with python 2.7 and run

**Creating a Keyspace using Cqlsh**

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

Use keyspace1;



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

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

Insert into dept (dept\_id, dept\_name, dept\_loc) values (1001, 'Accounts', 'Mumbai'); Insert into dept (dept\_id, dept\_name, dept\_loc) values (1002, 'Marketing', 'Delhi'); Insert into dept (dept\_id, dept\_name, dept\_loc) values (1003, 'HR', 'Chennai');

Insert into emp ( emp\_id, emp\_name, dept\_id, email, phone ) values (1001, 'ABCD', 1001, 'abcd@company.com', '1122334455');

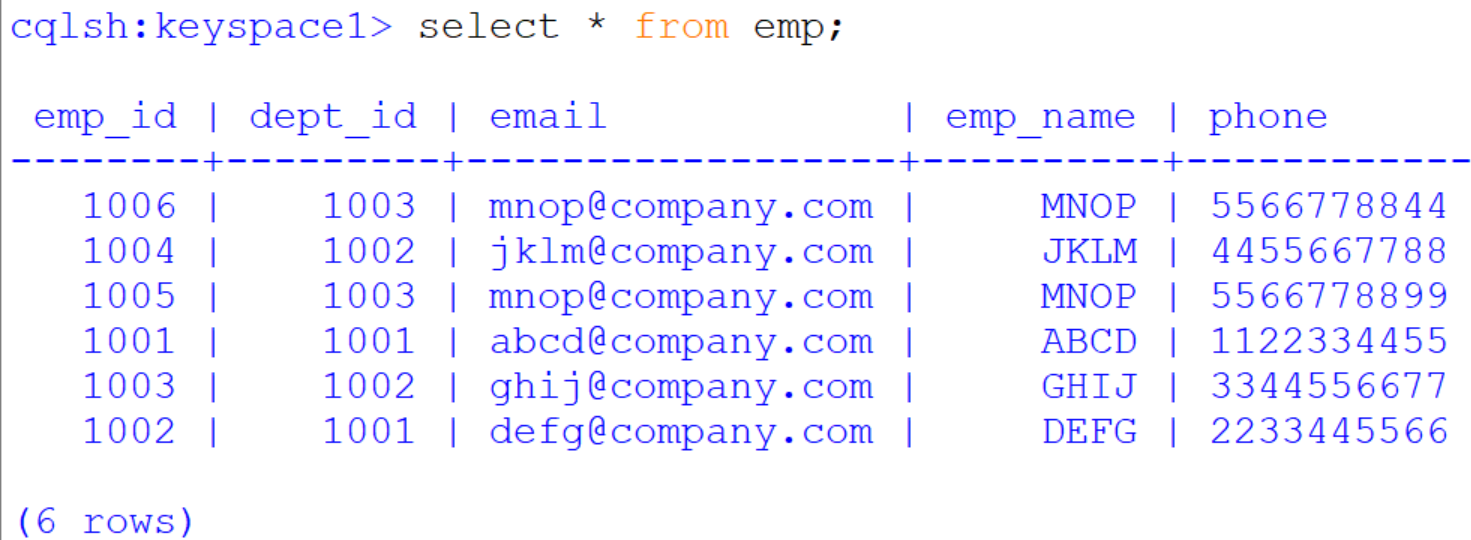
Insert into emp ( emp\_id, emp\_name, dept\_id, email, phone ) values (1002, 'DEFG', 1001, 'defg@company.com', '2233445566');

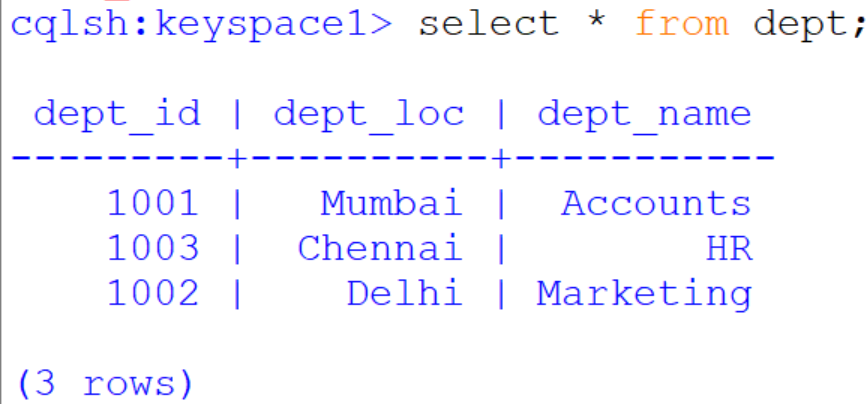
Insert into emp ( emp\_id, emp\_name, dept\_id, email, phone ) values (1003, 'GHIJ', 1002, 'ghij@company.com', '3344556677');

Insert into emp ( emp\_id, emp\_name, dept\_id, email, phone ) values (1004, 'JKLM', 1002, 'jklm@company.com', '4455667788');

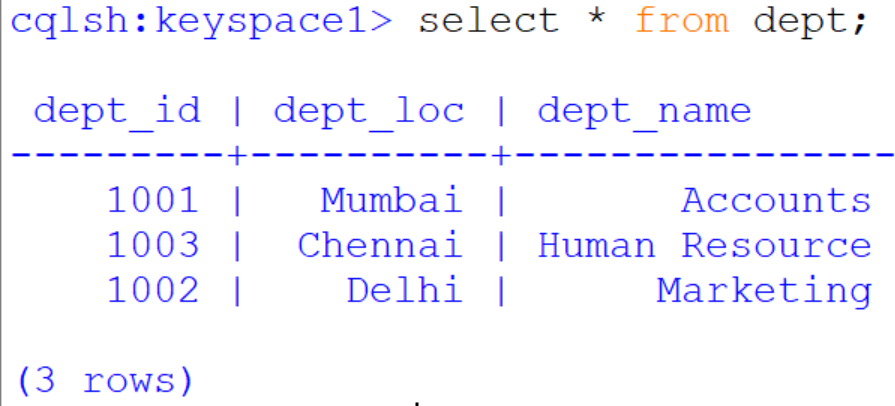
Insert into emp ( emp\_id, emp\_name, dept\_id, email, phone ) values (1005, 'MNOP', 1003, 'mnop@company.com', '5566778899');

Insert into emp ( emp\_id, emp\_name, dept\_id, email, phone ) values (1006, 'MNOP', 1003, 'mnop@company.com', '5566778844');





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





**Practical 2:**

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

1. **Text delimited CSVto HORUS format. Code:**

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

# Standard Tools import pandas as pd

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

sInputFileName='C:/VKHCG/05-DS/9999-Data/Country\_Code.csv' InputData=pd.read\_csv(sInputFileName,encoding="latin-1")

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

print(InputData) print('=====================================================') # Processing Rules ===========================================

ProcessData=InputData

# Remove columns ISO-2-Code and ISO-3-CODE ProcessData.drop('ISO-2-CODE', axis=1,inplace=True) ProcessData.drop('ISO-3-Code', axis=1,inplace=True) # Rename Country and ISO-M49

ProcessData.rename(columns={'Country': 'CountryName'}, inplace=True) ProcessData.rename(columns={'ISO-M49': 'CountryNumber'}, inplace=True) # Set new Index

ProcessData.set\_index('CountryNumber', inplace=True) # Sort data by CurrencyNumber

ProcessData.sort\_values('CountryName', axis=0, ascending=False, inplace=True) print('Process Data Values =================================') print(ProcessData) print('=====================================================') # Output Agreement ===========================================

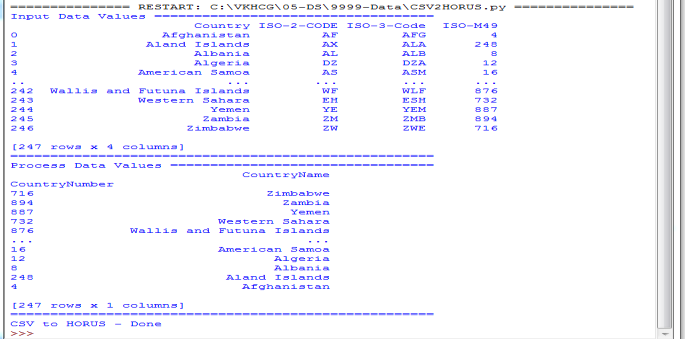
OutputData=ProcessData

sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-CSV-Country.csv' OutputData.to\_csv(sOutputFileName, index = False)

print('CSV to HORUS - Done')

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

**Output:**



1. **XML to HORUS Format**

**Code:**

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

# Standard Tools import pandas as pd

import xml.etree.ElementTree as ET def df2xml(data):

header = data.columns root = ET.Element('root')

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

entry = ET.SubElement(root,'entry') for index in range(data.shape[1]):

schild=str(header[index])

child = ET.SubElement(entry, schild) if str(data[schild][row]) != 'nan':

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

child.text = 'n/a' entry.append(child)

result = ET.tostring(root) return result

def xml2df(xml\_data):

root = ET.XML(xml\_data) all\_records = []

for i, child in enumerate(root): record = {}

for subchild in child: record[subchild.tag] = subchild.text

all\_records.append(record) return pd.DataFrame(all\_records)

sInputFileName='C:/VKHCG/05-DS/9999-Data/Country\_Code.xml' InputData = open(sInputFileName).read()

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

print(InputData) print('=====================================================') #============================================================

=

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

=

ProcessDataXML=InputData # XML to Data Frame

ProcessData=xml2df(ProcessDataXML)

# Remove columns ISO-2-Code and ISO-3-CODE ProcessData.drop('ISO-2-CODE', axis=1,inplace=True) ProcessData.drop('ISO-3-Code', axis=1,inplace=True) # Rename Country and ISO-M49

ProcessData.rename(columns={'Country': 'CountryName'}, inplace=True) ProcessData.rename(columns={'ISO-M49': 'CountryNumber'}, inplace=True) # Set new Index

ProcessData.set\_index('CountryNumber', inplace=True) # Sort data by CurrencyNumber

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

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

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

OutputData=ProcessData

sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-XML-Country.csv' OutputData.to\_csv(sOutputFileName, index = False) print('=====================================================')

print('XML to HORUS - Done') print('=====================================================') # Utility done ===============================================

**Output:**



1. **JSON to HORUS Format Code:**

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

# Standard Tools #============================================================

=

import pandas as pd

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

sInputFileName='C:/VKHCG/05-DS/9999-Data/Country\_Code.json' InputData=pd.read\_json(sInputFileName, orient='index', encoding="latin-1") print('Input Data Values ===================================')

print(InputData)

print('=====================================================') # Processing Rules ===========================================

ProcessData=InputData

# Remove columns ISO-2-Code and ISO-3-CODE ProcessData.drop('ISO-2-CODE', axis=1,inplace=True) ProcessData.drop('ISO-3-Code', axis=1,inplace=True) # Rename Country and ISO-M49

ProcessData.rename(columns={'Country': 'CountryName'}, inplace=True) ProcessData.rename(columns={'ISO-M49': 'CountryNumber'}, inplace=True) # Set new Index

ProcessData.set\_index('CountryNumber', inplace=True) # Sort data by CurrencyNumber

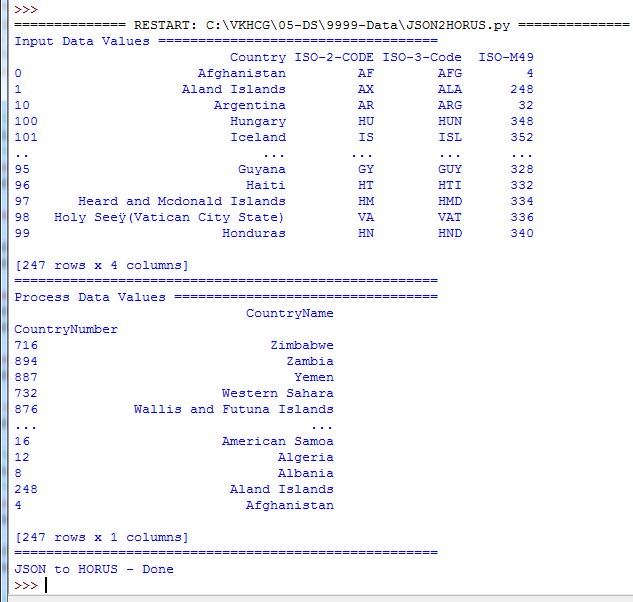
ProcessData.sort\_values('CountryName', axis=0, ascending=False, inplace=True) print('Process Data Values =================================') print(ProcessData) print('=====================================================') # Output Agreement ===========================================

OutputData=ProcessData

sOutputFileName='c:/VKHCG/05-DS/9999-Data/HORUS-JSON-Country.csv' OutputData.to\_csv(sOutputFileName, index = False)

print('JSON to HORUS - Done')

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

**Output:**

1. **MySql Database to HORUS Format Code:**

# Utility Start Database to HORUS ================================= # Standard Tools

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

=

import pandas as pd import sqlite3 as sq

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

sInputFileName='C:/VKHCG/05-DS/9999-Data/utility.db' sInputTable='Country\_Code'

conn = sq.connect(sInputFileName) sSQL='select \* FROM ' + sInputTable + ';' InputData=pd.read\_sql\_query(sSQL, conn)

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

print(InputData) print('=====================================================') # Processing Rules ===========================================

ProcessData=InputData

# Remove columns ISO-2-Code and ISO-3-CODE ProcessData.drop('ISO-2-CODE', axis=1,inplace=True) ProcessData.drop('ISO-3-Code', axis=1,inplace=True) # Rename Country and ISO-M49

ProcessData.rename(columns={'Country': 'CountryName'}, inplace=True) ProcessData.rename(columns={'ISO-M49': 'CountryNumber'}, inplace=True) # Set new Index

ProcessData.set\_index('CountryNumber', inplace=True) # Sort data by CurrencyNumber

ProcessData.sort\_values('CountryName', axis=0, ascending=False, inplace=True) print('Process Data Values =================================') print(ProcessData) print('=====================================================') # Output Agreement ===========================================

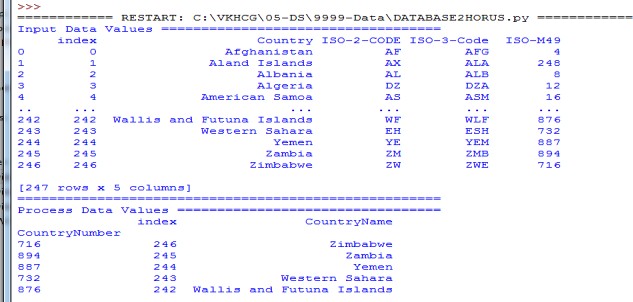
OutputData=ProcessData

sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-CSV-Country.csv' OutputData.to\_csv(sOutputFileName, index = False)

print('Database to HORUS - Done')

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

**Output:**



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

# Utility Start Picture to HORUS ================================= # Standard Tools

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

=

from scipy.misc import imread import pandas as pd

import matplotlib.pyplot as plt import numpy as np

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

sInputFileName='C:/VKHCG/05-DS/9999-Data/Angus.jpg' 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)

ProcessData=pd.DataFrame(np.reshape(ProcessRawData, (x, y))) sColumns= ['XAxis','YAxis','Red', 'Green', 'Blue','Alpha'] 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()

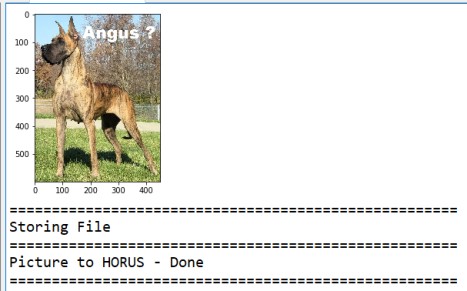
print('=====================================================') # Output Agreement ===========================================

OutputData=ProcessData print('Storing File')

sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-Picture.csv' OutputData.to\_csv(sOutputFileName, index = False) print('=====================================================')

print('Picture to HORUS - Done') print('=====================================================')

**Output:**



1. **Video to HORUS Format Code:**

**Movie to Frames**

# Utility Start Movie to HORUS (Part 1) ====================== # Standard Tools

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

import os

import shutil import cv2

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

sInputFileName='C:/VKHCG/05-DS/9999-Data/dog.mp4' sDataBaseDir='C:/VKHCG/05-DS/9999-Data/temp'

if os.path.exists(sDataBaseDir): shutil.rmtree(sDataBaseDir)

if not os.path.exists(sDataBaseDir): os.makedirs(sDataBaseDir)

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

print('Start Movie to Frames') print('=====================================================')

vidcap = cv2.VideoCapture(sInputFileName) success,image = vidcap.read()

count = 0 while success:

success,image = vidcap.read()

sFrame=sDataBaseDir + str('/dog-frame-' + str(format(count, '04d'))+ '.jpg') print('Extracted: ', sFrame)

cv2.imwrite(sFrame, image)

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

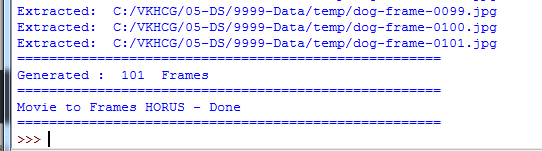
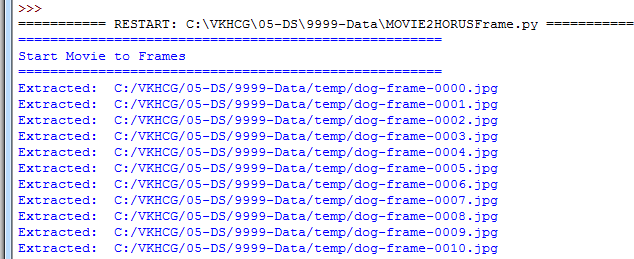
count += -1 os.remove(sFrame) print('Removed: ', sFrame)

if cv2.waitKey(10) == 27: # exit if Escape is hit break

count += 1 print('=====================================================')

print('Generated : ', count, ' Frames') print('=====================================================')

print('Movie to Frames HORUS - Done') print('=====================================================') # Utility done ===============================================



**Now frames are created and need to load them into HORUS.**

**Frames to Horus**

# Utility Start Movie to HORUS (Part 2) ====================== # Standard Tools

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

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])

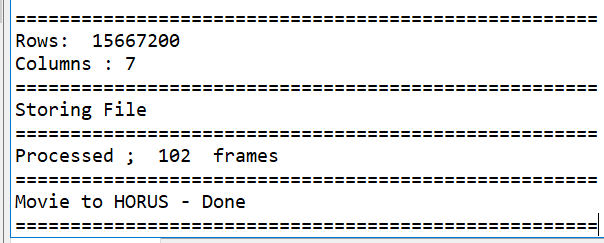
print('=====================================================') # Output Agreement ===========================================

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:**

|  |  |
| --- | --- |
| dog-frame-0000.jpg | dog-frame-0001.jpg |
| **dog-frame-0000.jpeg** | **dog-frame-0001.jpeg** |
| dog-frame-0100.jpg | dog-frame-0101.jpg |
| **dog-frame-0100.jpeg** | **dog-frame-0101.jpeg** |

**Check the files from** C:\VKHCG\05-DS\9999-Data\temp

The movie clip is converted into 102 picture frames and then to HORUS format.

1. **Audio to HORUS Format Code:**

# Utility Start Audio to HORUS ===============================

# Standard Tools #============================================================

=

from scipy.io import wavfile import pandas as pd

import matplotlib.pyplot as plt

import numpy as np #============================================================

=

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=sLegend+[str(c+1)] plt.plot(a[:,c], label=sLabel)

plt.legend(sLegend) plt.show()

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

=

sInputFileName='C:/VKHCG/05-DS/9999-Data/2ch-sound.wav' print('=====================================================')

print('Processing : ', sInputFileName) print('=====================================================')

InputRate, InputData = wavfile.read(sInputFileName) show\_info("2 channel", InputData,InputRate) ProcessData=pd.DataFrame(InputData)

sColumns= ['Ch1','Ch2'] ProcessData.columns=sColumns OutputData=ProcessData

sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-Audio-2ch.csv' OutputData.to\_csv(sOutputFileName, index = False) #============================================================

=

sInputFileName='C:/VKHCG/05-DS/9999-Data/4ch-sound.wav' print('=====================================================')

print('Processing : ', sInputFileName) print('=====================================================')

InputRate, InputData = wavfile.read(sInputFileName) show\_info("4 channel", InputData,InputRate) ProcessData=pd.DataFrame(InputData)

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

ProcessData.columns=sColumns OutputData=ProcessData

sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-Audio-4ch.csv' OutputData.to\_csv(sOutputFileName, index = False) #============================================================

=

sInputFileName='C:/VKHCG/05-DS/9999-Data/6ch-sound.wav' print('=====================================================')

print('Processing : ', sInputFileName) print('=====================================================')

InputRate, InputData = wavfile.read(sInputFileName) show\_info("6 channel", InputData,InputRate) ProcessData=pd.DataFrame(InputData)

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

ProcessData.columns=sColumns OutputData=ProcessData

sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-Audio-6ch.csv' OutputData.to\_csv(sOutputFileName, index = False) #============================================================

=

sInputFileName='C:/VKHCG/05-DS/9999-Data/8ch-sound.wav' print('=====================================================')

print('Processing : ', sInputFileName) print('=====================================================')

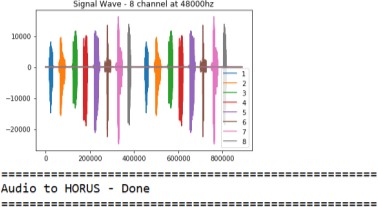
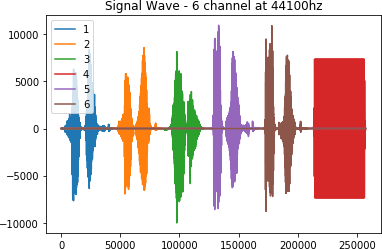
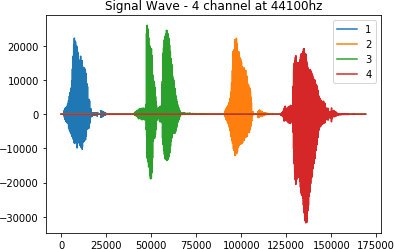
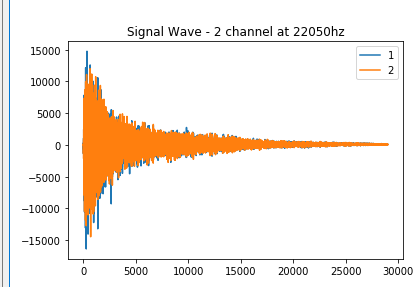
InputRate, InputData = wavfile.read(sInputFileName) show\_info("8 channel", InputData,InputRate) ProcessData=pd.DataFrame(InputData)

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

ProcessData.columns=sColumns OutputData=ProcessData

sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-Audio-8ch.csv' OutputData.to\_csv(sOutputFileName, index = False) print('=====================================================')

print('Audio to HORUS - Done')

**Output:**

**Practical 3: Utilities and Auditing**

1. **Fixers Utilities:**

**Fixers enable your solution to take your existing data and fix a specific quality issue.**

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

import string

import datetime as dt

**# 1 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,'<')

**# 2 Removing nonprintable characters from a data entry** print('#2 Removing nonprintable characters from a data entry') printable = set(string.printable)

baddata = "Data\x00Science with\x02 funny characters is \x10bad!!!" cleandata=''.join(filter(lambda x: x in string.printable,baddata)) print('Bad Data : ',baddata);

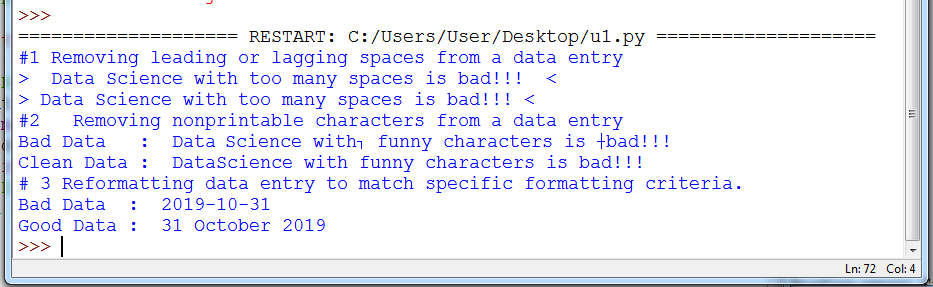
print('Clean Data : ',cleandata)

**# 3 Reformatting data entry to match specific formatting criteria**. # Convert YYYY/MM/DD to DD Month YYYY

print('# 3 Reformatting data entry to match specific formatting criteria.') baddate = dt.date(2019, 10, 31)

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

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

print('Bad Data : ',baddata) print('Good Data : ',gooddata) **Output:**

1. **Data Binning or Bucketing**

Binning is a data preprocessing technique used to reduce the effects of minor observation errors. Statistical data binning is a way to group a number of more or less continuous values into a smaller number of “bins.”

**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')

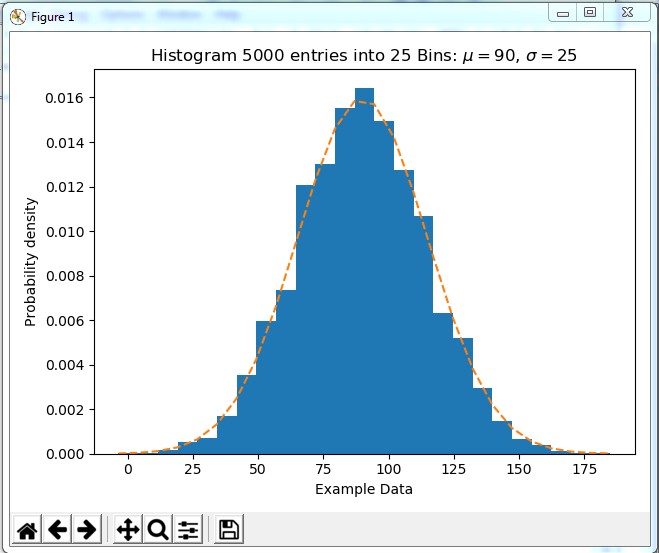
sTitle=r'Histogram ' + str(len(x)) + ' entries into ' + str(num\_bins) + ' Bins: $\mu=' + str(mu)

+ '$, $\sigma=' + str(sigma) + '$' ax.set\_title(sTitle) fig.tight\_layout()

sPathFig='C:/VKHCG/05-DS/4000-UL/0200-DU/DU-Histogram.png' fig.savefig(sPathFig)

plt.show()

**Output:**



1. **Averaging of Data**

The use of averaging of features value enables the reduction of data volumes in a control fashion to improve effective data processing.

C:\VKHCG\05-DS\4000-UL\0200-DU\DU-Mean.py

**Code:**

import pandas as pd ################################################################

InputFileName='IP\_DATA\_CORE.csv' OutputFileName='Retrieve\_Router\_Location.csv' Base='C:/VKHCG' print('################################')

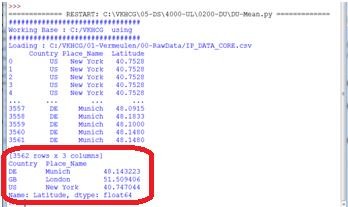
print('Working Base :',Base, ' using ') print('################################')

sFileName=Base + '/01-Vermeulen/00-RawData/' + InputFileName print('Loading :',sFileName) IP\_DATA\_ALL=pd.read\_csv(sFileName,header=0,low\_memory=False, usecols=['Country','Place Name','Latitude','Longitude'], encoding="latin-1") IP\_DATA\_ALL.rename(columns={'Place Name': 'Place\_Name'}, inplace=True) AllData=IP\_DATA\_ALL[['Country', 'Place\_Name','Latitude']]

print(AllData)

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

**Output:**



**Outlier Detection**

Outliers are data that is so different from the rest of the data in the data set that it may be caused by an error in the data source. There is a technique called outlier detection that, with good data science, will identify these outliers.

C:\VKHCG\05-DS\4000-UL\0200-DU\DU-Outliers.py

**Code:**

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

# -\*- coding: utf-8 -\*- ################################################################

import pandas as pd ################################################################

InputFileName='IP\_DATA\_CORE.csv' OutputFileName='Retrieve\_Router\_Location.csv' Base='C:/VKHCG' print('################################')

print('Working Base :',Base) print('################################')

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

sFileName=Base + '/01-Vermeulen/00-RawData/' + InputFileName print('Loading :',sFileName) IP\_DATA\_ALL=pd.read\_csv(sFileName,header=0,low\_memory=False, usecols=['Country','Place Name','Latitude','Longitude'], encoding="latin-1") IP\_DATA\_ALL.rename(columns={'Place Name': 'Place\_Name'}, inplace=True) LondonData=IP\_DATA\_ALL.loc[IP\_DATA\_ALL['Place\_Name']=='London'] AllData=LondonData[['Country', 'Place\_Name','Latitude']]

print('All Data') print(AllData)

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

UpperBound=float(MeanData+StdData) print('Higher than ', UpperBound)

OutliersHigher=AllData[AllData.Latitude>UpperBound] print(OutliersHigher)

LowerBound=float(MeanData-StdData) print('Lower than ', LowerBound)

OutliersLower=AllData[AllData.Latitude<LowerBound] print(OutliersLower)

print('Not Outliers') OutliersNot=AllData[(AllData.Latitude>=LowerBound) & (AllData.Latitude<=UpperBound)]

print(OutliersNot) ################################################################

**Output:**

=========== RESTART: C:\VKHCG\05-DS\4000-UL\0200-DU\DU-Outliers.py

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

Working Base : C:/VKHCG ################################

Loading : C:/VKHCG/01-Vermeulen/00-RawData/IP\_DATA\_CORE.csv All Data

Country Place\_Name Latitude

|  |  |  |  |
| --- | --- | --- | --- |
| 1910 | GB | London | 51.5130 |
| 1911 | GB | London | 51.5508 |
| 1912 | GB | London | 51.5649 |
| 1913 | GB | London | 51.5895 |
| 1914 | GB | London | 51.5232 |

... ... ... ...

[1502 rows x 3 columns] Outliers

Higher than 51.51263550786781 Country Place\_Name Latitude

1910 GB London 51.5130

|  |  |  |  |
| --- | --- | --- | --- |
| 1911 | GB | London | 51.5508 |
| 1912 | GB | London | 51.5649 |
| 1913 | GB | London | 51.5895 |
| 1914 | GB | London | 51.5232 |
| 1916 | GB | London | 51.5491 |
| 1919 | GB | London | 51.5161 |
| 1920 | GB | London | 51.5198 |
| 1921 | GB | London | 51.5198 |
| 1923 | GB | London | 51.5237 |
| 1924 | GB | London | 51.5237 |

Lower than 51.50617687562166 Country Place\_Name Latitude

1915 GB London 51.4739

Not Outliers

Country Place\_Name Latitude

1. **Logging**

**Write a Python / R program for basic logging in data science.**

C:\VKHCG\77-Yoke\Yoke\_Logging.py

**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)

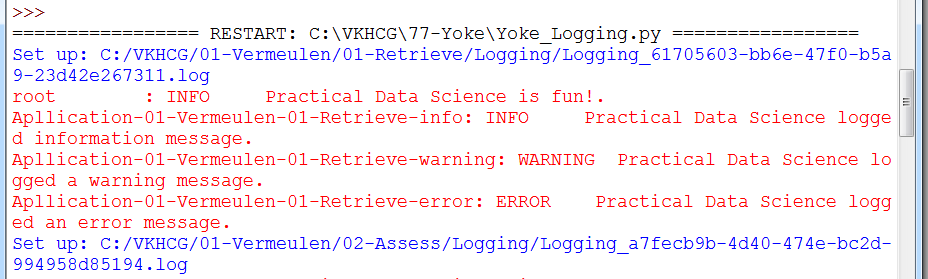
if sLevel == 'debug':

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 4**

1. Perform the following data processing using R.

Use R-Studio for the following:

>library(readr)

Warning message:package ‘readr’ was built under R version 3.4.4 Load a table named IP\_DATA\_ALL.csv.

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

Parsed with column specification: cols(

ID = col\_double(), Country = col\_character(),

`Place Name` = col\_character(),

`Post Code` = col\_double(), Latitude = col\_double(), Longitude = col\_double(),

`First IP Number` = col\_double(),

`Last IP Number` = col\_double()

)

>View(IP\_DATA\_ALL)

>spec(IP\_DATA\_ALL) cols(

ID = col\_double(), Country = col\_character(),

`Place Name` = col\_character(),

`Post Code` = col\_double(), Latitude = col\_double(), Longitude = col\_double(),

`First IP Number` = col\_double(),

`Last IP Number` = col\_double()

)

This informs you that you have the following eight columns:

* + ID of type integer
  + Place name of type character
  + Post code of type character
  + Latitude of type numeric double
  + Longitude of type numeric double
  + First IP number of type integer
  + Last IP number of type integer

>library(tibble)

>set\_tidy\_names(IP\_DATA\_ALL, syntactic = TRUE, quiet = FALSE) New names:

Place Name -> Place.Name Post Code -> Post.Code

First IP Number -> First.IP.Number Last IP Number -> Last.IP.Number

This informs you that four of the field names are not valid and suggests new field names that are valid.You can fix any detected invalid column names by executing IP\_DATA\_ALL\_FIX=set\_tidy\_names(IP\_DATA\_ALL, syntactic = TRUE, quiet = TRUE) By using command View(IP\_DATA\_ALL\_FIX), you can check that you have fixed the columns. The new table IP\_DATA\_ALL\_FIX.csv will fix the invalid column names with valid names.

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

ID Country Place.Name Post.Code Latitude "double" "character" "character" "double" "double"

Longitude First.IP.Number Last.IP.Number "double" "double" "double"

>library(data.table)

>hist\_country=data.table(Country=unique(IP\_DATA\_ALL\_FIX[is.na(IP\_DATA\_ALL\_FIX ['Country']) == 0, ]$Country))

>setorder(hist\_country,'Country')

>hist\_country\_with\_id=rowid\_to\_column(hist\_country, var = "RowIDCountry")

>View(hist\_country\_fix)

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

>View(IP\_DATA\_COUNTRY\_FREQ)

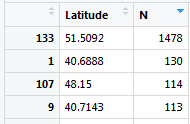


* + The two biggest subset volumes are from the US and GB.
  + The US has just over four times the data as GB.

hist\_latitude =data.table(Latitude=unique(IP\_DATA\_ALL\_FIX [is.na(IP\_DATA\_ALL\_with\_ID ['Latitude']) == 0, ]$Latitude)) setkeyv(hist\_latitude, 'Latitude')

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

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



* + The two biggest data volumes are from latitudes 51.5092 and 40.6888.
  + The spread appears to be nearly equal between the top-two latitudes.

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

Latitude 40.6888

What does this tell you?

Fact: The range of latitude for the Northern Hemisphere is from 0 to 90. So, if you do not have any latitudes farther south than 40.6888, you can improve your retrieve routine.

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

Minimum business frequency is from DE – Denmark.

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

51.5895

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

Country "US"

The result is 51.5895. What does this tell you?

Fact: The range in latitude for the Northern Hemisphere is from 0 to 90. So, if you do not have any latitudes more northerly than 51.5895, you can improve your retrieve routine.

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

46.69097

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

48.15

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

Latitude [1,] 40.6888

[2,] 51.5895

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

0% 40.6888

25% 40.7588

50% 48.1500

75% 51.5092

100% 51.5895

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

4.890387

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

38.01702

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

##### **C:\ VKHCG\01-Vermeulen\01-Retrieve\Retrive\_IP\_DATA\_ALL.py**### import sys

import os

import pandas as pd ################################################################ Base='C:/VKHCG' ################################################################

sFileName=Base + '/01-Vermeulen/00-RawData/IP\_DATA\_ALL.csv' print('Loading :',sFileName)

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

sFileDir=Base + '/01-Vermeulen/01-Retrieve/01-EDS/02-Python' if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

print('Rows:', IP\_DATA\_ALL.shape[0]) print('Columns:', IP\_DATA\_ALL.shape[1])

print('### Raw Data Set #####################################')

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

print('### Fixed Data Set ###################################')

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

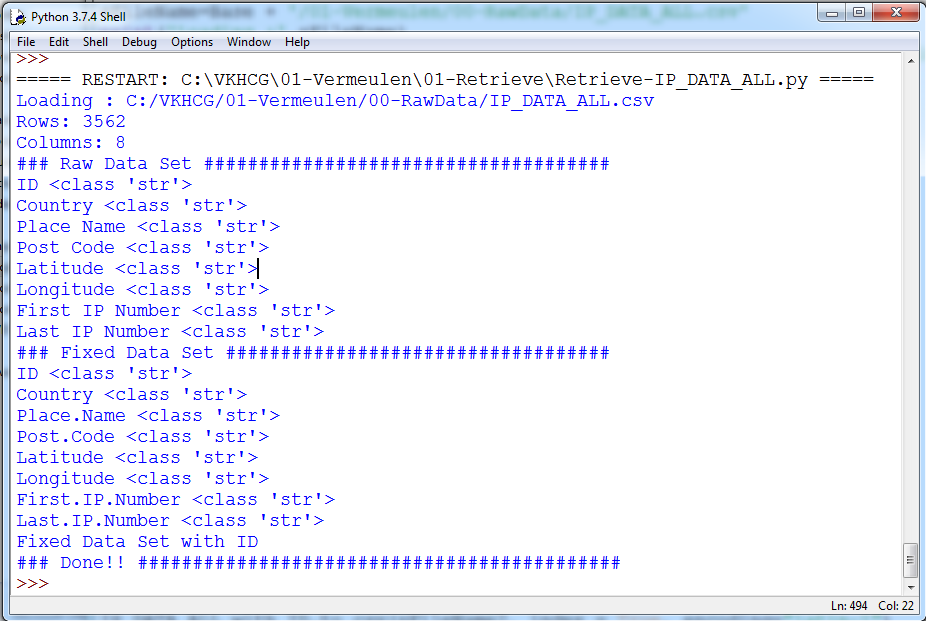
for i in range(0,len(IP\_DATA\_ALL.columns)): cNameOld=IP\_DATA\_ALL\_FIX.columns[i] + ' ' cNameNew=cNameOld.strip().replace(" ", ".") IP\_DATA\_ALL\_FIX.columns.values[i] = cNameNew print(IP\_DATA\_ALL.columns[i],type(IP\_DATA\_ALL.columns[i]))

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

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

print('Fixed Data Set with ID') IP\_DATA\_ALL\_with\_ID=IP\_DATA\_ALL\_FIX IP\_DATA\_ALL\_with\_ID.index.names = ['RowID'] #print(IP\_DATA\_ALL\_with\_ID.head()) sFileName2=sFileDir + '/Retrieve\_IP\_DATA.csv'

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



1. Data Pattern

To determine a pattern of the data values, Replace all alphabet values with an uppercase case *A*, all numbers with an uppercase *N*, and replace any spaces with a lowercase letter *b* and all other unknown characters with a lowercase *u*. As a result, “Good Book 101” becomes “AAAAbAAAAbNNNu.”This pattern creation is beneficial for designing any specific assess rules. This pattern view of data is a quick way to identify common patterns or determine standard layouts.

library(readr) library(data.table)

FileName=paste0('c:/VKHCG/01-Vermeulen/00-RawData/IP\_DATA\_ALL.csv') IP\_DATA\_ALL <- read\_csv(FileName) hist\_country=data.table(Country=unique(IP\_DATA\_ALL$Country)) pattern\_country=data.table(Country=hist\_country$Country,

PatternCountry=hist\_country$Country) oldchar=c(letters,LETTERS) newchar=replicate(length(oldchar),"A")

for (r in seq(nrow(pattern\_country))){ s=pattern\_country[r,]$PatternCountry; for (c in seq(length(oldchar))){ s=chartr(oldchar[c],newchar[c],s)

};

for (n in seq(0,9,1)){ s=chartr(as.character(n),"N",s)

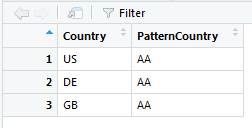
};

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

s=chartr(".","u",s) pattern\_country[r,]$PatternCountry=s;

};

View(pattern\_country)



**Example 2:** This is a common use of patterns to separate common standards and structures. Pattern can be loaded in separate retrieve procedures. If the same two patterns, NNNNuNNuNN and uuNNuNNuNN, are found, you can send NNNNuNNuNN directly to be converted into a date, while uuNNuNNuNN goes through a quality-improvement process to then route back to the same queue as NNNNuNNuNN, once it complies.

library(readr) library(data.table) Base='C:/VKHCG'

FileName=paste0(Base,'/01-Vermeulen/00-RawData/IP\_DATA\_ALL.csv') IP\_DATA\_ALL <- read\_csv(FileName) hist\_latitude=data.table(Latitude=unique(IP\_DATA\_ALL$Latitude))

pattern\_latitude=data.table(latitude=hist\_latitude$Latitude, Patternlatitude=as.character(hist\_latitude$Latitude))

oldchar=c(letters,LETTERS) newchar=replicate(length(oldchar),"A") for (r in seq(nrow(pattern\_latitude))){ s=pattern\_latitude[r,]$Patternlatitude; for (c in seq(length(oldchar))){

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

};

for (n in seq(0,9,1)){ s=chartr(as.character(n),"N",s)

};

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

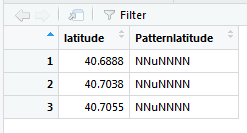
s=chartr("+","u",s)

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

s=chartr(".","u",s) pattern\_latitude[r,]$Patternlatitude=s;

};

setorder(pattern\_latitude,latitude) View(pattern\_latitude[1:3])



1. Loading IP\_DATA\_ALL:

This data set contains all the IP address allocations in the world. It will help you to locateyour customers when interacting with them online.

Create a new Python script file and save it as Retrieve-IP\_DATA\_ALL.py in directory C:\VKHCG\01-Vermeulen\01-Retrieve.

##############Retrieve-IP\_DATA\_ALL.py########################

# -\*- coding: utf-8 -\*- ################################################################

import sys import os

import pandas as pd ################################################################ Base='C:/VKHCG' ################################################################

sFileName=Base + '/01-Vermeulen/00-RawData/IP\_DATA\_ALL.csv' print('Loading :',sFileName)

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

sFileDir=Base + '/01-Vermeulen/01-Retrieve/01-EDS/02-Python' if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

print('Rows:', IP\_DATA\_ALL.shape[0]) print('Columns:', IP\_DATA\_ALL.shape[1])

print('### Raw Data Set #####################################')

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

print('### Fixed Data Set ###################################')

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

for i in range(0,len(IP\_DATA\_ALL.columns)): cNameOld=IP\_DATA\_ALL\_FIX.columns[i] + ' ' cNameNew=cNameOld.strip().replace(" ", ".") IP\_DATA\_ALL\_FIX.columns.values[i] = cNameNew print(IP\_DATA\_ALL.columns[i],type(IP\_DATA\_ALL.columns[i]))

#print(IP\_DATA\_ALL\_FIX.head()) print('Fixed Data Set with ID')

IP\_DATA\_ALL\_with\_ID=IP\_DATA\_ALL\_FIX IP\_DATA\_ALL\_with\_ID.index.names = ['RowID'] #print(IP\_DATA\_ALL\_with\_ID.head())

sFileName2=sFileDir + '/Retrieve\_IP\_DATA.csv' IP\_DATA\_ALL\_with\_ID.to\_csv(sFileName2, index = True, encoding="latin-1") ################################################################ print('### Done!! ############################################') ################################################################



Start your Python editor and create a text file named Retrieve-IP\_Routing.py in directory. C:\VKHCG\01-Vermeulen\01-Retrieve. ################################################################

# -\*- coding: utf-8 -\*- ################################################################

import sys import os

import pandas as pd

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

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

"""

Calculate the great circle distance between two points on the earth (specified in decimal degrees)

"""

# convert decimal degrees to radians

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

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 # Radius of earth in kilometers else:

r = 3956 # Radius of earth in miles d=round(c \* r,3)

return d ################################################################ Base='C:/VKHCG' ################################################################

sFileName=Base + '/01-Vermeulen/00-RawData/IP\_DATA\_CORE.csv' print('Loading :',sFileName) 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(

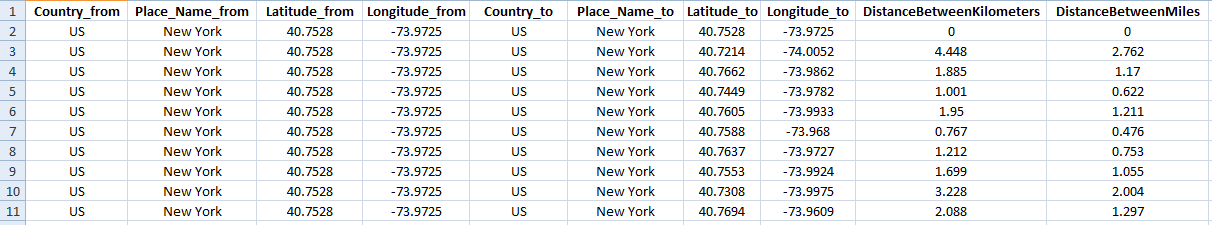
row['Longitude\_from'], row['Latitude\_from'], row['Longitude\_to'], row['Latitude\_to'], 'miles')

,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:**

See the file named Retrieve\_IP\_Routing.csv in C:\VKHCG\01-Vermeulen\01-Retrieve\01- EDS\02-Python.



**Total Records: 22501**

So, the distance between a router in New York (40.7528, -73.9725) to anoher router in New York (40.7214, -74.0052) is 4.448 kilometers, or 2.762 miles.

Building a Diagram for the Scheduling of Jobs

Start your Python editor and create a text file named Retrieve-Router-Location.py in directory.

C:\VKHCG\01-Vermeulen\01-Retrieve.

################### Retrieve-Router-Location.py ###################### # -\*- coding: utf-8 -\*- ################################################################

import sys import os

import pandas as pd ################################################################

InputFileName='IP\_DATA\_CORE.csv' OutputFileName='Retrieve\_Router\_Location.csv' ################################################################ Base='C:/VKHCG'

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

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])

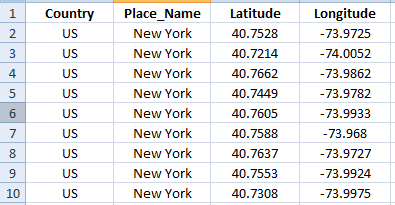
print('Columns :',ROUTERLOC.shape[1])

sFileName2=sFileDir + '/' + OutputFileName ROUTERLOC.to\_csv(sFileName2, index = False, encoding="latin-1") ################################################################ print('### Done!! ############################################') ################################################################

**Output:**



See the file named Retrieve\_Router\_Location.csv in C:\VKHCG\01-Vermeulen\01-Retrieve\01-EDS\02-Python.



Krennwallner AG

The company has two main jobs in need of your attention:

* + *Picking content for billboards*: I will guide you through the data science required to pick advertisements for each billboard in the company.
  + *Understanding your online visitor data*: I will guide you through the evaluation of the web traffic to the billboard’s online web servers.

Picking Content for Billboards

Start your Python editor and create a text file named Retrieve-DE-Billboard-Locations.py in directory.

C:\VKHCG\02-Krennwallner\01-Retrieve.

################# Retrieve-DE-Billboard-Locations.py ############### # -\*- coding: utf-8 -\*- ################################################################

import sys import os

import pandas as pd ################################################################

InputFileName='DE\_Billboard\_Locations.csv' OutputFileName='Retrieve\_DE\_Billboard\_Locations.csv' Company='02-Krennwallner'

################################################################ Base='C:/VKHCG'

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

print('Working Base :',Base, ' using ', sys.platform) print('################################') ################################################################ Base='C:/VKHCG'

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):

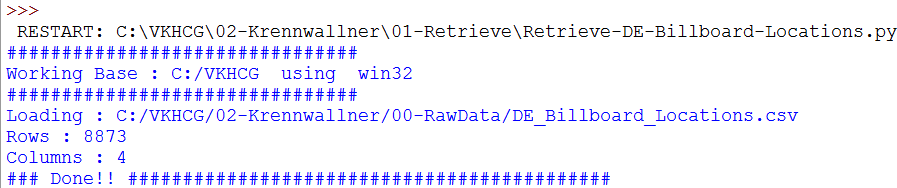
os.makedirs(sFileDir)

ROUTERLOC = IP\_DATA\_ALL.drop\_duplicates(subset=None, keep='first', inplace=False) print('Rows :',ROUTERLOC.shape[0])

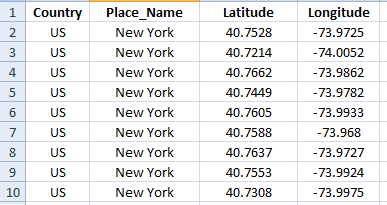
print('Columns :',ROUTERLOC.shape[1])

sFileName2=sFileDir + '/' + OutputFileName ROUTERLOC.to\_csv(sFileName2, index = False)

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



See the file named Retrieve\_Router\_Location.csv in C:\VKHCG\02-Krennwallner\01-Retrieve\01-EDS\02-Python.



Understanding Your Online Visitor Data

Let’s retrieve the visitor data for the billboard we have in Germany.

Several times it was found that common and important information is buried somewhere in the company’s various data sources. Investigating any direct suppliers or consumers’ upstream or downstream data sources attached to the specific business process is necessary. That is part of your skills that you are applying to data science. Numerous insightful fragments of information was found in the data sources surrounding a customer’s business processes.

Start your Python editor and create a file named Retrieve-Online-Visitor.py in directory C:\VKHCG\02-Krennwallner\01-Retrieve. ################################################################

# -\*- coding: utf-8 -\*- ################################################################

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)

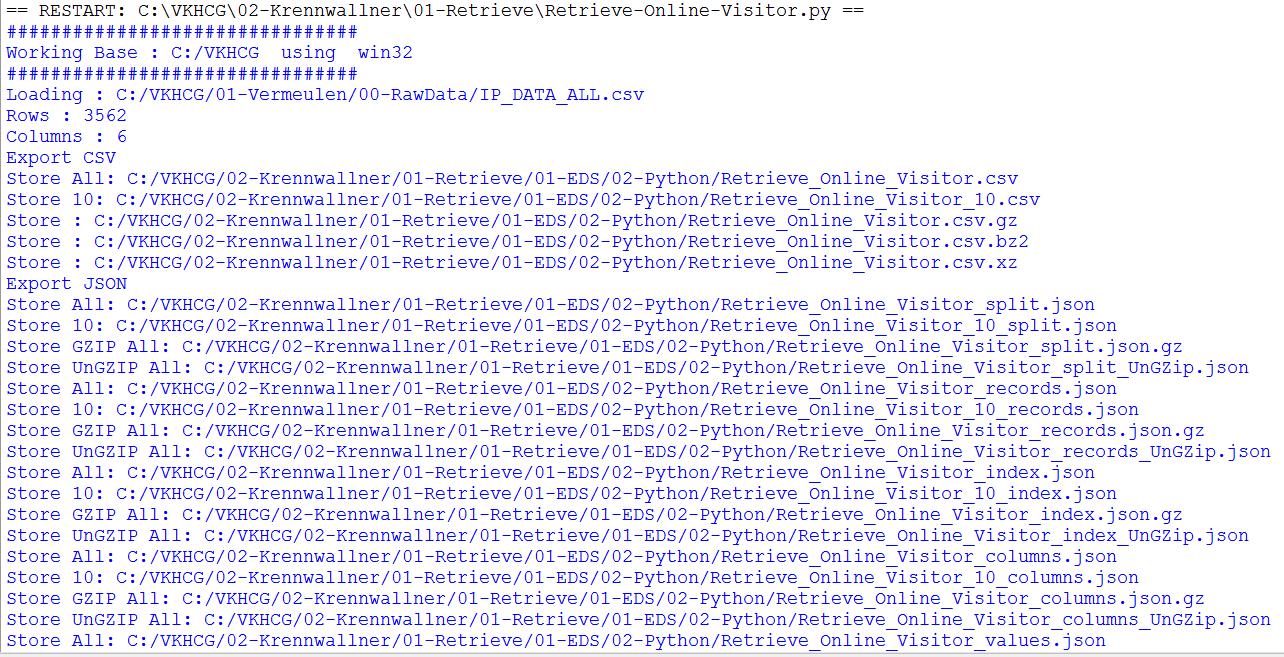
sFileName5=sFileDir + '/' + OutputFileName + '\_' + sOrient + '\_UnGZip.json' file\_in = gz.open(sFileName4, 'rb')

file\_out = open(sFileName5, 'wb') file\_out.writelines(file\_in) file\_in.close()

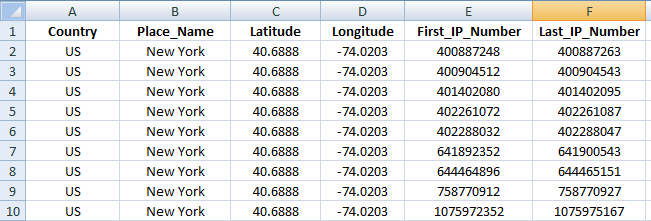
file\_out.close()

print('Store UnGZIP All:',sFileName5) ################################################################ print('### Done!! ############################################') ################################################################

**Output:**



See the file named Retrieve\_Online\_Visitor.csv in C:\VKHCG\02-Krennwallner\01-Retrieve\01-EDS\02-Python.



You can also see the following JSON files of only ten records. XML processing.

Start Python editor and create a file named Retrieve-Online-Visitor-XML.py indirectory C:\VKHCG\02-Krennwallner\01-Retrieve. ################################################################

# -\*- coding: utf-8 -\*- ################################################################

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.XML(xml\_data) all\_records = []

for i, child in enumerate(root): record = {}

for subchild in child: record[subchild.tag] = subchild.text

all\_records.append(record) return pd.DataFrame(all\_records)

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

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='C:/VKHCG' ################################################################ print('################################')

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

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')

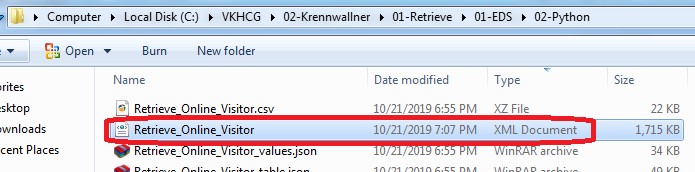
print('Rows :',unxmldata.shape[0]) print('Columns :',unxmldata.shape[1]) print(unxmldata)

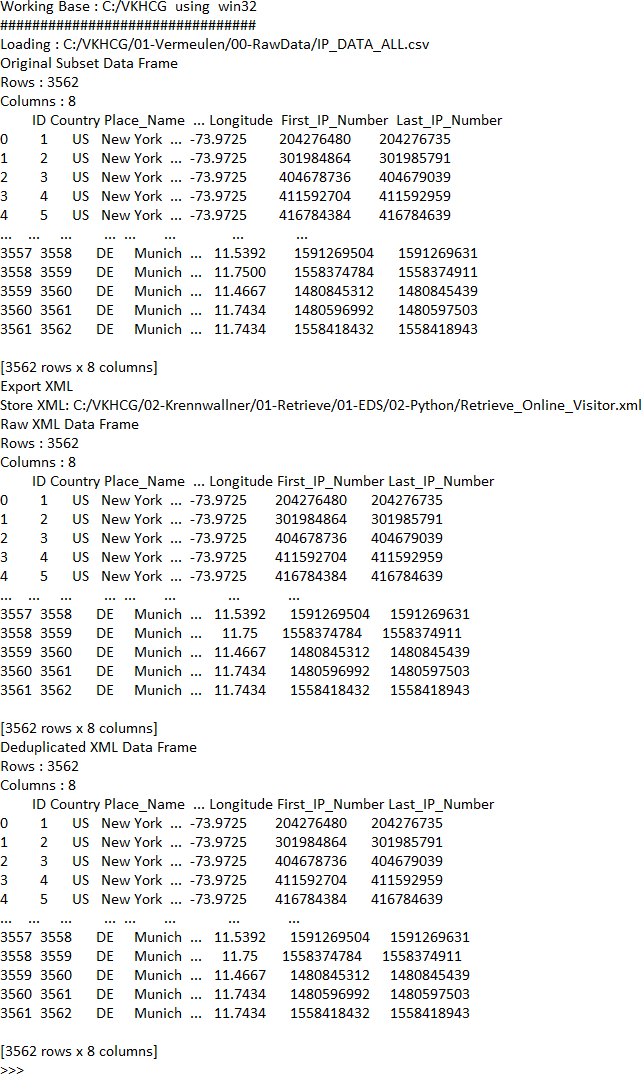
#print('### Done!! ############################################')

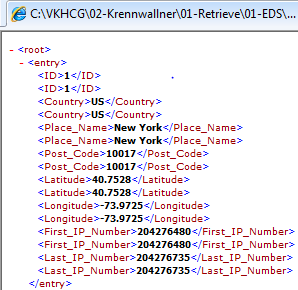
**Output:**

See a file named Retrieve\_Online\_Visitor.xml in C:\VKHCG\02-Krennwallner\01-Retrieve\01-EDS\02-Python.

This enables you to deliver XML format data as part of the retrieve step.







Hillman Ltd

Start yourPython editor and create a file named Retrieve-Incoterm-EXW.py in directory C:\VKHCG\03-Hillman\01-Retrieve.

import os

import sys

import pandas as pd IncoTerm='EXW' InputFileName='Incoterm\_2010.csv'

OutputFileName='Retrieve\_Incoterm\_' + IncoTerm + '\_RuleSet.csv' Company='03-Hillman' ################################################################ Base='C:/VKHCG'

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) ################################################################

### Import Incoterms ################################################################

sFileName=Base + '/' + Company + '/00-RawData/' + InputFileName print('###########')

print('Loading :',sFileName) IncotermGrid=pd.read\_csv(sFileName,header=0,low\_memory=False) IncotermRule=IncotermGrid[IncotermGrid.Shipping\_Term == IncoTerm] print('Rows :',IncotermRule.shape[0])

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

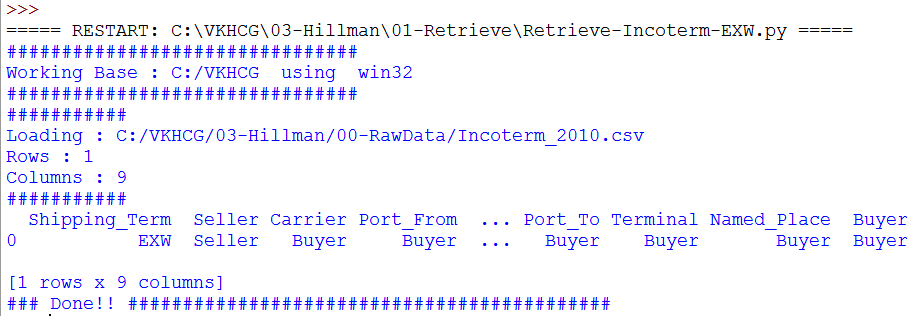
print(IncotermRule)

sFileName=sFileDir + '/' + OutputFileName IncotermRule.to\_csv(sFileName, index = False)

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

**Output**

See the file named Retrieve\_Incoterm\_EXW.csv in C:\VKHCG\03-Hillman\01-Retrieve\01- EDS\02-Python. Open this file,



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

import os import sys

import pandas as pd ################################################################

IncoTerm='FCA' InputFileName='Incoterm\_2010.csv'

OutputFileName='Retrieve\_Incoterm\_' + IncoTerm + '\_RuleSet.csv' Company='03-Hillman' ################################################################ Base='C:/VKHCG'

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) ################################################################

### Import Incoterms ################################################################

sFileName=Base + '/' + Company + '/00-RawData/' + InputFileName print('###########')

print('Loading :',sFileName) IncotermGrid=pd.read\_csv(sFileName,header=0,low\_memory=False) IncotermRule=IncotermGrid[IncotermGrid.Shipping\_Term == IncoTerm] print('Rows :',IncotermRule.shape[0])

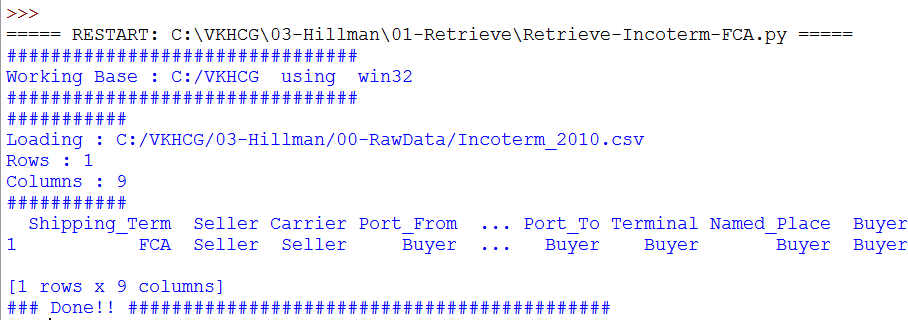
print('Columns :',IncotermRule.shape[1]) print('###########')

print(IncotermRule)

sFileName=sFileDir + '/' + OutputFileName IncotermRule.to\_csv(sFileName, index = False)

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

**Output:**



**CPT—Carriage Paid To (Named Place of Destination)**

C:\VKHCG\03-Hillman\01-Retrieve.

**CIP—Carriage and Insurance Paid To (Named Place of Destination)**

**DAT—Delivered at Terminal (Named Terminal at Port or Place of Destination) DAP—Delivered at Place (Named Place of Destination)**

**DDP—Delivered Duty Paid (Named Place of Destination)**

By this term, the seller is responsible for delivering the goods to the named place in the country of the buyer and pays all costs in bringing the goods to the destination, including import duties and taxes. The seller is not responsible for unloading. This term places the maximum obligations on the seller and minimum obligations on the buyer. No risk or responsibility is transferred to the buyer until delivery of the goods at the named place of destination.

**Possible Shipping Routes**

There are numerous potential shipping routes available to the company. The retrieve step can generate the potential set, by using a route combination generator. This will give you a set of routes, but it is highly unlikely that you will ship along all of them. It is simply a population of routes that can be used by the data science to find the optimum solution.

Start your Python editor and create a file named Retrieve-Warehouse-Incoterm-Chains.py in directory C:\VKHCG\03-Hillman\01-Retrieve.

**Adopt New Shipping Containers**

Adopting the best packing option for shipping in containers will require that I introduce a new concept. Shipping of containers is based on a concept reducing the packaging you use down to an optimum set of sizes having the following requirements:

* + The product must fit within the box formed by the four sides of a cube.
  + The product can be secured using packing foam, which will fill any void volume in the packaging.
  + Packaging must fit in shipping containers with zero space gaps.
  + Containers can only hold product that is shipped to a single warehouse, shop, or customer.

Start your Python editor and create a text file named Retrieve-Container-Plan.py in directory . C:\VKHCG\03-Hillman\01-Retrieve.

**\*\*\* Replace pd.DataFrame.from\_items with pd.DataFrame.from\_dict**

import sys import os

import pandas as pd ################################################################

ContainerFileName='Retrieve\_Container.csv' BoxFileName='Retrieve\_Box.csv' ProductFileName='Retrieve\_Product.csv' Company='03-Hillman'

################################################################ Base='C:/VKHCG' ################################################################ 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) ################################################################

### Create the Containers ################################################################

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) ################################################################

## Create valid Product ################################################################

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

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

sFileProductName=sFileDir + '/' + ProductFileName ProductFrame.to\_csv(sFileProductName, index = False) ################################################################ ################################################################# print('### Done!! ############################################') #################################################################

**Output:**



Your second simulation is the cardboard boxes for the packing of the products. The requirement is for boxes having a dimension of 100 centimeters × 100 centimeters × 100 centimeters to 2.1 meters × 2.1 meters × 2.1 meters. You can also use between zero and 600 centimeters of packing foam to secure any product in the box.

See the container data file Retrieve\_Container.csv and Retrieve\_Box.csv in C:\VKHCG\03-Hillman\01-Retrieve\01-EDS\02-Python.

Create a Delivery Route

The model enables you to generate a complex routing plan for the shipping routes of the company. Start your Python editor and create a text file named Retrieve-Route-Plan.py in directory .

C:\VKHCG\03-Hillman\01-Retrieve.

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'

################################################################ Base='C:/VKHCG'

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) ################################################################

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)

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

**Output:**

====== RESTART: C:\VKHCG\03-Hillman\01-Retrieve\Retrieve-Route-Plan.py ====== ################################

Working Base : C:/VKHCG using win32 ################################ ###########

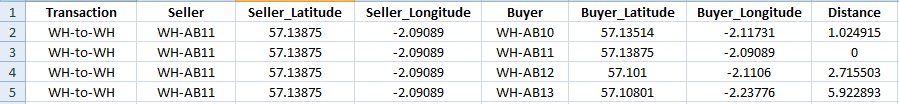
Loading : C:/VKHCG/03-Hillman/00-RawData/GB\_Postcode\_Warehouse.csv Run : 0 =======>>>>>>>>>> AB10

Run : 1 =======>>>>>>>>>> AB11 Run : 2 =======>>>>>>>>>> AB12 Run : 3 =======>>>>>>>>>> AB13 Run : 4 =======>>>>>>>>>> AB14 Run : 5 =======>>>>>>>>>> AB15 Run : 6 =======>>>>>>>>>> AB16 Run : 7 =======>>>>>>>>>> AB21 Run : 8 =======>>>>>>>>>> AB22 Run : 9 =======>>>>>>>>>> AB23 Run : 10 =======>>>>>>>>>> AB24 Run : 19 =======>>>>>>>>>> AB37

### Done!! ############################################

>>>

See the collection of files similar in format to Retrieve\_Route\_WH-AB11\_Route.csv in C:\VKHCG\03-Hillman\01-Retrieve\01-EDS\02-Python.



Global Post Codes

Open RStudio and use R to process the following R script: Retrieve-Postcode-Global.r.

library(readr)

All\_Countries <- read\_delim("C:/VKHCG/03-Hillman/00-RawData/All\_Countries.txt", "\t", col\_names = FALSE,

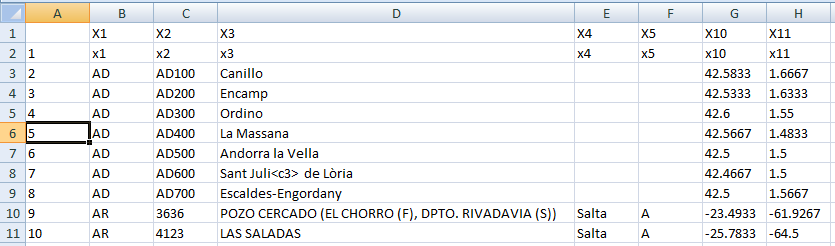
col\_types = cols( X12 = col\_skip(), X6 = col\_skip(), X7 = col\_skip(), X8 = col\_skip(), X9 = col\_skip()),

na = "null", trim\_ws = TRUE) write.csv(All\_Countries,

file = "C:/VKHCG/03-Hillman/01-Retrieve/01-EDS/01- R/Retrieve\_All\_Countries.csv")

**Output:**

The program will successfully uploaded a new file named Retrieve\_All\_Countries.csv, after removing column No. 6, 7, 8, 9 and 12 from All\_Countries.txt



Clark Ltd

Clark is the financial powerhouse of the group. It must process all the money-related data sources.

**Forex**-The first financial duty of the company is to perform any foreign exchange trading.

**Forex Base Data-**Previously, you found a single data source (Euro\_ExchangeRates.csv) for forex rates in Clark. Earlier in the chapter, I helped you to create the load, as part of your R processing.

The relevant file is Retrieve\_Retrieve\_Euro\_ExchangeRates.csv in directory C:\ VKHCG\04-Clark\01-Retrieve\01-EDS\01-R. So, that data is ready.

**Financials -** Clark generates the financial statements for all the group’s companies. **Financial Base Data** - You found a single data source (Profit\_And\_Loss.csv) in Clark for financials and, as mentioned previously, a single data source (Euro\_ExchangeRates.csv) for forex rates. The file relevant file is Retrieve\_Profit\_And\_Loss.csv in directory

C:\VKHCG\04-Clark\01-Retrieve\ 01-EDS\01-R.

**Person Base Data**

Start Python editor and create a file named Retrieve-PersonData.py in directory . C:\VKHCG\04-Clark\01-Retrieve.

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

# -\*- coding: utf-8 -\*-

import sys import os import shutil import zipfile

import pandas as pd Base='C:/VKHCG'

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

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

Company='04-Clark'

ZIPFiles=['Data\_female-names','Data\_male-names','Data\_last-names'] for ZIPFile in ZIPFiles:

InputZIPFile=Base+'/'+Company+'/00-RawData/' + ZIPFile + '.zip' OutputDir=Base+'/'+Company+'/01-Retrieve/01-EDS/02-Python/' + ZIPFile OutputFile=Base+'/'+Company+'/01-Retrieve/01-EDS/02-Python/Retrieve-'+ZIPFile+'.csv' zip\_file = zipfile.ZipFile(InputZIPFile, 'r')

zip\_file.extractall(OutputDir) zip\_file.close()

t=0

for dirname, dirnames, filenames in os.walk(OutputDir): for filename in filenames:

sCSVFile = dirname + '/' + filename t=t+1

if t==1:

NameRawData=pd.read\_csv(sCSVFile,header=None,low\_memory=False)

NameData=NameRawData else:

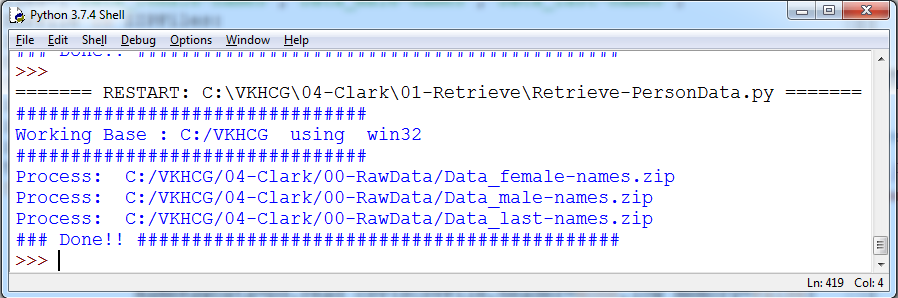
NameRawData=pd.read\_csv(sCSVFile,header=None,low\_memory=False) NameData=NameData.append(NameRawData)

NameData.rename(columns={0 : 'NameValues'},inplace=True) NameData.to\_csv(OutputFile, index = False) shutil.rmtree(OutputDir)

print('Process: ',InputZIPFile)

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

This generates three files named Retrieve-Data\_female-names.csv Retrieve-Data\_male-names.csv Retrieve-Data\_last-names.csv



**Connecting to other Data Sources**

1. **Program to connect to different data sources.**

SQLite:

# -\*- coding: utf-8 -\*- import sqlite3 as sq import pandas as pd Base='C:/VKHCG'

sDatabaseName=Base + '/01-Vermeulen/00-RawData/SQLite/vermeulen.db' conn = sq.connect(sDatabaseName)

sFileName='C:/VKHCG/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('################')

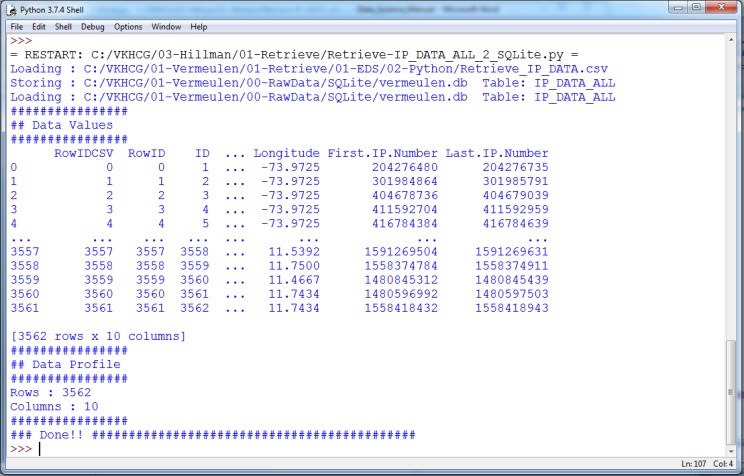
print('## Data Values') print('################')

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

print('## Data Profile') print('################')

print('Rows :',TestData.shape[0]) print('Columns :',TestData.shape[1]) print('################')

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



**MySQL:**

Open MySql

Create a database “DataScience”

Create a python file and add the following code:

################ Connection With MySQL ######################

import mysql.connector

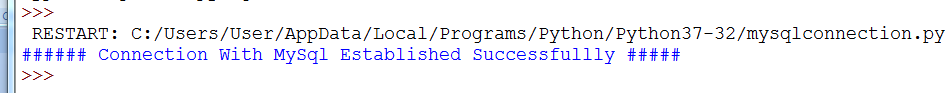
conn = mysql.connector.connect(host='localhost',

database='DataScience', user='root', password='root')

conn.connect if(conn.is\_connected):

print('###### Connection With MySql Established Successfullly ##### ') else:

print('Not Connected -- Check Connection Properites')



Microsoft Excel

##################Retrieve-Country-Currency.py ################################################################

# -\*- coding: utf-8 -\*- ################################################################

import os

import pandas as pd ################################################################ Base='C:/VKHCG' ################################################################

sFileDir=Base + '/01-Vermeulen/01-Retrieve/01-EDS/02-Python' #if not os.path.exists(sFileDir):

#os.makedirs(sFileDir) ################################################################

CurrencyRawData = pd.read\_excel('C:/VKHCG/01-Vermeulen/00- RawData/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 from Excel Sheet Retrived Successfully ~~~~~~~ ') ################################################################

sFileName=sFileDir + '/Retrieve-Country-Currency.csv' CurrencyData.to\_csv(sFileName, index = False) ################################################################ **OUTPUT:**



Assess Superstep

**Practical 5: Assessing Data**

Data quality refers to the condition of a set of qualitative or quantitative variables. Dataquality is a multidimensional measurement of the acceptability of specific data sets. Inbusiness, data quality is measured to determine whether data can be used as a basis forreliable intelligence extraction for supporting organizational decisions. Data profiling involves observing in your data sources all the viewpoints that theinformation offers. The main goal is to determine if individual viewpoints are accurateand complete. The Assess superstep determines what additional processing to apply tothe entries that are noncompliant.

Errors

Typically, one of four things can be done with an error to the data.

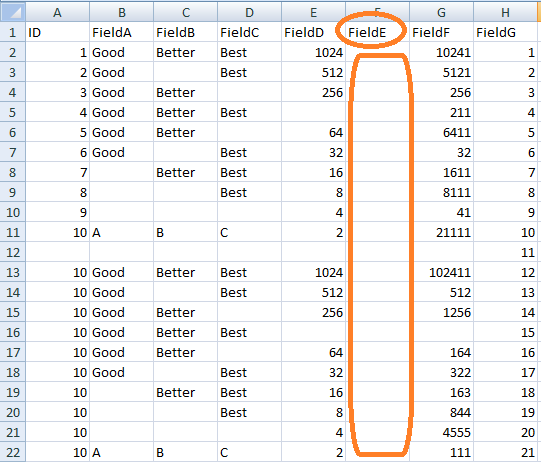
* 1. Accept the Error
  2. Reject the Error
  3. Correct the Error
  4. Create a Default Value
     1. **Perform error management on the given data using pandas package.**

Python pandas package enables several automatic error-management features.

**File Location:** C:\VKHCG\01-Vermeulen\02-Assess

## Missing Values in Pandas:

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



**Code :**

################### Assess-Good-Bad-01.py########################

# -\*- coding: utf-8 -\*- ################################################################

import sys import os

import pandas as pd ################################################################ Base='C:/VKHCG' ################################################################ print('################################')

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

sInputFileName='Good-or-Bad.csv' sOutputFileName='Good-or-Bad-01.csv'

Company='01-Vermeulen' ################################################################ Base='C:/VKHCG' ################################################################

sFileDir=Base + '/' + Company + '/02-Assess/01-EDS/02-Python' if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

### Import Warehouse ################################################################

sFileName=Base + '/' + Company + '/00-RawData/' + sInputFileName print('Loading :',sFileName) RawData=pd.read\_csv(sFileName,header=0) print('################################')

print('## Raw Data Values') print('################################')

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

print('## Data Profile') print('################################')

print('Rows :',RawData.shape[0]) print('Columns :',RawData.shape[1]) print('################################')

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

sFileName=sFileDir + '/' + sInputFileName RawData.to\_csv(sFileName, index = False)

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

TestData=RawData.dropna(axis=1, how='all') ################################################################ print('################################')

print('## Test Data Values') print('################################')

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

print('## Data Profile') print('################################')

print('Rows :',TestData.shape[0]) print('Columns :',TestData.shape[1]) print('################################')

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

sFileName=sFileDir + '/' + sOutputFileName TestData.to\_csv(sFileName, index = False)

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

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

**Output:**

>>>

======= RESTART: C:\VKHCG\01-Vermeulen\02-Assess\Assess-Good-Bad-01.py

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

Working Base : C:/VKHCG using win32 ################################

Loading : C:/VKHCG/01-Vermeulen/00-RawData/Good-or-Bad.csv ################################

## Raw Data Values ################################

ID FieldA FieldB FieldC FieldD FieldE FieldF FieldG 0 1.0 Good Better Best 1024.0 NaN 10241.0 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 2.0 | Good | NaN | Best | 512.0 | NaN | 5121.0 | 2 |
| 2 3.0 | Good | Better | NaN | 256.0 | NaN | 256.0 | 3 |
| 3 4.0 | Good | Better | Best | NaN | NaN | 211.0 | 4 |
| 4 5.0 | Good | Better | NaN | 64.0 | NaN | 6411.0 | 5 |
| 5 6.0 | Good | NaN | Best | 32.0 | NaN | 32.0 | 6 |
| 6 7.0 | NaN | Better | Best | 16.0 | NaN | 1611.0 | 7 |
| 7 8.0 | NaN | NaN | Best | 8.0 | NaN | 8111.0 | 8 |
| 8 9.0 | NaN | NaN | NaN | 4.0 | NaN | 41.0 | 9 |

9 10.0 A B C 2.0 NaN 21111.0 10

10 NaN NaN NaN NaN NaN NaN NaN 11 11 10.0 Good Better Best 1024.0 NaN 102411.0 12

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 12 10.0 | Good | NaN | Best | 512.0 | NaN | 512.0 13 | |
| 13 10.0 | Good | Better | NaN | 256.0 | NaN | 1256.0 14 | |
| 14 10.0 | Good | Better | Best | NaN | NaN | NaN 15 | |
| 15 10.0 | Good | Better | NaN | 64.0 | NaN | 164.0 16 | |
| 16 10.0 | Good | NaN | Best | 32.0 | NaN 322.0 | | 17 |
| 17 10.0 | NaN | Better | Best | 16.0 | NaN 163.0 | | 18 |
| 18 10.0 | NaN | NaN | Best | 8.0 | NaN 844.0 | | 19 |
| 19 10.0 | NaN | NaN | NaN | 4.0 | NaN 4555.0 | | 20 |
| 20 10.0 | A | B | C 2.0 | NaN | 111.0 21 | |  |

All of column E has been deleted, owing to the fact that all values in that column were missing values/errors.

1. **Drop the Columns Where Any of the Elements Is Missing Values** ################## Assess-Good-Bad-02.py########################### import sys

import os

import pandas as pd Base='C:/VKHCG'

sInputFileName='Good-or-Bad.csv' sOutputFileName='Good-or-Bad-02.csv' Company='01-Vermeulen'

################################################################ Base='C:/VKHCG' ################################################################ print('################################')

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

sFileDir=Base + '/' + Company + '/02-Assess/01-EDS/02-Python' if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

### Import Warehouse ################################################################

sFileName=Base + '/' + Company + '/00-RawData/' + sInputFileName print('Loading :',sFileName) RawData=pd.read\_csv(sFileName,header=0) print('################################')

print('## Raw Data Values') print('################################')

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

print('## Data Profile') print('################################')

print('Rows :',RawData.shape[0]) print('Columns :',RawData.shape[1]) print('################################')

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

sFileName=sFileDir + '/' + sInputFileName RawData.to\_csv(sFileName, index = False)

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

TestData=RawData.dropna(axis=1, how='any') ################################################################ print('################################')

print('## Test Data Values') print('################################')

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

print('## Data Profile') print('################################')

print('Rows :',TestData.shape[0]) print('Columns :',TestData.shape[1]) print('################################')

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

sFileName=sFileDir + '/' + sOutputFileName TestData.to\_csv(sFileName, index = False)

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

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

>>>

======= RESTART: C:\VKHCG\01-Vermeulen\02-Assess\Assess-Good-Bad-02.py

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

Working Base : C:/VKHCG using win32 ################################

Loading : C:/VKHCG/01-Vermeulen/00-RawData/Good-or-Bad.csv

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

## Raw Data Values ################################

ID FieldA FieldB FieldC FieldD FieldE FieldF FieldG 0 1.0 Good Better Best 1024.0 NaN 10241.0 1

1 2.0 Good NaN Best 512.0 NaN 5121.0 2 ################################

## Data Profile ################################

Rows : 21

Columns : 8 ################################ ################################

## Test Data Values ################################

FieldG

0 1

1 2

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

## Data Profile ################################

Rows : 21

Columns : 1 ################################ ################################ ### Done!! ##################### ################################

>>>

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

##################### Assess-Good-Bad-03.py ################

# -\*- coding: utf-8 -\*- ################################################################

import sys import os

import pandas as pd ################################################################

sInputFileName='Good-or-Bad.csv' sOutputFileName='Good-or-Bad-03.csv' Company='01-Vermeulen' Base='C:/VKHCG'

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

print('Working Base :',Base, ' using Windows ~~~~') print('################################') ################################################################

sFileDir=Base + '/' + Company + '/02-Assess/01-EDS/02-Python' if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

### Import Warehouse ################################################################

sFileName=Base + '/' + Company + '/00-RawData/' + sInputFileName print('Loading :',sFileName) RawData=pd.read\_csv(sFileName,header=0) print('################################')

print('## Raw Data Values') print('################################')

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

print('## Data Profile') print('################################')

print('Rows :',RawData.shape[0]) print('Columns :',RawData.shape[1]) print('################################')

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

sFileName=sFileDir + '/' + sInputFileName RawData.to\_csv(sFileName, index = False)

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

TestData=RawData.dropna(thresh=2) print('################################')

print('## Test Data Values') print('################################')

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

print('## Data Profile') print('################################')

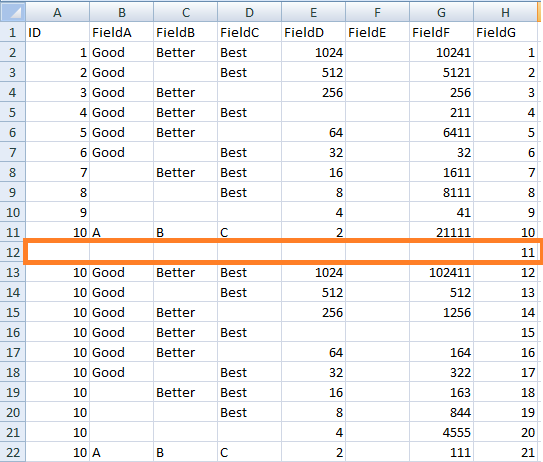
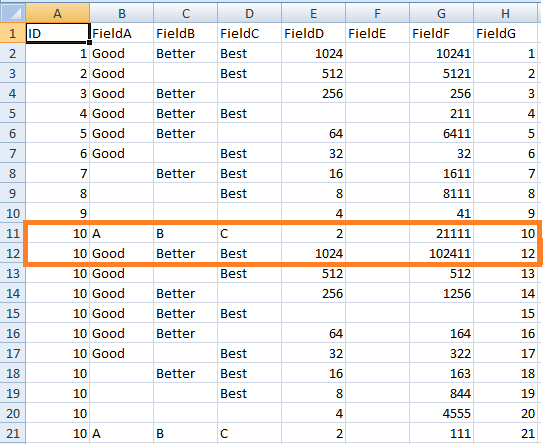
print('Rows :',TestData.shape[0]) print('Columns :',TestData.shape[1])

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

sFileName=sFileDir + '/' + sOutputFileName TestData.to\_csv(sFileName, index = False)

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

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

 **Before After**

Row with more than two missing values got deleted.

The next step along the route is to generate a full network routing solution for the company, to resolve the data issues in the retrieve data.

* + 1. **Write Python / R program to create the network routing diagram from the given data onrouters.**

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

import os

import pandas as pd ################################################################

pd.options.mode.chained\_assignment = None ################################################################ Base='C:/VKHCG' ################################################################ print('################################')

print('Working Base :',Base, ' using Windows') print('################################') ################################################################

sInputFileName1='01-Retrieve/01-EDS/01-R/Retrieve\_Country\_Code.csv' sInputFileName2='01-Retrieve/01-EDS/02-Python/Retrieve\_Router\_Location.csv' sInputFileName3='01-Retrieve/01-EDS/01-R/Retrieve\_IP\_DATA.csv' ################################################################

sOutputFileName='Assess-Network-Routing-Company.csv' Company='01-Vermeulen'

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

### Import Country Data ################################################################

sFileName=Base + '/' + Company + '/' + sInputFileName1

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

print('Loading :',sFileName) print('################################')

CountryData=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1") print('Loaded Country:',CountryData.columns.values) print('################################') ################################################################

## Assess Country Data ################################################################ print('################################')

print('Changed :',CountryData.columns.values) CountryData.rename(columns={'Country': 'Country\_Name'}, inplace=True) CountryData.rename(columns={'ISO-2-CODE': 'Country\_Code'}, inplace=True) CountryData.drop('ISO-M49', axis=1, inplace=True)

CountryData.drop('ISO-3-Code', axis=1, inplace=True) CountryData.drop('RowID', axis=1, inplace=True) print('To :',CountryData.columns.values) print('################################')

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

### Import Company Data ################################################################

sFileName=Base + '/' + Company + '/' + sInputFileName2 print('################################')

print('Loading :',sFileName) print('################################')

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

print('Loaded Company :',CompanyData.columns.values) print('################################') ################################################################

## Assess Company Data ################################################################ print('################################')

print('Changed :',CompanyData.columns.values) CompanyData.rename(columns={'Country': 'Country\_Code'}, inplace=True) print('To :',CompanyData.columns.values) print('################################')

### Import Customer Data ################################################################

sFileName=Base + '/' + Company + '/' + sInputFileName3 print('################################')

print('Loading :',sFileName) print('################################')

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

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

print('Loaded Customer :',CustomerRawData.columns.values) print('################################') ################################################################

CustomerData=CustomerRawData.dropna(axis=0, how='any') print('################################')

print('Remove Blank Country Code')

print('Reduce Rows from', CustomerRawData.shape[0],' to ', CustomerData.shape[0]) print('################################') ################################################################ print('################################')

print('Changed :',CustomerData.columns.values) CustomerData.rename(columns={'Country': 'Country\_Code'}, inplace=True) print('To :',CustomerData.columns.values) print('################################') ################################################################ print('################################')

print('Merge Company and Country Data') print('################################')

CompanyNetworkData=pd.merge( CompanyData,

CountryData, how='inner', on='Country\_Code'

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

print('Change ',CompanyNetworkData.columns.values) for i in CompanyNetworkData.columns.values:

j='Company\_'+i

CompanyNetworkData.rename(columns={i: j}, inplace=True) print('To ', CompanyNetworkData.columns.values) print('################################')

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

sFileDir=Base + '/' + Company + '/02-Assess/01-EDS/02-Python' if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sFileName=sFileDir + '/' + sOutputFileName print('################################')

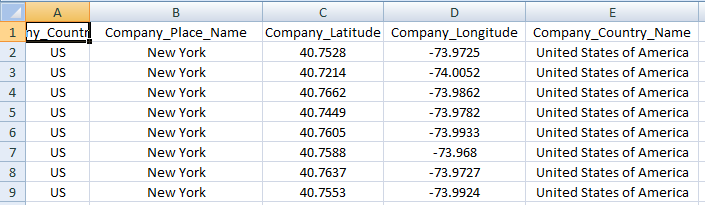
print('Storing :', sFileName) print('################################')

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

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

**Output:**

**Go to C:\VKHCG\01-Vermeulen\02-Assess\01-EDS\02-Python folder and open Assess-Network-Routing-Company.csv**



Next, Access the the customers location using network router location. ####################Assess-Network-Routing-Customer.py ###################### import sys

import os

import pandas as pd ################################################################

pd.options.mode.chained\_assignment = None ################################################################ Base='C:/VKHCG'

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

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

sInputFileName=Base+'/01-Vermeulen/02-Assess/01-EDS/02-Python/Assess-Network- Routing-Customer.csv' ################################################################

sOutputFileName='Assess-Network-Routing-Customer.gml' Company='01-Vermeulen'

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

### Import Country Data ################################################################

sFileName=sInputFileName print('################################')

print('Loading :',sFileName) print('################################')

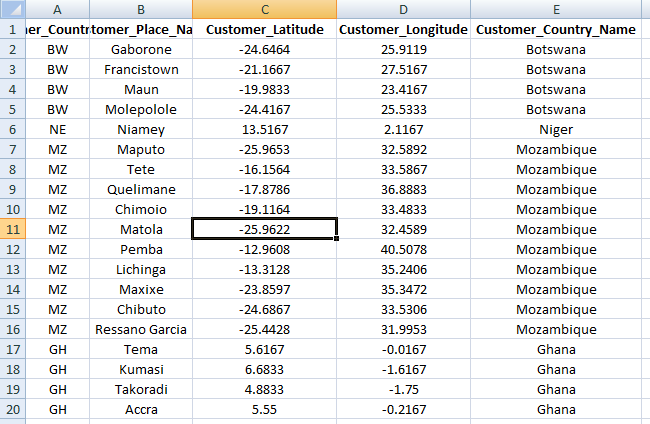
CustomerData=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1") print('Loaded Country:',CustomerData.columns.values) print('################################')

print(CustomerData.head()) print('################################') print('### Done!! #####################') print('################################')

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

**Output**

**Assess-Network-Routing-Customer.csv**



**Assess-Network-Routing-Node.py**

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

import sys import os

import pandas as pd ################################################################

pd.options.mode.chained\_assignment = None ################################################################ Base='C:/VKHCG' ################################################################ print('################################')

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

sInputFileName='01-Retrieve/01-EDS/02-Python/Retrieve\_IP\_DATA.csv' ################################################################

sOutputFileName='Assess-Network-Routing-Node.csv' Company='01-Vermeulen'

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

### Import IP Data ################################################################

sFileName=Base + '/' + Company + '/' + sInputFileName print('################################')

print('Loading :',sFileName) print('################################')

IPData=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1") print('Loaded IP :', IPData.columns.values) print('################################') ################################################################ print('################################')

print('Changed :',IPData.columns.values) IPData.drop('RowID', axis=1, inplace=True) IPData.drop('ID', axis=1, inplace=True)

IPData.rename(columns={'Country': 'Country\_Code'}, inplace=True) IPData.rename(columns={'Place.Name': 'Place\_Name'}, inplace=True) IPData.rename(columns={'Post.Code': 'Post\_Code'}, inplace=True)

IPData.rename(columns={'First.IP.Number': 'First\_IP\_Number'}, inplace=True) IPData.rename(columns={'Last.IP.Number': 'Last\_IP\_Number'}, inplace=True) print('To :',IPData.columns.values) print('################################') ################################################################ print('################################')

print('Change ',IPData.columns.values) for i in IPData.columns.values:

j='Node\_'+i

IPData.rename(columns={i: j}, inplace=True) print('To ', IPData.columns.values) print('################################')

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

sFileDir=Base + '/' + Company + '/02-Assess/01-EDS/02-Python' if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sFileName=sFileDir + '/' + sOutputFileName print('################################')

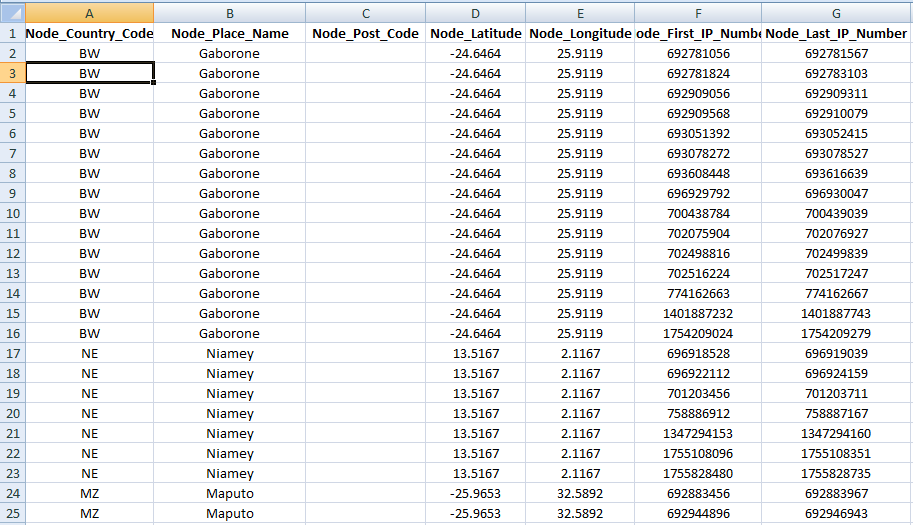
print('Storing :', sFileName) print('################################')

IPData.to\_csv(sFileName, index = False, encoding="latin-1") ################################################################ print('################################')

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

**Output:**

**C:/VKHCG/01-Vermeulen/02-Assess/01-EDS/02-Python/Assess-Network-Routing- Node.csv**



Directed Acyclic Graph (DAG)

A directed acyclic graph is a specific graph that only has one path through the graph.

* + 1. **Write a Python / R program to build directed acyclic graph.**

Open your python editor and create a file named Assess-DAG-Location.py in directory C:\VKHCG\01-Vermeulen\02-Assess

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

import networkx as nx

import matplotlib.pyplot as plt import sys

import os

import pandas as pd ################################################################ Base='C:/VKHCG' ################################################################ print('################################')

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

sInputFileName='01-Retrieve/01-EDS/02-Python/Retrieve\_Router\_Location.csv' sOutputFileName1='Assess-DAG-Company-Country.png' sOutputFileName2='Assess-DAG-Company-Country-Place.png'

Company='01-Vermeulen' ################################################################

### Import Company Data ################################################################

sFileName=Base + '/' + Company + '/' + sInputFileName print('################################')

print('Loading :',sFileName) print('################################')

CompanyData=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1") print('Loaded Company :',CompanyData.columns.values) print('################################') ################################################################

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

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

G1=nx.DiGraph() G2=nx.DiGraph()

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

for i in range(CompanyData.shape[0]): G1.add\_node(CompanyData['Country'][i])

sPlaceName= CompanyData['Place\_Name'][i] + '-' + CompanyData['Country'][i] G2.add\_node(sPlaceName)

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

for n1 in G1.nodes():

for n2 in G1.nodes(): if n1 != n2:

print('Link :',n1,' to ', n2) G1.add\_edge(n1,n2)

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

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

print("Nodes of graph: ") print(G1.nodes()) print("Edges of graph: ") print(G1.edges())

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

sFileDir=Base + '/' + Company + '/02-Assess/01-EDS/02-Python' if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sFileName=sFileDir + '/' + sOutputFileName1 print('################################')

print('Storing :', sFileName) print('################################')

nx.draw(G1,pos=nx.spectral\_layout(G1), nodecolor='r',edge\_color='g', with\_labels=True,node\_size=8000, font\_size=12)

plt.savefig(sFileName) # save as png plt.show() # display

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

for n1 in G2.nodes(): for n2 in G2.nodes():

if n1 != n2:

print('Link :',n1,' to ', n2) G2.add\_edge(n1,n2)

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

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

print("Nodes of graph: ") print(G2.nodes()) print("Edges of graph: ") print(G2.edges())

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

sFileDir=Base + '/' + Company + '/02-Assess/01-EDS/02-Python' if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sFileName=sFileDir + '/' + sOutputFileName2 print('################################')

print('Storing :', sFileName)

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

nx.draw(G2,pos=nx.spectral\_layout(G2), nodecolor='r',edge\_color='b', with\_labels=True,node\_size=8000, font\_size=12)

plt.savefig(sFileName) # save as png plt.show() # display

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

**Output: ################################**

**Rows : 150 ################################ ################################**

**Link : US to DE Link : US to GB Link : DE to US Link : DE to GB Link : GB to US Link : GB to DE**

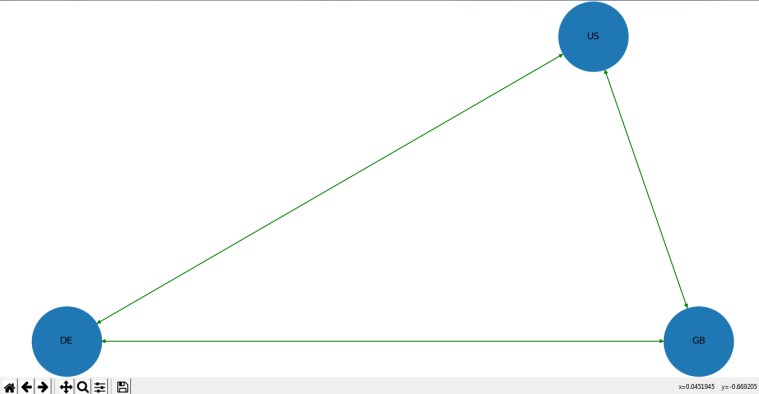
**################################ ################################**

**Nodes of graph:**

**['US', 'DE', 'GB']**

**Edges of graph:**

**[('US', 'DE'), ('US', 'GB'), ('DE', 'US'), ('DE', 'GB'), ('GB', 'US'), ('GB', 'DE')] ################################**



**Customer Location DAG**

################### **Assess-DAG-Location.py**###################

import networkx as nx

import matplotlib.pyplot as plt import sys

import os

import pandas as pd ################################################################ Base='C:/VKHCG' ################################################################ print('################################')

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

sInputFileName='01-Retrieve/01-EDS/02-Python/Retrieve\_Router\_Location.csv' sOutputFileName1='Assess-DAG-Company-Country.png' sOutputFileName2='Assess-DAG-Company-Country-Place.png'

Company='01-Vermeulen' ################################################################

### Import Company Data ################################################################

sFileName=Base + '/' + Company + '/' + sInputFileName print('################################')

print('Loading :',sFileName) print('################################')

CompanyData=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1") print('Loaded Company :',CompanyData.columns.values) print('################################') ################################################################

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

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

G1=nx.DiGraph() G2=nx.DiGraph()

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

for i in range(CompanyData.shape[0]): G1.add\_node(CompanyData['Country'][i])

sPlaceName= CompanyData['Place\_Name'][i] + '-' + CompanyData['Country'][i] G2.add\_node(sPlaceName)

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

for n1 in G1.nodes(): for n2 in G1.nodes():

if n1 != n2:

print('Link :',n1,' to ', n2) G1.add\_edge(n1,n2)

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

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

print("Nodes of graph: ") print(G1.nodes()) print("Edges of graph: ") print(G1.edges())

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

sFileDir=Base + '/' + Company + '/02-Assess/01-EDS/02-Python' if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sFileName=sFileDir + '/' + sOutputFileName1 print('################################')

print('Storing :', sFileName) print('################################')

nx.draw(G1,pos=nx.spectral\_layout(G1), nodecolor='r',edge\_color='g', with\_labels=True,node\_size=8000, font\_size=12)

plt.savefig(sFileName) # save as png plt.show() # display

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

for n1 in G2.nodes(): for n2 in G2.nodes():

if n1 != n2:

print('Link :',n1,' to ', n2) G2.add\_edge(n1,n2)

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

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

print("Nodes of graph: ") print(G2.nodes()) print("Edges of graph: ") print(G2.edges())

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

sFileDir=Base + '/' + Company + '/02-Assess/01-EDS/02-Python' if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sFileName=sFileDir + '/' + sOutputFileName2 print('################################')

print('Storing :', sFileName) print('################################')

nx.draw(G2,pos=nx.spectral\_layout(G2), nodecolor='r',edge\_color='b', with\_labels=True,node\_size=8000, font\_size=12)

plt.savefig(sFileName) # save as png

plt.show() # display ################################################################

**Output:**

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

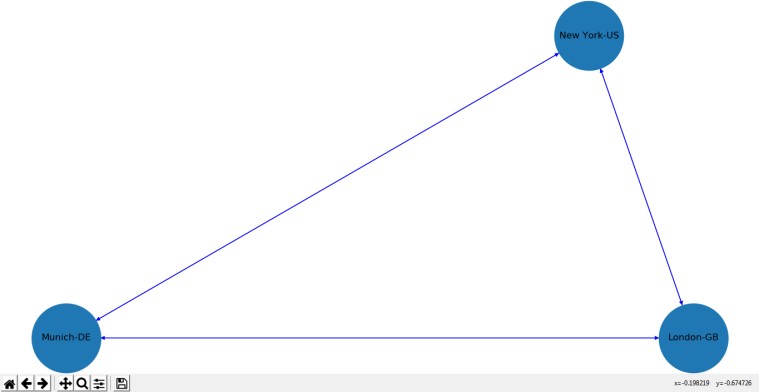
Link : New York-US to Munich-DE Link : New York-US to London-GB Link : Munich-DE to New York-US Link : Munich-DE to London-GB Link : London-GB to New York-US Link : London-GB to Munich-DE ################################ ################################

Nodes of graph:

['New York-US', 'Munich-DE', 'London-GB'] Edges of graph:

[('New York-US', 'Munich-DE'), ('New York-US', 'London-GB'), ('Munich-DE', 'New York-

US'), ('Munich-DE', 'London-GB'), ('London-GB', 'New York-US'), ('London-GB', 'Munich- DE')]



Open your Python editor and create a file named Assess-DAG-GPS.py in directory C:\VKHCG\01-Vermeulen\02-Assess.

import networkx as nx

import matplotlib.pyplot as plt import sys

import os

import pandas as pd Base='C:/VKHCG'

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

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

sInputFileName='01-Retrieve/01-EDS/02-Python/Retrieve\_Router\_Location.csv' sOutputFileName='Assess-DAG-Company-GPS.png'

Company='01-Vermeulen' ### Import Company Data

sFileName=Base + '/' + Company + '/' + sInputFileName print('################################')

print('Loading :',sFileName) print('################################')

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

print('Loaded Company :',CompanyData.columns.values) print('################################')

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

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

G=nx.Graph()

for i in range(CompanyData.shape[0]): nLatitude=round(CompanyData['Latitude'][i],2) nLongitude=round(CompanyData['Longitude'][i],2)

if nLatitude < 0:

sLatitude = str(nLatitude\*-1) + ' S' else:

sLatitude = str(nLatitude) + ' N'

if nLongitude < 0:

sLongitude = str(nLongitude\*-1) + ' W' else:

sLongitude = str(nLongitude) + ' E'

sGPS= sLatitude + '-' + sLongitude G.add\_node(sGPS)

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

for n1 in G.nodes(): for n2 in G.nodes():

if n1 != n2:

print('Link :',n1,' to ', n2) G.add\_edge(n1,n2)

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

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

print("Nodes of graph: ") print(G.number\_of\_nodes()) print("Edges of graph: ") print(G.number\_of\_edges())

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

**Output:**

=== RESTART: C:\VKHCG\01-Vermeulen\02-Assess\Assess-DAG-GPS-unsmoothed.py

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

Working Base : C:/VKHCG using win32 ################################

Loading : C:/VKHCG/01-Vermeulen/01-Retrieve/01-EDS/02- Python/Retrieve\_Router\_Location.csv ################################

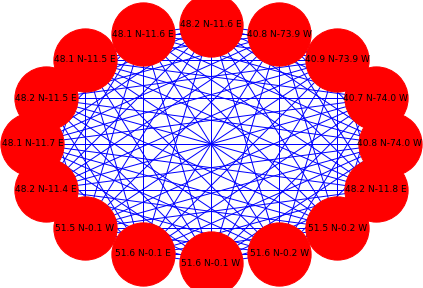
Loaded Company : ['Country' 'Place\_Name' 'Latitude' 'Longitude'] ################################

Country Place\_Name Latitude Longitude 0 US New York 40.7528 -73.9725

1 US New York 40.7214 -74.0052

-

-



* + 1. **Write a Python / R program to pick the content for Bill Boards from the given data.**

**Picking Content for Billboards**

The basic process required is to combine two sets of data and then calculate the number of visitors per day from the range of IP addresses that access the billboards in Germany.

**Bill Board Location: Rows - 8873 Access Visitors: Rows - 75999**

**Access Location Record: Rows – 1,81,235**

Open Python editor and create a file named **Assess-DE-Billboard.py** in directory C:\VKHCG\02-Krennwallner\02-Assess

################# **Assess-DE-Billboard.py**######################

import sys import os

import sqlite3 as sq import pandas as pd

################################################################ Base='C:/VKHCG'

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

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

sInputFileName1='01-Retrieve/01-EDS/02-Python/Retrieve\_DE\_Billboard\_Locations.csv' sInputFileName2='01-Retrieve/01-EDS/02-Python/Retrieve\_Online\_Visitor.csv' sOutputFileName='Assess-DE-Billboard-Visitor.csv'

Company='02-Krennwallner' ################################################################

sDataBaseDir=Base + '/' + Company + '/02-Assess/SQLite' if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir)

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

sDatabaseName=sDataBaseDir + '/krennwallner.db' conn = sq.connect(sDatabaseName)

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

### Import Billboard Data ################################################################

sFileName=Base + '/' + Company + '/' + sInputFileName1 print('################################')

print('Loading :',sFileName) print('################################')

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

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

BillboardData=BillboardRawData

print('Loaded Company :',BillboardData.columns.values) print('################################') ################################################################ print('################')

sTable='Assess\_BillboardData'

print('Storing :',sDatabaseName,' Table:',sTable) BillboardData.to\_sql(sTable, conn, if\_exists="replace") print('################')

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

print(BillboardData.head()) print('################################')

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

### Import Billboard Data ################################################################

sFileName=Base + '/' + Company + '/' + sInputFileName2 print('################################')

print('Loading :',sFileName) print('################################')

VisitorRawData=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1") VisitorRawData.drop\_duplicates(subset=None, keep='first', inplace=True) VisitorData=VisitorRawData[VisitorRawData.Country=='DE']

print('Loaded Company :',VisitorData.columns.values) print('################################') ################################################################ print('################')

sTable='Assess\_VisitorData'

print('Storing :',sDatabaseName,' Table:',sTable) VisitorData.to\_sql(sTable, conn, if\_exists="replace") print('################')

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

print(VisitorData.head()) print('################################')

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

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

sTable='Assess\_BillboardVisitorData' print('Loading :',sDatabaseName,' Table:',sTable) sSQL="select distinct"

sSQL=sSQL+ " A.Country AS BillboardCountry," sSQL=sSQL+ " A.Place\_Name AS BillboardPlaceName," sSQL=sSQL+ " A.Latitude AS BillboardLatitude, " sSQL=sSQL+ " A.Longitude AS BillboardLongitude," sSQL=sSQL+ " B.Country AS VisitorCountry," sSQL=sSQL+ " B.Place\_Name AS VisitorPlaceName," sSQL=sSQL+ " B.Latitude AS VisitorLatitude, " sSQL=sSQL+ " B.Longitude AS VisitorLongitude,"

sSQL=sSQL+ " (B.Last\_IP\_Number - B.First\_IP\_Number) \* 365.25 \* 24 \* 12 AS VisitorYearRate"

sSQL=sSQL+ " from"

sSQL=sSQL+ " Assess\_BillboardData as A" sSQL=sSQL+ " JOIN "

sSQL=sSQL+ " Assess\_VisitorData as B" sSQL=sSQL+ " ON "

sSQL=sSQL+ " A.Country = B.Country" sSQL=sSQL+ " AND "

sSQL=sSQL+ " A.Place\_Name = B.Place\_Name;" BillboardVistorsData=pd.read\_sql\_query(sSQL, conn) print('################') ################################################################ print('################')

sTable='Assess\_BillboardVistorsData' print('Storing :',sDatabaseName,' Table:',sTable)

BillboardVistorsData.to\_sql(sTable, conn, if\_exists="replace") print('################') ################################################################

print(BillboardVistorsData.head()) print('################################')

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

sFileDir=Base + '/' + Company + '/02-Assess/01-EDS/02-Python' if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################ print('################################')

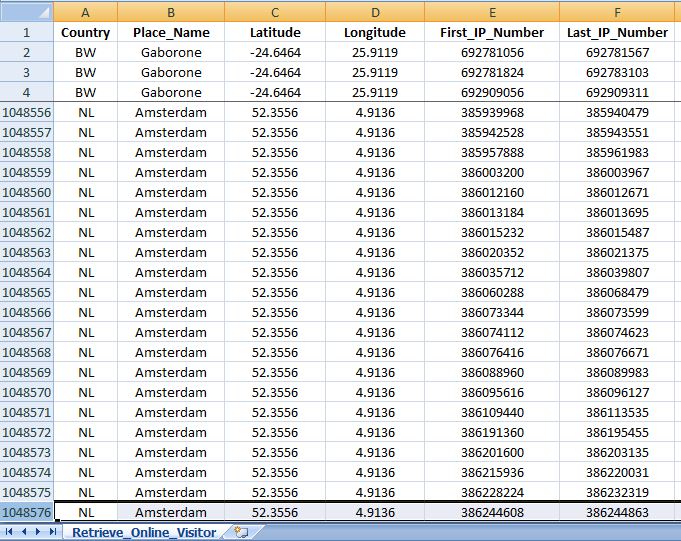
print('Storing :', sFileName) print('################################')

sFileName=sFileDir + '/' + sOutputFileName BillboardVistorsData.to\_csv(sFileName, index = False) print('################################') ################################################################ print('### Done!! ############################################') ################################################################

**Output:**

C:\VKHCG\02-Krennwallner\01-Retrieve\01-EDS\02-Python\Retrieve\_Online\_Visitor.csv containing, 10,48,576**(Ten lack Forty Eight Thousand Five Hundred and Seventy Six**

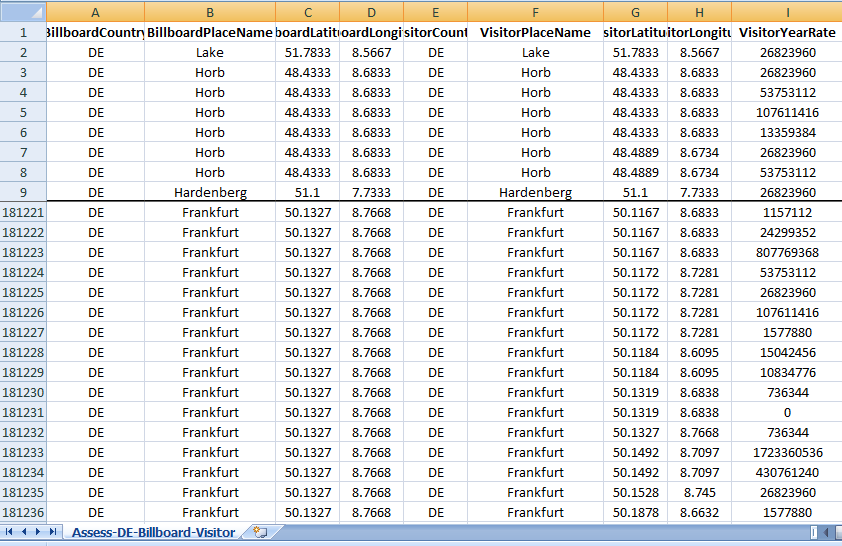
**)**rows.



**SQLite Visitor’s Database**

C:/VKHCG/02-Krennwallner/02-Assess/SQLite/krennwallner.db Table: BillboardCountry BillboardPlaceName ... VisitorLongitude VisitorYearRate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | DE | Lake | ... | 8.5667 | 26823960.0 |
| 1 | DE | Horb | ... | 8.6833 | 26823960.0 |
| 2 | DE | Horb | ... | 8.6833 | 53753112.0 |
| 3 | DE | Horb | ... | 8.6833 | 107611416.0 |
| 4 | DE | Horb | ... | 8.6833 | 13359384.0 |



* + 1. **Write a Python / R program to generate GML file from the given csv file. Understanding Your Online Visitor Data**

Online visitors have to be mapped to their closest billboard, to ensure we understand where and what they can access.

Open your Python editor and create a file called Assess-Billboard\_2\_Visitor.py in directory

C:\VKHCG\ 02-Krennwallner\02-Assess. import networkx as nx

import sys import os

import sqlite3 as sq import pandas as pd

from geopy.distance import vincenty ################################################################ Base='C:/VKHCG'

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

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

Company='02-Krennwallner' sTable='Assess\_BillboardVisitorData' sOutputFileName='Assess-DE-Billboard-Visitor.gml'

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

sDataBaseDir=Base + '/' + Company + '/02-Assess/SQLite' if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir) ################################################################

sDatabaseName=sDataBaseDir + '/krennwallner.db' conn = sq.connect(sDatabaseName)

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

print('Loading :',sDatabaseName,' Table:',sTable) sSQL="select "

sSQL=sSQL+ " A.BillboardCountry," sSQL=sSQL+ " A.BillboardPlaceName,"

sSQL=sSQL+ " ROUND(A.BillboardLatitude,3) AS BillboardLatitude, " sSQL=sSQL+ " ROUND(A.BillboardLongitude,3) AS BillboardLongitude," sSQL=sSQL+ " (CASE WHEN A.BillboardLatitude < 0 THEN " sSQL=sSQL+ " 'S' || ROUND(ABS(A.BillboardLatitude),3)"

sSQL=sSQL+ " ELSE "

sSQL=sSQL+ " 'N' || ROUND(ABS(A.BillboardLatitude),3)" sSQL=sSQL+ " END ) AS sBillboardLatitude,"

sSQL=sSQL+ " (CASE WHEN A.BillboardLongitude < 0 THEN " sSQL=sSQL+ " 'W' || ROUND(ABS(A.BillboardLongitude),3)" sSQL=sSQL+ " ELSE "

sSQL=sSQL+ " 'E' || ROUND(ABS(A.BillboardLongitude),3)" sSQL=sSQL+ " END ) AS sBillboardLongitude," sSQL=sSQL+ " A.VisitorCountry,"

sSQL=sSQL+ " A.VisitorPlaceName,"

sSQL=sSQL+ " ROUND(A.VisitorLatitude,3) AS VisitorLatitude, " sSQL=sSQL+ " ROUND(A.VisitorLongitude,3) AS VisitorLongitude," sSQL=sSQL+ " (CASE WHEN A.VisitorLatitude < 0 THEN " sSQL=sSQL+ " 'S' || ROUND(ABS(A.VisitorLatitude),3)" sSQL=sSQL+ " ELSE "

sSQL=sSQL+ " 'N' ||ROUND(ABS(A.VisitorLatitude),3)" sSQL=sSQL+ " END ) AS sVisitorLatitude,"

sSQL=sSQL+ " (CASE WHEN A.VisitorLongitude < 0 THEN " sSQL=sSQL+ " 'W' || ROUND(ABS(A.VisitorLongitude),3)" sSQL=sSQL+ " ELSE "

sSQL=sSQL+ " 'E' || ROUND(ABS(A.VisitorLongitude),3)" sSQL=sSQL+ " END ) AS sVisitorLongitude," sSQL=sSQL+ " A.VisitorYearRate"

sSQL=sSQL+ " from"

sSQL=sSQL+ " Assess\_BillboardVistorsData AS A;" BillboardVistorsData=pd.read\_sql\_query(sSQL, conn) print('################') ################################################################

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

vincenty((row['BillboardLatitude'],row['BillboardLongitude']), (row['VisitorLatitude'],row['VisitorLongitude'])).miles

,4)

,axis=1) ################################################################

G=nx.Graph() ################################################################

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

sNode0='MediaHub-' + BillboardVistorsData['BillboardCountry'][i] sNode1='B-'+ BillboardVistorsData['sBillboardLatitude'][i] + '-' sNode1=sNode1 + BillboardVistorsData['sBillboardLongitude'][i] G.add\_node(sNode1,

Nodetype='Billboard', Country=BillboardVistorsData['BillboardCountry'][i], PlaceName=BillboardVistorsData['BillboardPlaceName'][i], Latitude=round(BillboardVistorsData['BillboardLatitude'][i],3), Longitude=round(BillboardVistorsData['BillboardLongitude'][i],3))

sNode2='M-'+ BillboardVistorsData['sVisitorLatitude'][i] + '-' sNode2=sNode2 + BillboardVistorsData['sVisitorLongitude'][i] G.add\_node(sNode2,

Nodetype='Mobile', Country=BillboardVistorsData['VisitorCountry'][i], PlaceName=BillboardVistorsData['VisitorPlaceName'][i], Latitude=round(BillboardVistorsData['VisitorLatitude'][i],3), Longitude=round(BillboardVistorsData['VisitorLongitude'][i],3))

print('Link Media Hub :',sNode0,' to Billboard : ', sNode1) G.add\_edge(sNode0,sNode1)

print('Link Post Code :',sNode1,' to GPS : ', sNode2)

G.add\_edge(sNode1,sNode2,distance=round(BillboardVistorsData['Distance'][i])) ################################################################ print('################################')

print("Nodes of graph: ",nx.number\_of\_nodes(G)) print("Edges of graph: ",nx.number\_of\_edges(G)) print('################################')

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

sFileDir=Base + '/' + Company + '/02-Assess/01-EDS/02-Python' if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sFileName=sFileDir + '/' + sOutputFileName print('################################')

print('Storing :', sFileName) print('################################')

nx.write\_gml(G,sFileName) sFileName=sFileName +'.gz' nx.write\_gml(G,sFileName)

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

**Output:**

**This will produce a set of demonstrated values onscreen, plus a graph data file named Assess-DE-Billboard-Visitor.gml.**

(It takes a long time to complete the process, after completion the gml file can be viewed in text editor)

Hence, we have applied formulae to extract features, such as the distance between the billboard and the visitor.

Planning an Event for Top-Ten Customers

Open Python editor and create a file named Assess-Visitors.py in directory C:\VKHCG\02-Krennwallner\02-Assess ################################################################

import sys import os

import sqlite3 as sq import pandas as pd

from pandas.io import sql ################################################################ Base='C:/VKHCG'

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

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

Company='02-Krennwallner'

sInputFileName='01-Retrieve/01-EDS/02-Python/Retrieve\_Online\_Visitor.csv' ################################################################

sDataBaseDir=Base + '/' + Company + '/02-Assess/SQLite'

if not os.path.exists(sDataBaseDir): os.makedirs(sDataBaseDir)

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

sDatabaseName=sDataBaseDir + '/krennwallner.db' conn = sq.connect(sDatabaseName)

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

### Import Country Data ################################################################

sFileName=Base + '/' + Company + '/' + sInputFileName print('################################')

print('Loading :',sFileName) print('################################')

VisitorRawData=pd.read\_csv(sFileName, header=0, low\_memory=False, encoding="latin-1", skip\_blank\_lines=True)

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

VisitorData=VisitorRawData

print('Loaded Company :',VisitorData.columns.values) print('################################') ################################################################ print('################')

sTable='Assess\_Visitor'

print('Storing :',sDatabaseName,' Table:',sTable) VisitorData.to\_sql(sTable, conn, if\_exists="replace") print('################')

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

print(VisitorData.head()) print('################################')

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

sView='Assess\_Visitor\_UseIt'

print('Creating :',sDatabaseName,' View:',sView) sSQL="DROP VIEW IF EXISTS " + sView + ";"

sql.execute(sSQL,conn)

sSQL="CREATE VIEW " + sView + " AS" sSQL=sSQL+ " SELECT"

sSQL=sSQL+ " A.Country," sSQL=sSQL+ " A.Place\_Name," sSQL=sSQL+ " A.Latitude," sSQL=sSQL+ " A.Longitude,"

sSQL=sSQL+ " (A.Last\_IP\_Number - A.First\_IP\_Number) AS UsesIt" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " Assess\_Visitor as A" sSQL=sSQL+ " WHERE"

sSQL=sSQL+ " Country is not null" sSQL=sSQL+ " AND"

sSQL=sSQL+ " Place\_Name is not null;" sql.execute(sSQL,conn)

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

sView='Assess\_Total\_Visitors\_Location' print('Creating :',sDatabaseName,' View:',sView) sSQL="DROP VIEW IF EXISTS " + sView + ";"

sql.execute(sSQL,conn)

sSQL="CREATE VIEW " + sView + " AS" sSQL=sSQL+ " SELECT"

sSQL=sSQL+ " Country," sSQL=sSQL+ " Place\_Name,"

sSQL=sSQL+ " SUM(UsesIt) AS TotalUsesIt" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " Assess\_Visitor\_UseIt" sSQL=sSQL+ " GROUP BY"

sSQL=sSQL+ " Country," sSQL=sSQL+ " Place\_Name" sSQL=sSQL+ " ORDER BY"

sSQL=sSQL+ " TotalUsesIt DESC" sSQL=sSQL+ " LIMIT 10;"

sql.execute(sSQL,conn) ################################################################# print('################')

sView='Assess\_Total\_Visitors\_GPS' print('Creating :',sDatabaseName,' View:',sView) sSQL="DROP VIEW IF EXISTS " + sView + ";"

sql.execute(sSQL,conn)

sSQL="CREATE VIEW " + sView + " AS" sSQL=sSQL+ " SELECT"

sSQL=sSQL+ " Latitude," sSQL=sSQL+ " Longitude,"

sSQL=sSQL+ " SUM(UsesIt) AS TotalUsesIt" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " Assess\_Visitor\_UseIt" sSQL=sSQL+ " GROUP BY"

sSQL=sSQL+ " Latitude," sSQL=sSQL+ " Longitude" sSQL=sSQL+ " ORDER BY"

sSQL=sSQL+ " TotalUsesIt DESC" sSQL=sSQL+ " LIMIT 10;"

sql.execute(sSQL,conn) #################################################################

sTables=['Assess\_Total\_Visitors\_Location', 'Assess\_Total\_Visitors\_GPS'] for sTable in sTables:

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

print('Loading :',sDatabaseName,' Table:',sTable) sSQL=" SELECT "

sSQL=sSQL+ " \*" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " " + sTable + ";" TopData=pd.read\_sql\_query(sSQL, conn) print('################')

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

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

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

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

Output:



* + 1. **Write a Python / R program to plan the locations of the warehouses from the given data.**

Planning the Locations of the Warehouses

Planning the location of the warehouses requires the assessment of the GPS locations of these warehouses against the requirements for Hillman’s logistics needs.

Open your editor and create a file named Assess-Warehouse-Address.py in directory C:\VKHCG\03-Hillman\02-Assess.

################## Assess-Warehouse-Address.py ###################

import os

import pandas as pd

from geopy.geocoders import Nominatim geolocator = Nominatim()

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

InputDir='01-Retrieve/01-EDS/01-R' InputFileName='Retrieve\_GB\_Postcode\_Warehouse.csv' EDSDir='02-Assess/01-EDS'

OutputDir=EDSDir + '/02-Python' OutputFileName='Assess\_GB\_Warehouse\_Address.csv' Company='03-Hillman'

################################################################ Base='C:/VKHCG'

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

print('Working Base :',Base, ' using Windows') print('################################') ################################################################

sFileDir=Base + '/' + Company + '/' + EDSDir if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sFileDir=Base + '/' + Company + '/' + OutputDir if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sFileName=Base + '/' + Company + '/' + InputDir + '/' + InputFileName print('###########')

print('Loading :',sFileName) Warehouse=pd.read\_csv(sFileName,header=0,low\_memory=False) Warehouse.sort\_values(by='postcode', ascending=1) ################################################################

## Limited to 10 due to service limit on Address Service. ################################################################

WarehouseGoodHead=Warehouse[Warehouse.latitude != 0].head(5) WarehouseGoodTail=Warehouse[Warehouse.latitude != 0].tail(5) ################################################################

WarehouseGoodHead['Warehouse\_Point']=WarehouseGoodHead.apply(lambda row: (str(row['latitude'])+','+str(row['longitude']))

,axis=1) WarehouseGoodHead['Warehouse\_Address']=WarehouseGoodHead.apply(lambda row:

geolocator.reverse(row['Warehouse\_Point']).address

,axis=1)

WarehouseGoodHead.drop('Warehouse\_Point', axis=1, inplace=True) WarehouseGoodHead.drop('id', axis=1, inplace=True) WarehouseGoodHead.drop('postcode', axis=1, inplace=True) ################################################################

WarehouseGoodTail['Warehouse\_Point']=WarehouseGoodTail.apply(lambda row: (str(row['latitude'])+','+str(row['longitude']))

,axis=1) WarehouseGoodTail['Warehouse\_Address']=WarehouseGoodTail.apply(lambda row:

geolocator.reverse(row['Warehouse\_Point']).address

,axis=1)

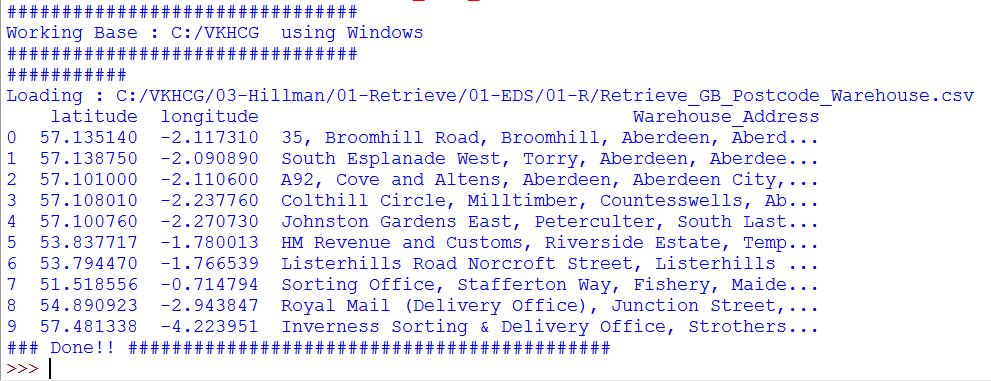
WarehouseGoodTail.drop('Warehouse\_Point', axis=1, inplace=True) WarehouseGoodTail.drop('id', axis=1, inplace=True) WarehouseGoodTail.drop('postcode', axis=1, inplace=True) ################################################################

WarehouseGood=WarehouseGoodHead.append(WarehouseGoodTail, ignore\_index=True) print(WarehouseGood) ################################################################

sFileName=sFileDir + '/' + OutputFileName WarehouseGood.to\_csv(sFileName, index = False)

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

## Output:



* + 1. **Write a Python / R program using data science via clustering to determine new warehouses using the given data.**

**Global New Warehouse:**Hillman wants to add extra global warehouses, and you are required to assess wherethey should be located. We only have to collect the possible locations for warehouses.

The following example will show you how to modify the data columns you read in that are totally ambiguous. Open Python editor and create a file named Assess-Warehouse-Global.py in directory

C:\VKHCG\03-Hillman\02-Assess

################# Assess-Warehouse-Global.py############## import sys

import os

import pandas as pd ################################################################ Base='C:/VKHCG' ################################################################ print('################################')

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

Company='03-Hillman'

InputDir='01-Retrieve/01-EDS/01-R' InputFileName='Retrieve\_All\_Countries.csv' EDSDir='02-Assess/01-EDS' OutputDir=EDSDir + '/02-Python' OutputFileName='Assess\_All\_Warehouse.csv'

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

sFileDir=Base + '/' + Company + '/' + EDSDir if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sFileDir=Base + '/' + Company + '/' + OutputDir

if not os.path.exists(sFileDir): os.makedirs(sFileDir)

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

sFileName=Base + '/' + Company + '/' + InputDir + '/' + InputFileName print('###########')

print('Loading :',sFileName) Warehouse=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1") ################################################################

sColumns={'X1' : 'Country', 'X2' : 'PostCode',

'X3' : 'PlaceName',

'X4' : 'AreaName',

'X5' : 'AreaCode',

'X10' : 'Latitude',

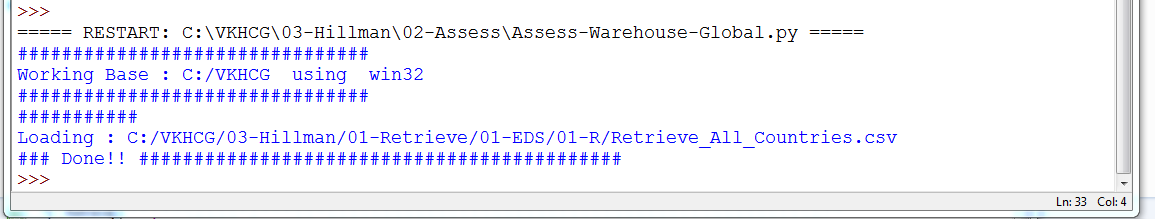
'X11' : 'Longitude'} Warehouse.rename(columns=sColumns,inplace=True) WarehouseGood=Warehouse ################################################################

sFileName=sFileDir + '/' + OutputFileName WarehouseGood.to\_csv(sFileName, index = False)

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

This will produce a set of demonstrated values onscreen, plus a graph data file named Assess\_All\_Warehouse.csv.

**Output:**



Open Assess0\_All\_Warehose.csv from C:\VKHCG\03-Hillman\02-Assess\01-EDS\02- Python



* + 1. **Using the given data, write a Python / R program to plan the shipping routes for best-fit international logistics.**

Hillman requires an international logistics solution to support all the required shippingroutes.

Open Python editor and create a file named Assess-Best-Fit-Logistics.py in directory C:\VKHCG\03-Hillman\02-Assess

import sys import os

import pandas as pd import networkx as nx

from geopy.distance import vincenty import sqlite3 as sq

from pandas.io import sql ################################################################

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='01-Retrieve/01-EDS/01-R' InputFileName='Retrieve\_All\_Countries.csv' EDSDir='02-Assess/01-EDS' OutputDir=EDSDir + '/02-Python' OutputFileName='Assess\_Best\_Logistics.gml'

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

sFileDir=Base + '/' + Company + '/' + EDSDir if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

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

sFileDir=Base + '/' + Company + '/' + OutputDir if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sDataBaseDir=Base + '/' + Company + '/02-Assess/SQLite' if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir) ################################################################

sDatabaseName=sDataBaseDir + '/Hillman.db' conn = sq.connect(sDatabaseName)

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

sFileName=Base + '/' + Company + '/' + InputDir + '/' + InputFileName print('###########')

print('Loading :',sFileName) Warehouse=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1") ################################################################

sColumns={'X1' : 'Country', 'X2' : 'PostCode',

'X3' : 'PlaceName',

'X4' : 'AreaName',

'X5' : 'AreaCode',

'X10' : 'Latitude',

'X11' : 'Longitude'} Warehouse.rename(columns=sColumns,inplace=True) WarehouseGood=Warehouse #print(WarehouseGood.head())

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

RoutePointsCountry=pd.DataFrame(WarehouseGood.groupby(['Country'])[['Latitude','Longitude']].mean()) #print(RoutePointsCountry.head())

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

sTable='Assess\_RoutePointsCountry' print('Storing :',sDatabaseName,' Table:',sTable)

RoutePointsCountry.to\_sql(sTable, conn, if\_exists="replace") print('################') ################################################################

RoutePointsPostCode=pd.DataFrame(WarehouseGood.groupby(['Country', 'PostCode'])[['Latitude','Longitude']].mean()) #print(RoutePointsPostCode.head())

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

sTable='Assess\_RoutePointsPostCode' print('Storing :',sDatabaseName,' Table:',sTable)

RoutePointsPostCode.to\_sql(sTable, conn, if\_exists="replace") print('################') ################################################################

RoutePointsPlaceName=pd.DataFrame(WarehouseGood.groupby(['Country', 'PostCode','PlaceName'])[['Latitude','Longitude']].mean()) #print(RoutePointsPlaceName.head())

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

sTable='Assess\_RoutePointsPlaceName' print('Storing :',sDatabaseName,' Table:',sTable)

RoutePointsPlaceName.to\_sql(sTable, conn, if\_exists="replace") print('################') ################################################################

### Fit Country to Country ################################################################ print('################')

sView='Assess\_RouteCountries'

print('Creating :',sDatabaseName,' View:',sView) sSQL="DROP VIEW IF EXISTS " + sView + ";"

sql.execute(sSQL,conn)

sSQL="CREATE VIEW " + sView + " AS" sSQL=sSQL+ " SELECT DISTINCT"

sSQL=sSQL+ " S.Country AS SourceCountry," sSQL=sSQL+ " S.Latitude AS SourceLatitude," sSQL=sSQL+ " S.Longitude AS SourceLongitude," sSQL=sSQL+ " T.Country AS TargetCountry," sSQL=sSQL+ " T.Latitude AS TargetLatitude," sSQL=sSQL+ " T.Longitude AS TargetLongitude" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " Assess\_RoutePointsCountry AS S" sSQL=sSQL+ " ,"

sSQL=sSQL+ " Assess\_RoutePointsCountry AS T" sSQL=sSQL+ " WHERE S.Country <> T.Country" sSQL=sSQL+ " AND"

sSQL=sSQL+ " S.Country in ('GB','DE','BE','AU','US','IN')" sSQL=sSQL+ " AND"

sSQL=sSQL+ " T.Country in ('GB','DE','BE','AU','US','IN');"

sql.execute(sSQL,conn)

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

print('Loading :',sDatabaseName,' Table:',sView) sSQL=" SELECT "

sSQL=sSQL+ " \*" sSQL=sSQL+ " FROM" sSQL=sSQL+ " " + sView + ";"

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

RouteCountries['Distance']=RouteCountries.apply(lambda row: round( vincenty((row['SourceLatitude'],row['SourceLongitude']),

(row['TargetLatitude'],row['TargetLongitude'])).miles,4),axis=1)

print(RouteCountries.head(5)) ################################################################

### Fit Country to Post Code ################################################################ print('################')

sView='Assess\_RoutePostCode'

print('Creating :',sDatabaseName,' View:',sView) sSQL="DROP VIEW IF EXISTS " + sView + ";"

sql.execute(sSQL,conn)

sSQL="CREATE VIEW " + sView + " AS" sSQL=sSQL+ " SELECT DISTINCT"

sSQL=sSQL+ " S.Country AS SourceCountry," sSQL=sSQL+ " S.Latitude AS SourceLatitude," sSQL=sSQL+ " S.Longitude AS SourceLongitude," sSQL=sSQL+ " T.Country AS TargetCountry," sSQL=sSQL+ " T.PostCode AS TargetPostCode," sSQL=sSQL+ " T.Latitude AS TargetLatitude," sSQL=sSQL+ " T.Longitude AS TargetLongitude" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " Assess\_RoutePointsCountry AS S" sSQL=sSQL+ " ,"

sSQL=sSQL+ " Assess\_RoutePointsPostCode AS T" sSQL=sSQL+ " WHERE S.Country = T.Country" sSQL=sSQL+ " AND"

sSQL=sSQL+ " S.Country in ('GB','DE','BE','AU','US','IN')"

sSQL=sSQL+ " AND"

sSQL=sSQL+ " T.Country in ('GB','DE','BE','AU','US','IN');"

sql.execute(sSQL,conn)

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

print('Loading :',sDatabaseName,' Table:',sView) sSQL=" SELECT "

sSQL=sSQL+ " \*" sSQL=sSQL+ " FROM" sSQL=sSQL+ " " + sView + ";"

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

RoutePostCode['Distance']=RoutePostCode.apply(lambda row: round( vincenty((row['SourceLatitude'],row['SourceLongitude']),

(row['TargetLatitude'],row['TargetLongitude'])).miles

,4)

,axis=1)

print(RoutePostCode.head(5)) ################################################################

### Fit Post Code to Place Name ################################################################ print('################')

sView='Assess\_RoutePlaceName' print('Creating :',sDatabaseName,' View:',sView)

sSQL="DROP VIEW IF EXISTS " + sView + ";"

sql.execute(sSQL,conn)

sSQL="CREATE VIEW " + sView + " AS" sSQL=sSQL+ " SELECT DISTINCT"

sSQL=sSQL+ " S.Country AS SourceCountry," sSQL=sSQL+ " S.PostCode AS SourcePostCode," sSQL=sSQL+ " S.Latitude AS SourceLatitude," sSQL=sSQL+ " S.Longitude AS SourceLongitude," sSQL=sSQL+ " T.Country AS TargetCountry," sSQL=sSQL+ " T.PostCode AS TargetPostCode," sSQL=sSQL+ " T.PlaceName AS TargetPlaceName," sSQL=sSQL+ " T.Latitude AS TargetLatitude," sSQL=sSQL+ " T.Longitude AS TargetLongitude" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " Assess\_RoutePointsPostCode AS S" sSQL=sSQL+ " ,"

sSQL=sSQL+ " Assess\_RoutePointsPLaceName AS T" sSQL=sSQL+ " WHERE"

sSQL=sSQL+ " S.Country = T.Country" sSQL=sSQL+ " AND"

sSQL=sSQL+ " S.PostCode = T.PostCode" sSQL=sSQL+ " AND"

sSQL=sSQL+ " S.Country in ('GB','DE','BE','AU','US','IN')" sSQL=sSQL+ " AND"

sSQL=sSQL+ " T.Country in ('GB','DE','BE','AU','US','IN');"

sql.execute(sSQL,conn)

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

print('Loading :',sDatabaseName,' Table:',sView) sSQL=" SELECT "

sSQL=sSQL+ " \*" sSQL=sSQL+ " FROM" sSQL=sSQL+ " " + sView + ";"

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

RoutePlaceName['Distance']=RoutePlaceName.apply(lambda row: round( vincenty((row['SourceLatitude'],row['SourceLongitude']),

(row['TargetLatitude'],row['TargetLongitude'])).miles

,4)

,axis=1)

print(RoutePlaceName.head(5)) ################################################################

G=nx.Graph() ################################################################

print('Countries:',RouteCountries.shape) for i in range(RouteCountries.shape[0]):

sNode0='C-' + RouteCountries['SourceCountry'][i] G.add\_node(sNode0,

Nodetype='Country', Country=RouteCountries['SourceCountry'][i], Latitude=round(RouteCountries['SourceLatitude'][i],4), Longitude=round(RouteCountries['SourceLongitude'][i],4))

sNode1='C-' + RouteCountries['TargetCountry'][i] G.add\_node(sNode1,

Nodetype='Country', Country=RouteCountries['TargetCountry'][i], Latitude=round(RouteCountries['TargetLatitude'][i],4), Longitude=round(RouteCountries['TargetLongitude'][i],4))

G.add\_edge(sNode0,sNode1,distance=round(RouteCountries['Distance'][i],3)) #print(sNode0,sNode1)

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

print('Post Code:',RoutePostCode.shape) for i in range(RoutePostCode.shape[0]):

sNode0='C-' + RoutePostCode['SourceCountry'][i] G.add\_node(sNode0,

Nodetype='Country', Country=RoutePostCode['SourceCountry'][i], Latitude=round(RoutePostCode['SourceLatitude'][i],4), Longitude=round(RoutePostCode['SourceLongitude'][i],4))

sNode1='P-' + RoutePostCode['TargetPostCode'][i] + '-' + RoutePostCode['TargetCountry'][i] G.add\_node(sNode1,

Nodetype='PostCode', Country=RoutePostCode['TargetCountry'][i], PostCode=RoutePostCode['TargetPostCode'][i], Latitude=round(RoutePostCode['TargetLatitude'][i],4), Longitude=round(RoutePostCode['TargetLongitude'][i],4))

G.add\_edge(sNode0,sNode1,distance=round(RoutePostCode['Distance'][i],3)) #print(sNode0,sNode1)

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

print('Place Name:',RoutePlaceName.shape) for i in range(RoutePlaceName.shape[0]):

sNode0='P-' + RoutePlaceName['TargetPostCode'][i] + '-' sNode0=sNode0 + RoutePlaceName['TargetCountry'][i] G.add\_node(sNode0,

Nodetype='PostCode', Country=RoutePlaceName['SourceCountry'][i], PostCode=RoutePlaceName['TargetPostCode'][i], Latitude=round(RoutePlaceName['SourceLatitude'][i],4), Longitude=round(RoutePlaceName['SourceLongitude'][i],4))

sNode1='L-' + RoutePlaceName['TargetPlaceName'][i] + '-' sNode1=sNode1 + RoutePlaceName['TargetPostCode'][i] + '-' sNode1=sNode1 + RoutePlaceName['TargetCountry'][i] G.add\_node(sNode1,

Nodetype='PlaceName', Country=RoutePlaceName['TargetCountry'][i], PostCode=RoutePlaceName['TargetPostCode'][i], PlaceName=RoutePlaceName['TargetPlaceName'][i], Latitude=round(RoutePlaceName['TargetLatitude'][i],4), Longitude=round(RoutePlaceName['TargetLongitude'][i],4))

G.add\_edge(sNode0,sNode1,distance=round(RoutePlaceName['Distance'][i],3)) #print(sNode0,sNode1)

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

sFileName=sFileDir + '/' + OutputFileName print('################################')

print('Storing :', sFileName) print('################################')

nx.write\_gml(G,sFileName) sFileName=sFileName +'.gz' nx.write\_gml(G,sFileName)

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

print('Path:', nx.shortest\_path(G,source='P-SW1-GB',target='P-01001-US',weight='distance'))

print('Path length:', nx.shortest\_path\_length(G,source='P-SW1-GB',target='P-01001-US',weight='distance')) print('Path length (1):', nx.shortest\_path\_length(G,source='P-SW1-GB',target='C-GB',weight='distance')) print('Path length (2):', nx.shortest\_path\_length(G,source='C-GB',target='C-US',weight='distance')) print('Path length (3):', nx.shortest\_path\_length(G,source='C-US',target='P-01001-US',weight='distance')) print('################################')

print('Routes from P-SW1-GB < 2: ', nx.single\_source\_shortest\_path(G,source='P-SW1-GB' ,cutoff=1)) print('Routes from P-01001-US < 2: ', nx.single\_source\_shortest\_path(G,source='P-01001-US' ,cutoff=1)) print('################################') ################################################################ print('################')

print('Vacuum Database') sSQL="VACUUM;"

sql.execute(sSQL,conn) print('################')

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

### Output:

You can now query features out of a graph, such as shortage pathsbetween locations and paths from a given location, using Assess\_Best\_Logistics.gml with appropirate application.

## Write a Python / R program to decide the best packing option to ship in container from the given data.

Hillman wants to introduce new shipping containers into its logistics strategy. This program will through a process of assessing the possible container sizes.This example introduces features with ranges or tolerances.

Open Python editor and create a file named Assess-Shipping-Containers.py in directory C:\VKHCG\03-Hillman\02-Assess

import sys import os

import pandas as pd import sqlite3 as sq

from pandas.io import sql

################################################################ Base='C:/VKHCG'

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

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

Company='03-Hillman'

InputDir='01-Retrieve/01-EDS/02-Python' InputFileName1='Retrieve\_Product.csv' InputFileName2='Retrieve\_Box.csv' InputFileName3='Retrieve\_Container.csv' EDSDir='02-Assess/01-EDS' OutputDir=EDSDir + '/02-Python'

OutputFileName='Assess\_Shipping\_Containers.csv' ################################################################

sFileDir=Base + '/' + Company + '/' + EDSDir if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sFileDir=Base + '/' + Company + '/' + OutputDir if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sDataBaseDir=Base + '/' + Company + '/02-Assess/SQLite' if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir) ################################################################

sDatabaseName=sDataBaseDir + '/hillman.db' conn = sq.connect(sDatabaseName)

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

### Import Product Data ################################################################

sFileName=Base + '/' + Company + '/' + InputDir + '/' + InputFileName1 print('###########')

print('Loading :',sFileName) ProductRawData=pd.read\_csv(sFileName,

header=0, low\_memory=False, encoding="latin-1"

)

ProductRawData.drop\_duplicates(subset=None, keep='first', inplace=True) ProductRawData.index.name = 'IDNumber' ProductData=ProductRawData[ProductRawData.Length <= 0.5].head(10) print('Loaded Product :',ProductData.columns.values) print('################################') ################################################################ print('################')

sTable='Assess\_Product'

print('Storing :',sDatabaseName,' Table:',sTable) ProductData.to\_sql(sTable, conn, if\_exists="replace") print('################')

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

print(ProductData.head()) print('################################')

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

### Import Box Data

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

sFileName=Base + '/' + Company + '/' + InputDir + '/' + InputFileName2 print('###########')

print('Loading :',sFileName) BoxRawData=pd.read\_csv(sFileName,

header=0, low\_memory=False, encoding="latin-1"

)

BoxRawData.drop\_duplicates(subset=None, keep='first', inplace=True) BoxRawData.index.name = 'IDNumber' BoxData=BoxRawData[BoxRawData.Length <= 1].head(1000) print('Loaded Product :',BoxData.columns.values) print('################################')

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

sTable='Assess\_Box'

print('Storing :',sDatabaseName,' Table:',sTable) BoxData.to\_sql(sTable, conn, if\_exists="replace") print('################')

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

print(BoxData.head()) print('################################')

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

### Import Container Data ################################################################

sFileName=Base + '/' + Company + '/' + InputDir + '/' + InputFileName3 print('###########')

print('Loading :',sFileName) ContainerRawData=pd.read\_csv(sFileName,

header=0, low\_memory=False, encoding="latin-1"

)

ContainerRawData.drop\_duplicates(subset=None, keep='first', inplace=True) ContainerRawData.index.name = 'IDNumber' ContainerData=ContainerRawData[ContainerRawData.Length <= 2].head(10) print('Loaded Product :',ContainerData.columns.values) print('################################') ################################################################ print('################')

sTable='Assess\_Container'

print('Storing :',sDatabaseName,' Table:',sTable) BoxData.to\_sql(sTable, conn, if\_exists="replace") print('################')

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

print(ContainerData.head()) print('################################')

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

### Fit Product in Box ################################################################ print('################')

sView='Assess\_Product\_in\_Box'

print('Creating :',sDatabaseName,' View:',sView)

sSQL="DROP VIEW IF EXISTS " + sView + ";"

sql.execute(sSQL,conn)

sSQL="CREATE VIEW " + sView + " AS" sSQL=sSQL+ " SELECT"

sSQL=sSQL+ " P.UnitNumber AS ProductNumber," sSQL=sSQL+ " B.UnitNumber AS BoxNumber," sSQL=sSQL+ " (B.Thickness \* 1000) AS PackSafeCode,"

sSQL=sSQL+ " (B.BoxVolume - P.ProductVolume) AS PackFoamVolume,"

sSQL=sSQL+ " ((B.Length\*10) \* (B.Width\*10) \* (B.Height\*10)) \* 167 AS Air\_Dimensional\_Weight," sSQL=sSQL+ " ((B.Length\*10) \* (B.Width\*10) \* (B.Height\*10)) \* 333 AS Road\_Dimensional\_Weight," sSQL=sSQL+ " ((B.Length\*10) \* (B.Width\*10) \* (B.Height\*10)) \* 1000 AS Sea\_Dimensional\_Weight," sSQL=sSQL+ " P.Length AS Product\_Length,"

sSQL=sSQL+ " P.Width AS Product\_Width," sSQL=sSQL+ " P.Height AS Product\_Height,"

sSQL=sSQL+ " P.ProductVolume AS Product\_cm\_Volume,"

sSQL=sSQL+ " ((P.Length\*10) \* (P.Width\*10) \* (P.Height\*10)) AS Product\_ccm\_Volume," sSQL=sSQL+ " (B.Thickness \* 0.95) AS Minimum\_Pack\_Foam,"

sSQL=sSQL+ " (B.Thickness \* 1.05) AS Maximum\_Pack\_Foam,"

sSQL=sSQL+ " B.Length - (B.Thickness \* 1.10) AS Minimum\_Product\_Box\_Length," sSQL=sSQL+ " B.Length - (B.Thickness \* 0.95) AS Maximum\_Product\_Box\_Length," sSQL=sSQL+ " B.Width - (B.Thickness \* 1.10) AS Minimum\_Product\_Box\_Width," sSQL=sSQL+ " B.Width - (B.Thickness \* 0.95) AS Maximum\_Product\_Box\_Width," sSQL=sSQL+ " B.Height - (B.Thickness \* 1.10) AS Minimum\_Product\_Box\_Height," sSQL=sSQL+ " B.Height - (B.Thickness \* 0.95) AS Maximum\_Product\_Box\_Height," sSQL=sSQL+ " B.Length AS Box\_Length,"

sSQL=sSQL+ " B.Width AS Box\_Width," sSQL=sSQL+ " B.Height AS Box\_Height," sSQL=sSQL+ " B.BoxVolume AS Box\_cm\_Volume,"

sSQL=sSQL+ " ((B.Length\*10) \* (B.Width\*10) \* (B.Height\*10)) AS Box\_ccm\_Volume," sSQL=sSQL+ " (2 \* B.Length \* B.Width) + (2 \* B.Length \* B.Height) + (2 \* B.Width \* B.Height) AS Box\_sqm\_Area,"

sSQL=sSQL+ " ((B.Length\*10) \* (B.Width\*10) \* (B.Height\*10)) \* 3.5 AS Box\_A\_Max\_Kg\_Weight," sSQL=sSQL+ " ((B.Length\*10) \* (B.Width\*10) \* (B.Height\*10)) \* 7.7 AS Box\_B\_Max\_Kg\_Weight," sSQL=sSQL+ " ((B.Length\*10) \* (B.Width\*10) \* (B.Height\*10)) \* 10.0 AS Box\_C\_Max\_Kg\_Weight" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " Assess\_Product as P" sSQL=sSQL+ " ,"

sSQL=sSQL+ " Assess\_Box as B" sSQL=sSQL+ " WHERE"

sSQL=sSQL+ " P.Length >= (B.Length - (B.Thickness \* 1.10))" sSQL=sSQL+ " AND"

sSQL=sSQL+ " P.Width >= (B.Width - (B.Thickness \* 1.10))" sSQL=sSQL+ " AND"

sSQL=sSQL+ " P.Height >= (B.Height - (B.Thickness \* 1.10))" sSQL=sSQL+ " AND"

sSQL=sSQL+ " P.Length <= (B.Length - (B.Thickness \* 0.95))" sSQL=sSQL+ " AND"

sSQL=sSQL+ " P.Width <= (B.Width - (B.Thickness \* 0.95))" sSQL=sSQL+ " AND"

sSQL=sSQL+ " P.Height <= (B.Height - (B.Thickness \* 0.95))" sSQL=sSQL+ " AND"

sSQL=sSQL+ " (B.Height - B.Thickness) >= 0" sSQL=sSQL+ " AND"

sSQL=sSQL+ " (B.Width - B.Thickness) >= 0" sSQL=sSQL+ " AND"

sSQL=sSQL+ " (B.Height - B.Thickness) >= 0" sSQL=sSQL+ " AND"

sSQL=sSQL+ " B.BoxVolume >= P.ProductVolume;" sql.execute(sSQL,conn)

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

### Fit Box in Pallet ################################################################

t=0

for l in range(2,8):

for w in range(2,8): for h in range(4):

t += 1

PalletLine=[('IDNumber',[t]),

('ShipType', ['Pallet']),

('UnitNumber', ('L-'+format(t,"06d"))),

('Box\_per\_Length',(format(2\*\*l,"4d"))),

('Box\_per\_Width',(format(2\*\*w,"4d"))),

('Box\_per\_Height',(format(2\*\*h,"4d")))] if t==1:

PalletFrame = pd.DataFrame.from\_items(PalletLine) else:

PalletRow = pd.DataFrame.from\_items(PalletLine) PalletFrame = PalletFrame.append(PalletRow)

PalletFrame.set\_index(['IDNumber'],inplace=True) ################################################################

PalletFrame.head() print('################################')

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

### Fit Box on Pallet ################################################################ print('################')

sView='Assess\_Box\_on\_Pallet'

print('Creating :',sDatabaseName,' View:',sView) sSQL="DROP VIEW IF EXISTS " + sView + ";"

sql.execute(sSQL,conn)

sSQL="CREATE VIEW " + sView + " AS" sSQL=sSQL+ " SELECT DISTINCT"

sSQL=sSQL+ " P.UnitNumber AS PalletNumber," sSQL=sSQL+ " B.UnitNumber AS BoxNumber,"

sSQL=sSQL+ " round(B.Length\*P.Box\_per\_Length,3) AS Pallet\_Length," sSQL=sSQL+ " round(B.Width\*P.Box\_per\_Width,3) AS Pallet\_Width," sSQL=sSQL+ " round(B.Height\*P.Box\_per\_Height,3) AS Pallet\_Height,"

sSQL=sSQL+ " P.Box\_per\_Length \* P.Box\_per\_Width \* P.Box\_per\_Height AS Pallet\_Boxes" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " Assess\_Box as B" sSQL=sSQL+ " ,"

sSQL=sSQL+ " Assess\_Pallet as P" sSQL=sSQL+ " WHERE"

sSQL=sSQL+ " round(B.Length\*P.Box\_per\_Length,3) <= 20" sSQL=sSQL+ " AND"

sSQL=sSQL+ " round(B.Width\*P.Box\_per\_Width,3) <= 9" sSQL=sSQL+ " AND"

sSQL=sSQL+ " round(B.Height\*P.Box\_per\_Height,3) <= 5;" sql.execute(sSQL,conn) ################################################################

sTables=['Assess\_Product\_in\_Box','Assess\_Box\_on\_Pallet'] for sTable in sTables:

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

print('Loading :',sDatabaseName,' Table:',sTable) sSQL=" SELECT "

sSQL=sSQL+ " \*"

sSQL=sSQL+ " FROM"

sSQL=sSQL+ " " + sTable + ";" SnapShotData=pd.read\_sql\_query(sSQL, conn) print('################')

sTableOut=sTable + '\_SnapShot'

print('Storing :',sDatabaseName,' Table:',sTable) SnapShotData.to\_sql(sTableOut, conn, if\_exists="replace") print('################')

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

### Fit Pallet in Container ################################################################

sTables=['Length','Width','Height'] for sTable in sTables:

sView='Assess\_Pallet\_in\_Container\_' + sTable print('Creating :',sDatabaseName,' View:',sView) sSQL="DROP VIEW IF EXISTS " + sView + ";"

sql.execute(sSQL,conn)

sSQL="CREATE VIEW " + sView + " AS" sSQL=sSQL+ " SELECT DISTINCT"

sSQL=sSQL+ " C.UnitNumber AS ContainerNumber," sSQL=sSQL+ " P.PalletNumber,"

sSQL=sSQL+ " P.BoxNumber,"

sSQL=sSQL+ " round(C." + sTable + "/P.Pallet\_" + sTable + ",0)" sSQL=sSQL+ " AS Pallet\_per\_" + sTable + ","

sSQL=sSQL+ " round(C." + sTable + "/P.Pallet\_" + sTable + ",0)" sSQL=sSQL+ " \* P.Pallet\_Boxes AS Pallet\_" + sTable + "\_Boxes," sSQL=sSQL+ " P.Pallet\_Boxes"

sSQL=sSQL+ " FROM"

sSQL=sSQL+ " Assess\_Container as C" sSQL=sSQL+ " ,"

sSQL=sSQL+ " Assess\_Box\_on\_Pallet\_SnapShot as P" sSQL=sSQL+ " WHERE"

sSQL=sSQL+ " round(C.Length/P.Pallet\_Length,0) > 0" sSQL=sSQL+ " AND"

sSQL=sSQL+ " round(C.Width/P.Pallet\_Width,0) > 0" sSQL=sSQL+ " AND"

sSQL=sSQL+ " round(C.Height/P.Pallet\_Height,0) > 0;" sql.execute(sSQL,conn)

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

print('Loading :',sDatabaseName,' Table:',sView) sSQL=" SELECT "

sSQL=sSQL+ " \*" sSQL=sSQL+ " FROM" sSQL=sSQL+ " " + sView + ";"

SnapShotData=pd.read\_sql\_query(sSQL, conn) print('################')

sTableOut= sView + '\_SnapShot'

print('Storing :',sDatabaseName,' Table:',sTableOut) SnapShotData.to\_sql(sTableOut, conn, if\_exists="replace") print('################')

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

sView='Assess\_Pallet\_in\_Container' print('Creating :',sDatabaseName,' View:',sView) sSQL="DROP VIEW IF EXISTS " + sView + ";"

sql.execute(sSQL,conn)

sSQL="CREATE VIEW " + sView + " AS" sSQL=sSQL+ " SELECT"

sSQL=sSQL+ " CL.ContainerNumber," sSQL=sSQL+ " CL.PalletNumber," sSQL=sSQL+ " CL.BoxNumber,"

sSQL=sSQL+ " CL.Pallet\_Boxes AS Boxes\_per\_Pallet," sSQL=sSQL+ " CL.Pallet\_per\_Length,"

sSQL=sSQL+ " CW.Pallet\_per\_Width," sSQL=sSQL+ " CH.Pallet\_per\_Height,"

sSQL=sSQL+ " CL.Pallet\_Length\_Boxes \* CW.Pallet\_Width\_Boxes \* CH.Pallet\_Height\_Boxes AS Container\_Boxes"

sSQL=sSQL+ " FROM"

sSQL=sSQL+ " Assess\_Pallet\_in\_Container\_Length\_SnapShot as CL" sSQL=sSQL+ " JOIN"

sSQL=sSQL+ " Assess\_Pallet\_in\_Container\_Width\_SnapShot as CW" sSQL=sSQL+ " ON"

sSQL=sSQL+ " CL.ContainerNumber = CW.ContainerNumber" sSQL=sSQL+ " AND"

sSQL=sSQL+ " CL.PalletNumber = CW.PalletNumber" sSQL=sSQL+ " AND"

sSQL=sSQL+ " CL.BoxNumber = CW.BoxNumber" sSQL=sSQL+ " JOIN"

sSQL=sSQL+ " Assess\_Pallet\_in\_Container\_Height\_SnapShot as CH" sSQL=sSQL+ " ON"

sSQL=sSQL+ " CL.ContainerNumber = CH.ContainerNumber" sSQL=sSQL+ " AND"

sSQL=sSQL+ " CL.PalletNumber = CH.PalletNumber" sSQL=sSQL+ " AND"

sSQL=sSQL+ " CL.BoxNumber = CH.BoxNumber;" sql.execute(sSQL,conn)

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

sTables=['Assess\_Product\_in\_Box','Assess\_Pallet\_in\_Container'] for sTable in sTables:

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

print('Loading :',sDatabaseName,' Table:',sTable) sSQL=" SELECT "

sSQL=sSQL+ " \*" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " " + sTable + ";" PackData=pd.read\_sql\_query(sSQL, conn) print('################')

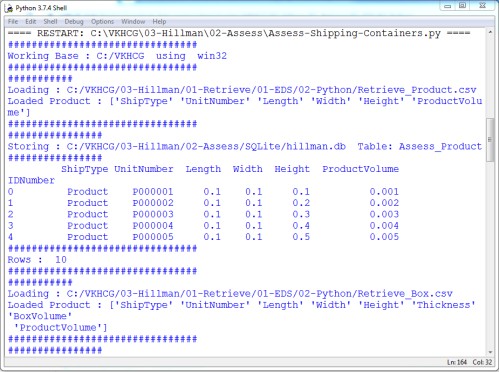
print(PackData) print('################')

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

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

sFileName=sFileDir + '/' + sTable + '.csv' print(sFileName) PackData.to\_csv(sFileName, index = False)

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



## Write a Python program to create a delivery route using the given data.

Creating a Delivery Route

Hillman requires the complete grid plan of the delivery routes for the company, to ensure the suppliers, warehouses, shops, and customers can be reached by its new strategy. This new plan will enable the optimum routes between suppliers, warehouses, shops, and customers.

Open Python editor and create a file named Assess-Shipping-Routes.py in directory C:\VKHCG\03-Hillman\02-Assess.

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

import sys import os

import pandas as pd import sqlite3 as sq

from pandas.io import sql import networkx as nx

from geopy.distance import vincenty ################################################################

nMax=3 nMaxPath=10 nSet=False nVSet=False

################################################################ Base='C:/VKHCG' ################################################################ print('################################')

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

Company='03-Hillman'

InputDir1='01-Retrieve/01-EDS/01-R' InputDir2='01-Retrieve/01-EDS/02-Python'

InputFileName1='Retrieve\_GB\_Postcode\_Warehouse.csv' InputFileName2='Retrieve\_GB\_Postcodes\_Shops.csv' EDSDir='02-Assess/01-EDS'

OutputDir=EDSDir + '/02-Python' OutputFileName1='Assess\_Shipping\_Routes.gml' OutputFileName2='Assess\_Shipping\_Routes.txt' ################################################################

sFileDir=Base + '/' + Company + '/' + EDSDir if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sFileDir=Base + '/' + Company + '/' + OutputDir if not os.path.exists(sFileDir):

os.makedirs(sFileDir) ################################################################

sDataBaseDir=Base + '/' + Company + '/02-Assess/SQLite'

if not os.path.exists(sDataBaseDir): os.makedirs(sDataBaseDir)

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

sDatabaseName=sDataBaseDir + '/hillman.db' conn = sq.connect(sDatabaseName)

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

### Import Warehouse Data ################################################################

sFileName=Base + '/' + Company + '/' + InputDir1 + '/' + InputFileName1 print('###########')

print('Loading :',sFileName) WarehouseRawData=pd.read\_csv(sFileName,

header=0, low\_memory=False, encoding="latin-1"

)

WarehouseRawData.drop\_duplicates(subset=None, keep='first', inplace=True) WarehouseRawData.index.name = 'IDNumber' WarehouseData=WarehouseRawData.head(nMax) WarehouseData=WarehouseData.append(WarehouseRawData.tail(nMax))

WarehouseData=WarehouseData.append(WarehouseRawData[WarehouseRawData.postcode=='KA13']) if nSet==True:

WarehouseData=WarehouseData.append(WarehouseRawData[WarehouseRawData.postcode=='SW1W']) WarehouseData.drop\_duplicates(subset=None, keep='first', inplace=True)

print('Loaded Warehouses :',WarehouseData.columns.values) print('################################') ################################################################ print('################')

sTable='Assess\_Warehouse\_UK'

print('Storing :',sDatabaseName,' Table:',sTable) WarehouseData.to\_sql(sTable, conn, if\_exists="replace") print('################') ################################################################

print(WarehouseData.head()) print('################################')

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

### Import Shop Data ################################################################

sFileName=Base + '/' + Company + '/' + InputDir1 + '/' + InputFileName2 print('###########')

print('Loading :',sFileName) ShopRawData=pd.read\_csv(sFileName,

header=0, low\_memory=False, encoding="latin-1"

)

ShopRawData.drop\_duplicates(subset=None, keep='first', inplace=True) ShopRawData.index.name = 'IDNumber'

ShopData=ShopRawData

print('Loaded Shops :',ShopData.columns.values) print('################################') ################################################################ print('################')

sTable='Assess\_Shop\_UK'

print('Storing :',sDatabaseName,' Table:',sTable) ShopData.to\_sql(sTable, conn, if\_exists="replace")

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

print(ShopData.head()) print('################################')

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

### Connect HQ ################################################################ print('################')

sView='Assess\_HQ'

print('Creating :',sDatabaseName,' View:',sView) sSQL="DROP VIEW IF EXISTS " + sView + ";"

sql.execute(sSQL,conn)

sSQL="CREATE VIEW " + sView + " AS" sSQL=sSQL+ " SELECT"

sSQL=sSQL+ " W.postcode AS HQ\_PostCode," sSQL=sSQL+ " 'HQ-' || W.postcode AS HQ\_Name," sSQL=sSQL+ " round(W.latitude,6) AS HQ\_Latitude," sSQL=sSQL+ " round(W.longitude,6) AS HQ\_Longitude" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " Assess\_Warehouse\_UK as W" sSQL=sSQL+ " WHERE"

sSQL=sSQL+ " TRIM(W.postcode) in ('KA13','SW1W');"

sql.execute(sSQL,conn) ################################################################

### Connect Warehouses ################################################################ print('################')

sView='Assess\_Warehouse'

print('Creating :',sDatabaseName,' View:',sView) sSQL="DROP VIEW IF EXISTS " + sView + ";"

sql.execute(sSQL,conn)

sSQL="CREATE VIEW " + sView + " AS" sSQL=sSQL+ " SELECT"

sSQL=sSQL+ " W.postcode AS Warehouse\_PostCode," sSQL=sSQL+ " 'WH-' || W.postcode AS Warehouse\_Name," sSQL=sSQL+ " round(W.latitude,6) AS Warehouse\_Latitude," sSQL=sSQL+ " round(W.longitude,6) AS Warehouse\_Longitude" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " Assess\_Warehouse\_UK as W;" sql.execute(sSQL,conn)

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

### Connect Warehouse to Shops by PostCode

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

sView='Assess\_Shop'

print('Creating :',sDatabaseName,' View:',sView) sSQL="DROP VIEW IF EXISTS " + sView + ";"

sql.execute(sSQL,conn)

sSQL="CREATE VIEW " + sView + " AS" sSQL=sSQL+ " SELECT"

sSQL=sSQL+ " TRIM(S.postcode) AS Shop\_PostCode,"

sSQL=sSQL+ " 'SP-' || TRIM(S.FirstCode) || '-' || TRIM(S.SecondCode) AS Shop\_Name," sSQL=sSQL+ " TRIM(S.FirstCode) AS Warehouse\_PostCode,"

sSQL=sSQL+ " round(S.latitude,6) AS Shop\_Latitude," sSQL=sSQL+ " round(S.longitude,6) AS Shop\_Longitude"

sSQL=sSQL+ " FROM"

sSQL=sSQL+ " Assess\_Warehouse\_UK as W" sSQL=sSQL+ " JOIN"

sSQL=sSQL+ " Assess\_Shop\_UK as S" sSQL=sSQL+ " ON"

sSQL=sSQL+ " TRIM(W.postcode) = TRIM(S.FirstCode);" sql.execute(sSQL,conn) ################################################################ ################################################################

G=nx.Graph() ################################################################ print('################')

sTable = 'Assess\_HQ'

print('Loading :',sDatabaseName,' Table:',sTable) sSQL=" SELECT DISTINCT"

sSQL=sSQL+ " \*" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " " + sTable + ";" RouteData=pd.read\_sql\_query(sSQL, conn) print('################')

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

print(RouteData.head()) print('################################')

print('HQ Rows : ',RouteData.shape[0]) print('################################') ################################################################

for i in range(RouteData.shape[0]): sNode0=RouteData['HQ\_Name'][i] G.add\_node(sNode0,

Nodetype='HQ', PostCode=RouteData['HQ\_PostCode'][i], Latitude=round(RouteData['HQ\_Latitude'][i],6), Longitude=round(RouteData['HQ\_Longitude'][i],6))

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

sTable = 'Assess\_Warehouse'

print('Loading :',sDatabaseName,' Table:',sTable) sSQL=" SELECT DISTINCT"

sSQL=sSQL+ " \*" sSQL=sSQL+ " FROM"

sSQL=sSQL+ " " + sTable + ";" RouteData=pd.read\_sql\_query(sSQL, conn) print('################')

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

print(RouteData.head()) print('################################')

print('Warehouse Rows : ',RouteData.shape[0]) print('################################')

for i in range(RouteData.shape[0]): sNode0=RouteData['Warehouse\_Name'][i] G.add\_node(sNode0,

Nodetype='Warehouse', PostCode=RouteData['Warehouse\_PostCode'][i], Latitude=round(RouteData['Warehouse\_Latitude'][i],6), Longitude=round(RouteData['Warehouse\_Longitude'][i],6))

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

sTable = 'Assess\_Shop'

print('Loading :',sDatabaseName,' Table:',sTable) sSQL=" SELECT DISTINCT"

sSQL=sSQL+ " \*"

sSQL=sSQL+ " FROM"

sSQL=sSQL+ " " + sTable + ";" RouteData=pd.read\_sql\_query(sSQL, conn) print('################')

print(RouteData.head()) print('################################')

print('Shop Rows : ',RouteData.shape[0]) print('################################')

for i in range(RouteData.shape[0]): sNode0=RouteData['Shop\_Name'][i] G.add\_node(sNode0,

Nodetype='Shop', PostCode=RouteData['Shop\_PostCode'][i], WarehousePostCode=RouteData['Warehouse\_PostCode'][i], Latitude=round(RouteData['Shop\_Latitude'][i],6), Longitude=round(RouteData['Shop\_Longitude'][i],6))

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

## Create Edges ################################################################ print('################################')

print('Loading Edges') print('################################')

for sNode0 in nx.nodes\_iter(G): for sNode1 in nx.nodes\_iter(G):

if G.node[sNode0]['Nodetype']=='HQ' and \ G.node[sNode1]['Nodetype']=='HQ' and \ sNode0 != sNode1:

distancemeters=round(\

vincenty(\

(\

G.node[sNode0]['Latitude'],\ G.node[sNode0]['Longitude']\

),\ (\

G.node[sNode1]['Latitude']\

,\ G.node[sNode1]['Longitude']\

)\

).meters\

,0)

distancemiles=round(\

vincenty(\

(\

G.node[sNode0]['Latitude'],\ G.node[sNode0]['Longitude']\

),\ (\

G.node[sNode1]['Latitude']\

,\ G.node[sNode1]['Longitude']\

)\

).miles\

,3)

if distancemiles >= 0.05:

cost = round(150+(distancemiles \* 2.5),6) vehicle='V001'

else:

cost = round(2+(distancemiles \* 0.10),6)

vehicle='ForkLift'

G.add\_edge(sNode0,sNode1,DistanceMeters=distancemeters, \ DistanceMiles=distancemiles, \ Cost=cost,Vehicle=vehicle)

if nVSet==True:

print('Edge-H-H:',sNode0,' to ', sNode1, \ ' Distance:',distancemeters,'meters',\

distancemiles,'miles','Cost', cost,'Vehicle',vehicle)

if G.node[sNode0]['Nodetype']=='HQ' and \ G.node[sNode1]['Nodetype']=='Warehouse' and \ sNode0 != sNode1:

distancemeters=round(\

vincenty(\

(\

G.node[sNode0]['Latitude'],\ G.node[sNode0]['Longitude']\

),\ (\

G.node[sNode1]['Latitude']\

,\ G.node[sNode1]['Longitude']\

)\

).meters\

,0)

distancemiles=round(\

vincenty(\

(\

G.node[sNode0]['Latitude'],\ G.node[sNode0]['Longitude']\

),\ (\

G.node[sNode1]['Latitude']\

,\ G.node[sNode1]['Longitude']\

)\

).miles\

,3)

if distancemiles >= 10:

cost = round(50+(distancemiles \* 2),6) vehicle='V002'

else:

cost = round(5+(distancemiles \* 1.5),6) vehicle='V003'

if distancemiles <= 50: G.add\_edge(sNode0,sNode1,DistanceMeters=distancemeters, \

DistanceMiles=distancemiles, \ Cost=cost,Vehicle=vehicle)

if nVSet==True:

print('Edge-H-W:',sNode0,' to ', sNode1, \ ' Distance:',distancemeters,'meters',\

distancemiles,'miles','Cost', cost,'Vehicle',vehicle)

if nSet==True and \ G.node[sNode0]['Nodetype']=='Warehouse' and \ G.node[sNode1]['Nodetype']=='Warehouse' and \ sNode0 != sNode1:

distancemeters=round(\

vincenty(\

,0)

(\

G.node[sNode0]['Latitude'],\ G.node[sNode0]['Longitude']\

),\ (\

G.node[sNode1]['Latitude']\

,\ G.node[sNode1]['Longitude']\

)\

).meters\

distancemiles=round(\

vincenty(\

(\

G.node[sNode0]['Latitude'],\ G.node[sNode0]['Longitude']\

),\ (\

G.node[sNode1]['Latitude']\

,\ G.node[sNode1]['Longitude']\

)\

).miles\

,3)

if distancemiles >= 10:

cost = round(50+(distancemiles \* 1.10),6) vehicle='V004'

else:

cost = round(5+(distancemiles \* 1.05),6) vehicle='V005'

if distancemiles <= 20: G.add\_edge(sNode0,sNode1,DistanceMeters=distancemeters, \

DistanceMiles=distancemiles, \ Cost=cost,Vehicle=vehicle)

if nVSet==True:

print('Edge-W-W:',sNode0,' to ', sNode1, \ ' Distance:',distancemeters,'meters',\

distancemiles,'miles','Cost', cost,'Vehicle',vehicle)

if G.node[sNode0]['Nodetype']=='Warehouse' and \ G.node[sNode1]['Nodetype']=='Shop' and \ G.node[sNode0]['PostCode']==G.node[sNode1]['WarehousePostCode'] and \ sNode0 != sNode1:

distancemeters=round(\

vincenty(\

(\

G.node[sNode0]['Latitude'],\ G.node[sNode0]['Longitude']\

),\ (\

G.node[sNode1]['Latitude']\

,\ G.node[sNode1]['Longitude']\

)\

).meters\

,0)

distancemiles=round(\

vincenty(\

(\

G.node[sNode0]['Latitude'],\ G.node[sNode0]['Longitude']\

),\ (\

G.node[sNode1]['Latitude']\

,\ G.node[sNode1]['Longitude']\

)\

).miles\

,3)

if distancemiles >= 10:

cost = round(50+(distancemiles \* 1.50),6) vehicle='V006'

else:

cost = round(5+(distancemiles \* 0.75),6) vehicle='V007'

if distancemiles <= 10: G.add\_edge(sNode0,sNode1,DistanceMeters=distancemeters, \

DistanceMiles=distancemiles, \ Cost=cost,Vehicle=vehicle)

if nVSet==True:

print('Edge-W-S:',sNode0,' to ', sNode1, \ ' Distance:',distancemeters,'meters',\

distancemiles,'miles','Cost', cost,'Vehicle',vehicle)

if nSet==True and \ G.node[sNode0]['Nodetype']=='Shop' and \ G.node[sNode1]['Nodetype']=='Shop' and \

G.node[sNode0]['WarehousePostCode']==G.node[sNode1]['WarehousePostCode'] and \ sNode0 != sNode1:

distancemeters=round(\

vincenty(\

(\

G.node[sNode0]['Latitude'],\ G.node[sNode0]['Longitude']\

),\ (\

G.node[sNode1]['Latitude']\

,\ G.node[sNode1]['Longitude']\

)\

).meters\

,0)

distancemiles=round(\

vincenty(\

(\

G.node[sNode0]['Latitude'],\ G.node[sNode0]['Longitude']\

),\ (\

G.node[sNode1]['Latitude']\

,\ G.node[sNode1]['Longitude']\

)\

).miles\

,3)

if distancemiles >= 0.05:

cost = round(5+(distancemiles \* 0.5),6) vehicle='V008'

else:

cost = round(1+(distancemiles \* 0.1),6) vehicle='V009'

if distancemiles <= 0.075: G.add\_edge(sNode0,sNode1,DistanceMeters=distancemeters, \

DistanceMiles=distancemiles, \ Cost=cost,Vehicle=vehicle)

if nVSet==True:

print('Edge-S-S:',sNode0,' to ', sNode1, \ ' Distance:',distancemeters,'meters',\

distancemiles,'miles','Cost', cost,'Vehicle',vehicle)

if nSet==True and \ G.node[sNode0]['Nodetype']=='Shop' and \ G.node[sNode1]['Nodetype']=='Shop' and \

G.node[sNode0]['WarehousePostCode']!=G.node[sNode1]['WarehousePostCode'] and \ sNode0 != sNode1:

distancemeters=round(\

vincenty(\

(\

G.node[sNode0]['Latitude'],\ G.node[sNode0]['Longitude']\

),\ (\

G.node[sNode1]['Latitude']\

,\ G.node[sNode1]['Longitude']\

)\

).meters\

,0)

distancemiles=round(\

vincenty(\

(\

G.node[sNode0]['Latitude'],\ G.node[sNode0]['Longitude']\

),\ (\

G.node[sNode1]['Latitude']\

,\ G.node[sNode1]['Longitude']\

)\

).miles\

,3)

cost = round(1+(distancemiles \* 0.1),6) vehicle='V010'

if distancemiles <= 0.025: G.add\_edge(sNode0,sNode1,DistanceMeters=distancemeters, \

DistanceMiles=distancemiles, \ Cost=cost,Vehicle=vehicle)

if nVSet==True:

print('Edge-S-S:',sNode0,' to ', sNode1, \ ' Distance:',distancemeters,'meters',\

distancemiles,'miles','Cost', cost,'Vehicle',vehicle) sFileName=sFileDir + '/' + OutputFileName1

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

print('Storing :', sFileName) print('################################')

nx.write\_gml(G,sFileName) sFileName=sFileName +'.gz' nx.write\_gml(G,sFileName) print('Nodes:',nx.number\_of\_nodes(G)) print('Edges:',nx.number\_of\_edges(G)) sFileName=sFileDir + '/' + OutputFileName2 print('################################')

print('Storing :', sFileName) print('################################')

## Create Paths print('################################')

print('Loading Paths') print('################################')

f = open(sFileName,'w') l=0

sline = 'ID|Cost|StartAt|EndAt|Path|Measure' if nVSet==True: print ('0', sline) f.write(sline+ '\n')

for sNode0 in nx.nodes\_iter(G): for sNode1 in nx.nodes\_iter(G):

if sNode0 != sNode1 and \

nx.has\_path(G, sNode0, sNode1)==True and \ nx.shortest\_path\_length(G, \

source=sNode0, \ target=sNode1, \

weight='DistanceMiles') < nMaxPath: l+=1

sID='{:.0f}'.format(l)

spath = ','.join(nx.shortest\_path(G, \ source=sNode0, \

target=sNode1, \ weight='DistanceMiles')) slength= '{:.6f}'.format(\ nx.shortest\_path\_length(G, \ source=sNode0, \ target=sNode1, \ weight='DistanceMiles'))

sline = sID + '|"DistanceMiles"|"' + sNode0 + '"|"' \

+ sNode1 + '"|"' + spath + '"|' + slength if nVSet==True: print (sline) f.write(sline + '\n')

l+=1

sID='{:.0f}'.format(l)

spath = ','.join(nx.shortest\_path(G, \ source=sNode0, \

target=sNode1, \ weight='DistanceMeters')) slength= '{:.6f}'.format(\ nx.shortest\_path\_length(G, \ source=sNode0, \ target=sNode1, \ weight='DistanceMeters'))

sline = sID + '|"DistanceMeters"|"' + sNode0 + '"|"' \

+ sNode1 + '"|"' + spath + '"|' + slength if nVSet==True: print (sline) f.write(sline + '\n')

l+=1

sID='{:.0f}'.format(l)

spath = ','.join(nx.shortest\_path(G, \ source=sNode0, \

target=sNode1, \ weight='Cost'))

slength= '{:.6f}'.format(\ nx.shortest\_path\_length(G, \ source=sNode0, \ target=sNode1, \ weight='Cost'))

sline = sID + '|"Cost"|"' + sNode0 + '"|"' \

+ sNode1 + '"|"' + spath + '"|' + slength if nVSet==True: print (sline) f.write(sline + '\n')

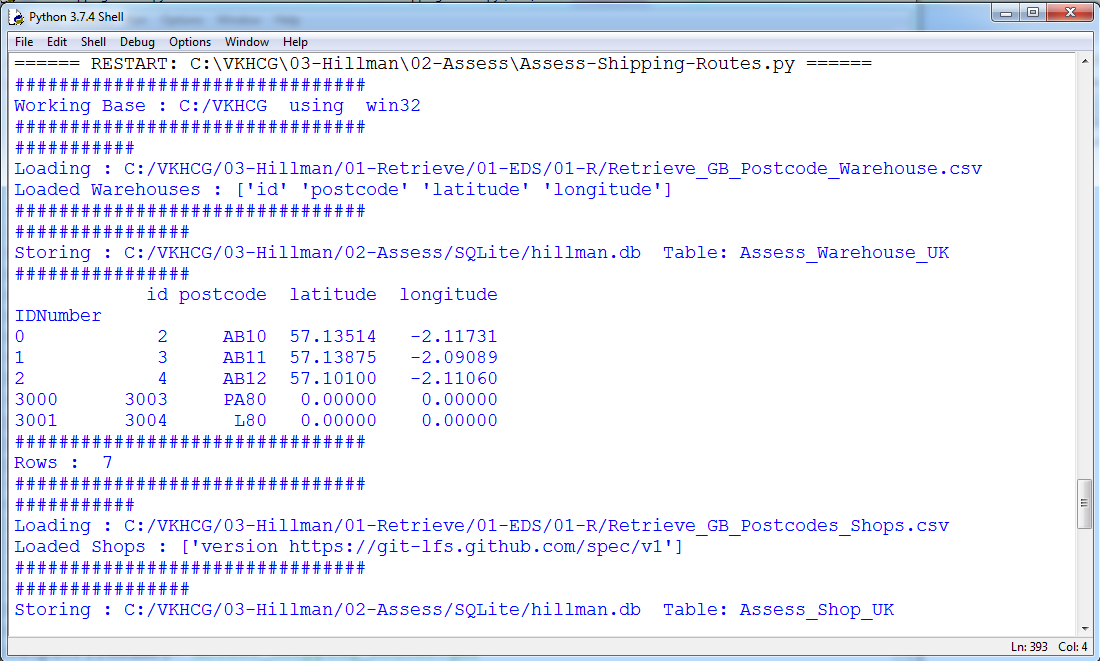
f.close()

print('Nodes:',nx.number\_of\_nodes(G)) print('Edges:',nx.number\_of\_edges(G)) print('Paths:',sID) print('################')

print('Vacuum Database') sSQL="VACUUM;"

sql.execute(sSQL,conn) print('################')

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



## Clark Ltd

Clark Ltd is the accountancy company that handles everything related to the VKHCG’s finances and personnel. Let’s investigate Clark with new knowledge.

1. **Write a Python program to create Simple forex trading planner from the given data.**

Simple Forex Trading Planner

Clark requires the assessment of the group’s forex data, for processing and data qualityissues. I will guide you through an example of a forex solution.

Open your Python editor and create a file named Assess-Forex.py in directory C:\VKHCG\04-Clark\02-Assess. ################################################################

import sys import os

import sqlite3 as sq import pandas as pd

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

Base='C:/VKHCG' print('################################')

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

Company='04-Clark'

sInputFileName1='01-Vermeulen/01-Retrieve/01-EDS/02-Python/Retrieve-Country-Currency.csv' sInputFileName2='04-Clark/01-Retrieve/01-EDS/01-R/Retrieve\_Euro\_EchangeRates.csv' ################################################################

sDataBaseDir=Base + '/' + Company + '/02-Assess/SQLite' if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir) ################################################################

sDatabaseName=sDataBaseDir + '/clark.db' conn = sq.connect(sDatabaseName)

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

### Import Country Data ################################################################

sFileName1=Base + '/' + sInputFileName1 print('################################')

print('Loading :',sFileName1) print('################################')

CountryRawData=pd.read\_csv(sFileName1,header=0,low\_memory=False, encoding="latin-1") CountryRawData.drop\_duplicates(subset=None, keep='first', inplace=True)

CountryData=CountryRawData

print('Loaded Company :',CountryData.columns.values) print('################################') ################################################################ print('################')

sTable='Assess\_Country'

print('Storing :',sDatabaseName,' Table:',sTable) CountryData.to\_sql(sTable, conn, if\_exists="replace") print('################')

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

print(CountryData.head()) print('################################')

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

### Import Forex Data ################################################################

sFileName2=Base + '/' + sInputFileName2 print('################################')

print('Loading :',sFileName2) print('################################')

ForexRawData=pd.read\_csv(sFileName2,header=0,low\_memory=False, encoding="latin-1") ForexRawData.drop\_duplicates(subset=None, keep='first', inplace=True) ForexData=ForexRawData.head(5)

print('Loaded Company :',ForexData.columns.values) print('################################') ################################################################ print('################')

sTable='Assess\_Forex'

print('Storing :',sDatabaseName,' Table:',sTable) ForexData.to\_sql(sTable, conn, if\_exists="replace") print('################')

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

print(ForexData.head()) print('################################')

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

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

sTable='Assess\_Forex'

print('Loading :',sDatabaseName,' Table:',sTable) sSQL="select distinct"

sSQL=sSQL+ " A.CodeIn" sSQL=sSQL+ " from"

sSQL=sSQL+ " Assess\_Forex as A;" CodeData=pd.read\_sql\_query(sSQL, conn) print('################')

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

for c in range(CodeData.shape[0]): print('################')

sTable='Assess\_Forex & 2x Country > ' + CodeData['CodeIn'][c] print('Loading :',sDatabaseName,' Table:',sTable)

sSQL="select distinct" sSQL=sSQL+ " A.Date," sSQL=sSQL+ " A.CodeIn,"

sSQL=sSQL+ " B.Country as CountryIn," sSQL=sSQL+ " B.Currency as CurrencyNameIn," sSQL=sSQL+ " A.CodeOut,"

sSQL=sSQL+ " C.Country as CountryOut," sSQL=sSQL+ " C.Currency as CurrencyNameOut," sSQL=sSQL+ " A.Rate"

sSQL=sSQL+ " from" sSQL=sSQL+ " Assess\_Forex as A" sSQL=sSQL+ " JOIN"

sSQL=sSQL+ " Assess\_Country as B" sSQL=sSQL+ " ON A.CodeIn = B.CurrencyCode" sSQL=sSQL+ " JOIN"

sSQL=sSQL+ " Assess\_Country as C" sSQL=sSQL+ " ON A.CodeOut = C.CurrencyCode" sSQL=sSQL+ " WHERE"

sSQL=sSQL+ " A.CodeIn ='" + CodeData['CodeIn'][c] + "';" ForexData=pd.read\_sql\_query(sSQL, conn).head(1000) print('################')

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

sTable='Assess\_Forex\_' + CodeData['CodeIn'][c] print('Storing :',sDatabaseName,' Table:',sTable) ForexData.to\_sql(sTable, conn, if\_exists="replace") print('################') print('################################')

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

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

### Output:

This will produce a set of demonstrated values onscreen by removing duplicate records and other related data processing.

### Write a Python program to process the balance sheet to ensure that only good data is processing.

Financials

Clark requires you to process the balance sheet for the VKHCG group companies. Go through a sample balance sheet data assessment, to ensure that only the good data is processed.

Open Python editor and create a file named Assess-Financials.py in directory C:\VKHCG\04-Clark\02-Assess.

import sys import os

import sqlite3 as sq import pandas as pd

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='01-Retrieve/01-EDS/01-R/Retrieve\_Profit\_And\_Loss.csv' sDataBaseDir=Base + '/' + Company + '/02-Assess/SQLite'

if not os.path.exists(sDataBaseDir): os.makedirs(sDataBaseDir)

sDatabaseName=sDataBaseDir + '/clark.db' conn = sq.connect(sDatabaseName)

### Import Financial Data

sFileName=Base + '/' + Company + '/' + sInputFileName print('################################')

print('Loading :',sFileName) print('################################')

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

FinancialData=FinancialRawData

print('Loaded Company :',FinancialData.columns.values) print('################################') print('################')

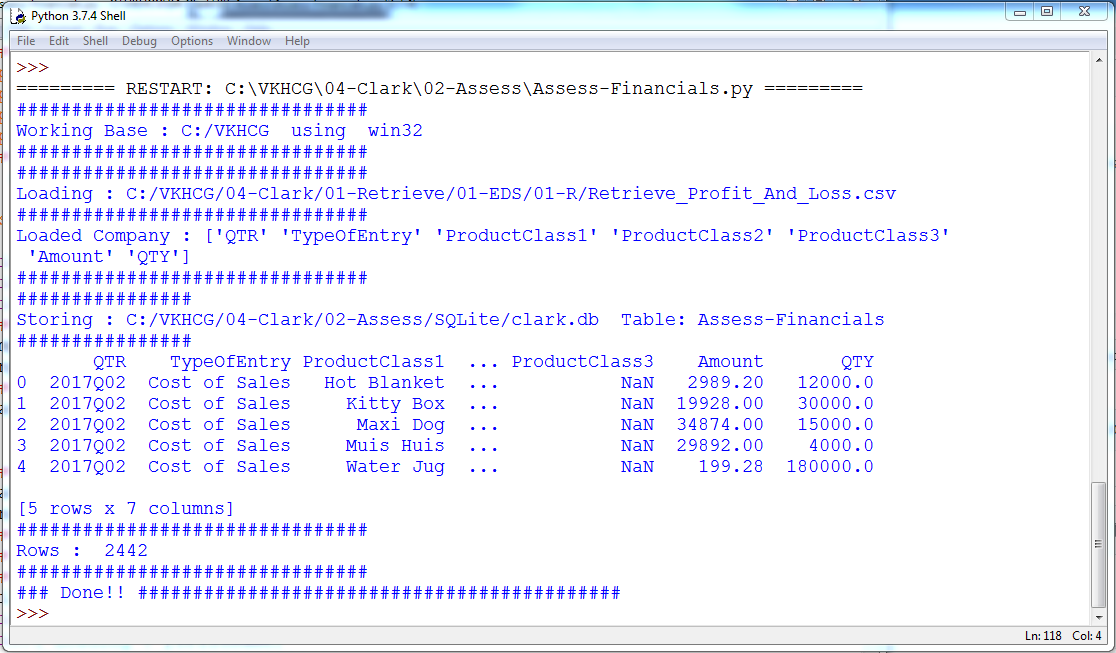
sTable='Assess-Financials'

print('Storing :',sDatabaseName,' Table:',sTable) FinancialData.to\_sql(sTable, conn, if\_exists="replace") print('################')

print(FinancialData.head()) print('################################')

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

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



### Write a Python program to store all master records for the financial calendar

Financial Calendar

Clark stores all the master records for the financial calendar. So we import thecalendar from the retrieve step’s data storage.

Open Python editor and create a file named Assess-Calendar.py in directory C:\VKHCG\04-Clark\02-Assess.

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

import sys import os

import sqlite3 as sq import pandas as pd

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

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' ################################################################

sDataBaseDirIn=Base + '/' + Company + '/01-Retrieve/SQLite' if not os.path.exists(sDataBaseDirIn):

os.makedirs(sDataBaseDirIn) sDatabaseNameIn=sDataBaseDirIn + '/clark.db' connIn = sq.connect(sDatabaseNameIn)

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

sDataBaseDirOut=Base + '/' + Company + '/01-Retrieve/SQLite' if not os.path.exists(sDataBaseDirOut):

os.makedirs(sDataBaseDirOut) sDatabaseNameOut=sDataBaseDirOut + '/clark.db' connOut = sq.connect(sDatabaseNameOut)

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

sTableIn='Retrieve\_Date'

sSQL='select \* FROM ' + sTableIn + ';' print('################')

sTableOut='Assess\_Time'

print('Loading :',sDatabaseNameIn,' Table:',sTableIn) dateRawData=pd.read\_sql\_query(sSQL, connIn) dateData=dateRawData

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

print('Load Rows : ',dateRawData.shape[0], ' records') print('################################')

dateData.drop\_duplicates(subset='FinDate', keep='first', inplace=True) ################################################################ print('################')

sTableOut='Assess\_Date'

print('Storing :',sDatabaseNameOut,' Table:',sTableOut) dateData.to\_sql(sTableOut, connOut, if\_exists="replace") print('################') ################################################################ print('################################')

print('Store Rows : ',dateData.shape[0], ' records') print('################################') ################################################################ ################################################################

sTableIn='Retrieve\_Time'

sSQL='select \* FROM ' + sTableIn + ';' print('################')

sTableOut='Assess\_Time'

print('Loading :',sDatabaseNameIn,' Table:',sTableIn) timeRawData=pd.read\_sql\_query(sSQL, connIn) timeData=timeRawData

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

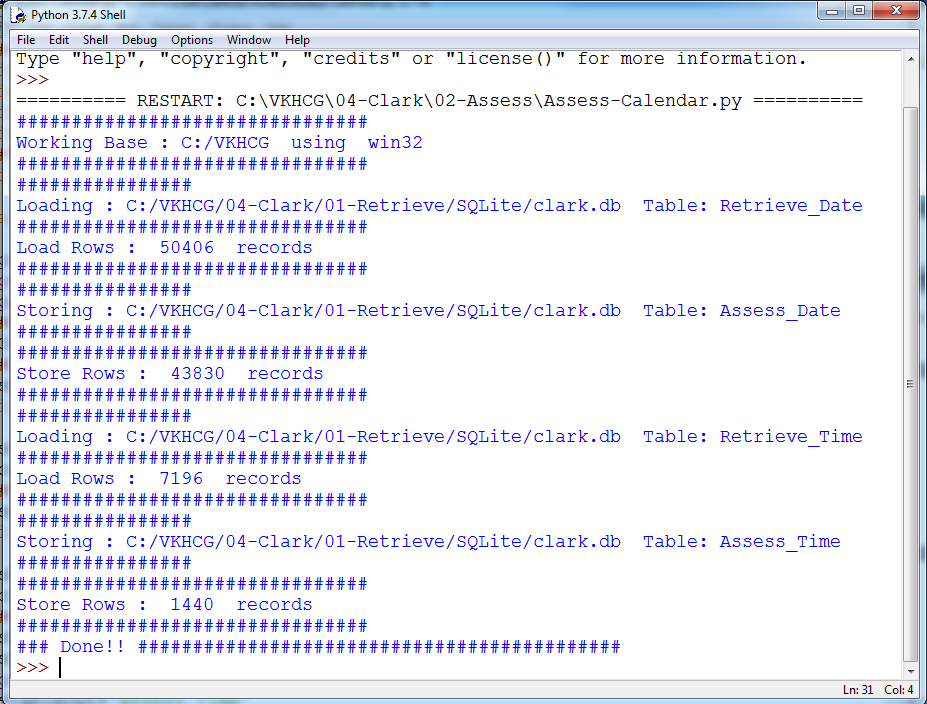
print('Load Rows : ',timeData.shape[0], ' records') print('################################')

timeData.drop\_duplicates(subset=None, keep='first', inplace=True) ################################################################ print('################')

sTableOut='Assess\_Time'

print('Storing :',sDatabaseNameOut,' Table:',sTableOut) timeData.to\_sql(sTableOut, connOut, if\_exists="replace") print('################') ################################################################ print('################################')

print('Store Rows : ',timeData.shape[0], ' records') print('################################') ################################################################ print('### Done!! ############################################') ################################################################



### Write a Python program to generate payroll from the given data.

People

Clark Ltd generates the payroll, so it holds all the staff records. Clark also handles all payments to suppliers and receives payments from customers’ details on all companies.

Open Python editor and create a file named Assess-People.py in directory C:\VKHCG\04-Clark\02-Assess.

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

import sys import os

import sqlite3 as sq import pandas as pd

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

Base='C:/VKHCG' print('################################')

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

Company='04-Clark'

sInputFileName1='01-Retrieve/01-EDS/02-Python/Retrieve-Data\_female-names.csv' sInputFileName2='01-Retrieve/01-EDS/02-Python/Retrieve-Data\_male-names.csv' sInputFileName3='01-Retrieve/01-EDS/02-Python/Retrieve-Data\_last-names.csv' sOutputFileName1='Assess-Staff.csv'

sOutputFileName2='Assess-Customers.csv' ################################################################

sDataBaseDir=Base + '/' + Company + '/02-Assess/SQLite' if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir) ################################################################

sDatabaseName=sDataBaseDir + '/clark.db' conn = sq.connect(sDatabaseName)

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

### Import Female Data ################################################################

sFileName=Base + '/' + Company + '/' + sInputFileName1 print('################################')

print('Loading :',sFileName) print('################################')

print(sFileName)

FemaleRawData=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1") FemaleRawData.rename(columns={'NameValues' : 'FirstName'},inplace=True) FemaleRawData.drop\_duplicates(subset=None, keep='first', inplace=True) FemaleData=FemaleRawData.sample(100)

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

sTable='Assess\_FemaleName'

print('Storing :',sDatabaseName,' Table:',sTable) FemaleData.to\_sql(sTable, conn, if\_exists="replace") print('################') print('################################')

print('Rows : ',FemaleData.shape[0], ' records') print('################################') ################################################################

### Import Male Data

sFileName=Base + '/' + Company + '/' + sInputFileName2 print('################################')

print('Loading :',sFileName) print('################################')

MaleRawData=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1") MaleRawData.rename(columns={'NameValues' : 'FirstName'},inplace=True) MaleRawData.drop\_duplicates(subset=None, keep='first', inplace=True) MaleData=MaleRawData.sample(100)

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

sTable='Assess\_MaleName'

print('Storing :',sDatabaseName,' Table:',sTable) MaleData.to\_sql(sTable, conn, if\_exists="replace") print('################')

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

print('Rows : ',MaleData.shape[0], ' records') print('################################') ################################################################

### Import Surname Data

sFileName=Base + '/' + Company + '/' + sInputFileName3 print('################################')

print('Loading :',sFileName) print('################################')

SurnameRawData=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1") SurnameRawData.rename(columns={'NameValues' : 'LastName'},inplace=True) SurnameRawData.drop\_duplicates(subset=None, keep='first', inplace=True) SurnameData=SurnameRawData.sample(200)

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

sTable='Assess\_Surname'

print('Storing :',sDatabaseName,' Table:',sTable) SurnameData.to\_sql(sTable, conn, if\_exists="replace") print('################') print('################################')

print('Rows : ',SurnameData.shape[0], ' records') print('################################') print('################')

sTable='Assess\_FemaleName & Assess\_MaleName' print('Loading :',sDatabaseName,' Table:',sTable) sSQL="select distinct"

sSQL=sSQL+ " A.FirstName," sSQL=sSQL+ " 'Female' as Gender" sSQL=sSQL+ " from"

sSQL=sSQL+ " Assess\_FemaleName as A" sSQL=sSQL+ " UNION"

sSQL=sSQL+ " select distinct" sSQL=sSQL+ " A.FirstName," sSQL=sSQL+ " 'Male' as Gender" sSQL=sSQL+ " from"

sSQL=sSQL+ " Assess\_MaleName as A;" FirstNameData=pd.read\_sql\_query(sSQL, conn) print('################')

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

sTable='Assess\_FirstName'

print('Storing :',sDatabaseName,' Table:',sTable) FirstNameData.to\_sql(sTable, conn, if\_exists="replace") print('################')

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

sTable='Assess\_FirstName x2 & Assess\_Surname' print('Loading :',sDatabaseName,' Table:',sTable) sSQL="select distinct"

sSQL=sSQL+ " A.FirstName,"

sSQL=sSQL+ " B.FirstName AS SecondName," sSQL=sSQL+ " C.LastName,"

sSQL=sSQL+ " A.Gender" sSQL=sSQL+ " from"

sSQL=sSQL+ " Assess\_FirstName as A" sSQL=sSQL+ " ,"

sSQL=sSQL+ " Assess\_FirstName as B" sSQL=sSQL+ " ,"

sSQL=sSQL+ " Assess\_Surname as C" sSQL=sSQL+ " WHERE"

sSQL=sSQL+ " A.Gender = B.Gender" sSQL=sSQL+ " AND"

sSQL=sSQL+ " A.FirstName <> B.FirstName;" PeopleRawData=pd.read\_sql\_query(sSQL, conn)

People1Data=PeopleRawData.sample(10000)

sTable='Assess\_FirstName & Assess\_Surname' print('Loading :',sDatabaseName,' Table:',sTable) sSQL="select distinct"

sSQL=sSQL+ " A.FirstName," sSQL=sSQL+ " '' AS SecondName," sSQL=sSQL+ " B.LastName," sSQL=sSQL+ " A.Gender" sSQL=sSQL+ " from"

sSQL=sSQL+ " Assess\_FirstName as A" sSQL=sSQL+ " ,"

sSQL=sSQL+ " Assess\_Surname as B;" PeopleRawData=pd.read\_sql\_query(sSQL, conn) People2Data=PeopleRawData.sample(10000) PeopleData=People1Data.append(People2Data) print(PeopleData)

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

sTable='Assess\_People'

print('Storing :',sDatabaseName,' Table:',sTable) PeopleData.to\_sql(sTable, conn, if\_exists="replace") print('################')

sFileDir=Base + '/' + Company + '/02-Assess/01-EDS/02-Python' if not os.path.exists(sFileDir):

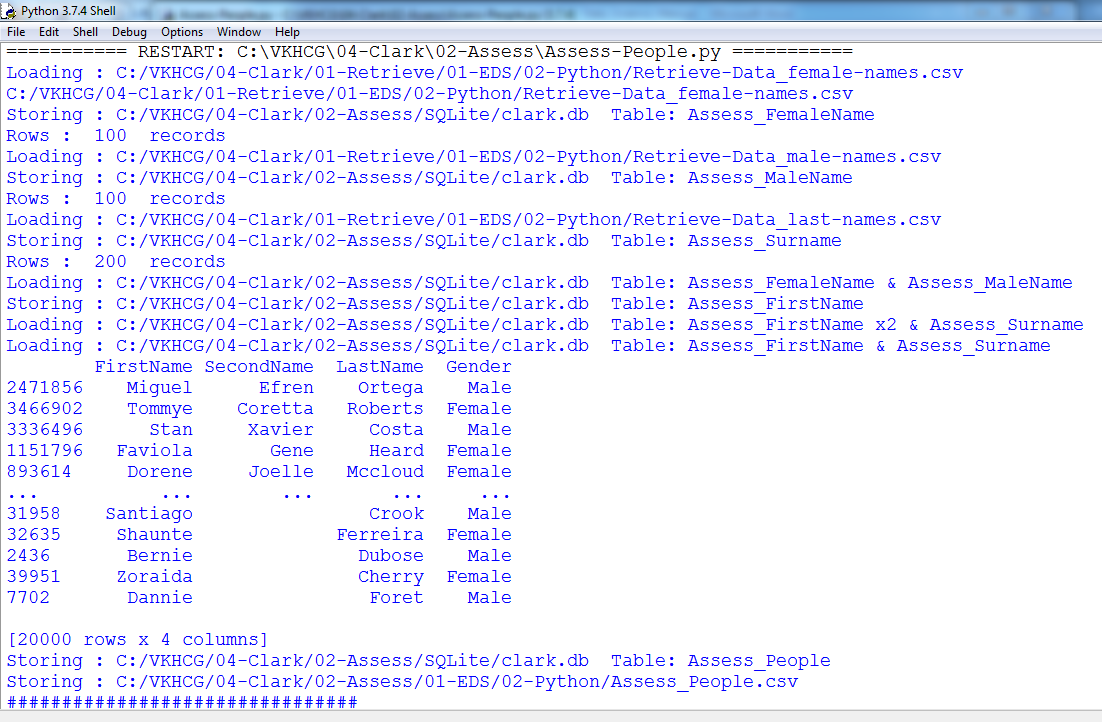
os.makedirs(sFileDir) sOutputFileName = sTable+'.csv'

sFileName=sFileDir + '/' + sOutputFileName print('################################')

print('Storing :', sFileName) print('################################')

PeopleData.to\_csv(sFileName, index = False) print('################################') ################################################################ print('### Done!! ############################################') ################################################################

### OUTPUT:



**Practical 6:**

**Processing Data**

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

Open your Python editor and create a file named Process\_Time.py. Save it into directory C:\VKHCG\01-Vermeulen\03-Process.

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

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

You have built your first hub and satellites for time in the data vault.

The data vault has been built in directory ..\ VKHCG\88-DV\datavault.db. You can access it with your SQLite tools

Golden Nominal

A golden nominal record is a single person’s record, with distinctive references for use by all systems. This gives the system a single view of the person. I use first name, other names, last name, and birth date as my golden nominal. The data we have in the assess directory requires a birth date to become a golden nominal. The proram will generate a golden nominal using our sample data set.

Open your Python editor and create a file called Process-People.py in the ..

C:\VKHCG\04-Clark\03-Process directory. ################################################################

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

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

### Output :

It will apply golden nominal rules by assuming nobody born before January 1, 1900, droping to two ISO complex date time structures, as the code does not translate into SQLite’s data types and saves your new golden nominal to a CSV file.

### Load the person into the data vault

========== RESTART: C:\VKHCG\04-Clark\03-Process\Process-People.py ========== Working Base : C:/VKHCG using win32

Loading : C:/VKHCG/04-Clark/02-Assess/01-EDS/02-Python/Assess\_People.csv C:/VKHCG/04-Clark/02-Assess/01-EDS/02-Python/Assess\_People.csv

Storing : C:/VKHCG/88-DV/datavault.db Table: Process\_Person

Storing : C:/VKHCG/88-DV/datavault.db Table: Satellite-Person-Gender Storing : C:/VKHCG/88-DV/datavault.db Table: Satellite-Person-Names

Storing : C:/VKHCG/04-Clark/03-Process/01-EDS/02-Python/Satellite-Person-Names.csv Vacuum Databases

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

### Done!! ############################################

Vehicles

The international classification of vehicles is a complex process. There are standards, but these are not universally applied or similar between groups or countries.

Let’s load the vehicle data for Hillman Ltd into the data vault, as we will need it later. Create a new file named Process-Vehicle-Logistics.py in the Python editor in directory ..\VKHCG\03-Hillman\03-Process.

# -\*- coding: utf-8 -\*- 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)

### Vehicle Hub #

VehicleHub=VehicleKey[['ObjectType','ObjectKey','ObjectUUID']].copy() VehicleHub.index.name='ObjectHubID'

sTable = 'Hub-Object-Vehicle'

print('Storing :',sDatabaseName,' Table:',sTable) VehicleHub.to\_sql(sTable, conn2, if\_exists="replace") ### Vehicle Satellite

#

VehicleSatellite=VehicleKey[['ObjectType','ObjectKey','ObjectUUID','Make','Model']].copy() VehicleSatellite.index.name='ObjectSatelliteID'

sTable = 'Satellite-Object-Make-Model' print('Storing :',sDatabaseName,' Table:',sTable)

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

### Vehicle Dimension sView='Dim-Object'

print('Storing :',sDatabaseName,' View:',sView) sSQL="CREATE VIEW IF NOT EXISTS [" + sView + "] AS" sSQL=sSQL+ " SELECT DISTINCT"

sSQL=sSQL+ " H.ObjectType,"

sSQL=sSQL+ " H.ObjectKey AS VehicleKey," sSQL=sSQL+ " TRIM(S.Make) AS VehicleMake," sSQL=sSQL+ " TRIM(S.Model) AS VehicleModel" sSQL=sSQL+ " FROM"

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

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

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

sSQL=sSQL+ " H.ObjectUUID=S.ObjectUUID;" sql.execute(sSQL,conn2)

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

print('Loading :',sDatabaseName,' Table:',sView) sSQL=" SELECT DISTINCT"

sSQL=sSQL+ " VehicleMake," sSQL=sSQL+ " VehicleModel" sSQL=sSQL+ " FROM" sSQL=sSQL+ " [" + sView + "]"

sSQL=sSQL+ " ORDER BY"

sSQL=sSQL+ " VehicleMake" sSQL=sSQL+ " AND"

sSQL=sSQL+ " VehicleMake;" DimObjectData=pd.read\_sql\_query(sSQL, conn2)

DimObjectData.index.name='ObjectDimID' DimObjectData.sort\_values(['VehicleMake','VehicleModel'],inplace=True, ascending=True) print('################')

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

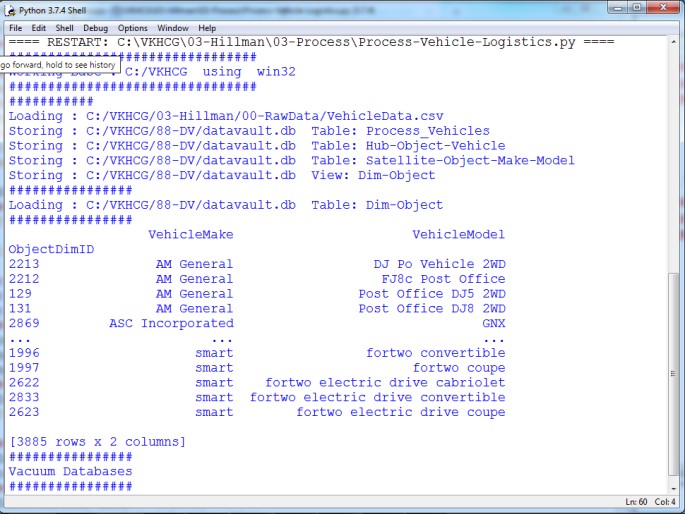
print('Vacuum Databases') sSQL="VACUUM;"

sql.execute(sSQL,conn1) sql.execute(sSQL,conn2) print('################')

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

conn1.close() conn2.close()

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



## Human-Environment Interaction

In the Python editor, open a new file named Process\_Location.py in directory ..\VKHCG\01- Vermeulen\03-Process.

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'

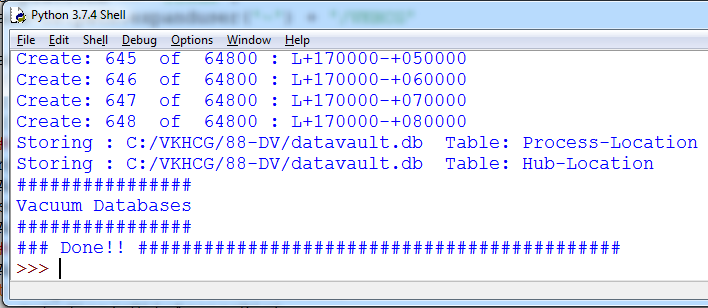
print('Storing :',sDatabaseName,' Table:',sTable) LocationHubIndex.to\_sql(sTable, conn1, if\_exists="replace") sTable = 'Hub-Location'

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

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



Forecasting

Forecasting is the ability to project a possible future, by looking at historical data. The datavault enables these types of investigations, owing to the complete history it collects as itprocesses the source’s systems data. A data scientist supply answers to such questions as the following:

* + What should we buy?
  + What should we sell?
  + Where will our next business come from?

People want to know what you calculate to determine what is about to happen.

Open a new file in your Python editor and save it as Process-Shares-Data.py in directory C: \VKHCG\04-Clark\03-Process. I will guide you through this

process. You will require a library called quandl

### type pip install quandl in cmd

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

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)

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

### Import Share Names Data ################################################################

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

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

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

### Output:

======== RESTART: C:\VKHCG\04-Clark\03-Process\Process-Shares-Data.py ======== Working Base : C:/VKHCG using win32

Loading : C:/VKHCG/04-Clark/00-RawData/VKHCG\_Shares.csv Rows : 10

Columns: 3

Storing : C:/VKHCG/04-Clark/01-Retrieve/01-EDS/02-Python/Retrieve\_Shares.csv Storing : C:/VKHCG/04-Clark/02-Assess/01-EDS/02-Python/Assess\_Shares.csv Storing : C:/VKHCG/04-Clark/03-Process/01-EDS/02-Python/Process\_Shares.csv Share : WIKI/GOOGL

Rows : 3424

Columns: 14

Storing : C:/VKHCG/04-Clark/03-Process/SQLite/clark.db Table: WIKI\_Google

Storing : C:/VKHCG/04-Clark/01-Retrieve/01-EDS/02-Python/Retrieve\_WIKI\_Google.csv Storing : C:/VKHCG/04-Clark/02-Assess/01-EDS/02-Python/Assess\_WIKI\_Google.

## Transform Superstep

**Practical 7:**

**Transforming Data**

C: \VKHCG\01-Vermeulen\04-Transform.

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 :** Guðmundur Gunnarsson was born on December 20, 1960, at 9:15 in Landspítali,Hringbraut 101, 101 Reykjavík, Iceland.

You must build three items: **dimension Person**, **dimension Time**, and **factPersonBornAtTime**. Open your Python editor and create a file named Transform-Gunnarsson-Sun-Model.py in directory C:\VKHCG\01-Vermeulen\04-Transform.

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

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)

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

sDataWarehousetDir=Base + '/99-DW'

if not os.path.exists(sDataWarehousetDir): os.makedirs(sDataWarehousetDir)

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

sDatabaseName=sDataWarehousetDir + '/datawarehouse.db' conn2 = sq.connect(sDatabaseName)

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

print('Time Dimension') BirthZone = 'Atlantic/Reykjavik'

BirthDateUTC = datetime(1960,12,20,10,15,0) BirthDateZoneUTC=BirthDateUTC.replace(tzinfo=timezone('UTC')) BirthDateZoneStr=BirthDateZoneUTC.strftime("%Y-%m-%d %H:%M:%S") BirthDateZoneUTCStr=BirthDateZoneUTC.strftime("%Y-%m-%d %H:%M:%S (%Z) (%z)") BirthDate = BirthDateZoneUTC.astimezone(timezone(BirthZone)) BirthDateStr=BirthDate.strftime("%Y-%m-%d %H:%M:%S (%Z) (%z)") BirthDateLocal=BirthDate.strftime("%Y-%m-%d %H:%M:%S") ################################################################

IDTimeNumber=str(uuid.uuid4()) TimeLine=[('TimeID', [IDTimeNumber]),

('UTCDate', [BirthDateZoneStr]), ('LocalTime', [BirthDateLocal]), ('TimeZone', [BirthZone])]

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

DimTime=TimeFrame DimTimeIndex=DimTime.set\_index(['TimeID'],inplace=False) ################################################################

sTable = 'Dim-Time' print('\n#################################')

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

DimTimeIndex.to\_sql(sTable, conn1, if\_exists="replace") DimTimeIndex.to\_sql(sTable, conn2, if\_exists="replace") ################################################################ print('\n#################################')

print('Dimension Person') print('\n#################################')

FirstName = 'Guðmundur' LastName = 'Gunnarsson'

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

IDPersonNumber=str(uuid.uuid4()) PersonLine=[('PersonID', [IDPersonNumber]),

('FirstName', [FirstName]),

('LastName', [LastName]),

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

('DateTimeValue', [BirthDateZoneStr])] PersonFrame = pd.DataFrame.from\_items(PersonLine)

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

DimPerson=PersonFrame DimPersonIndex=DimPerson.set\_index(['PersonID'],inplace=False) ################################################################

sTable = 'Dim-Person' print('\n#################################')

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

DimPersonIndex.to\_sql(sTable, conn1, if\_exists="replace") DimPersonIndex.to\_sql(sTable, conn2, if\_exists="replace") ################################################################ print('\n#################################')

print('Fact - Person - time') print('\n#################################')

IDFactNumber=str(uuid.uuid4())

PersonTimeLine=[('IDNumber', [IDFactNumber]), ('IDPersonNumber', [IDPersonNumber]), ('IDTimeNumber', [IDTimeNumber])]

PersonTimeFrame = pd.DataFrame.from\_items(PersonTimeLine) ################################################################

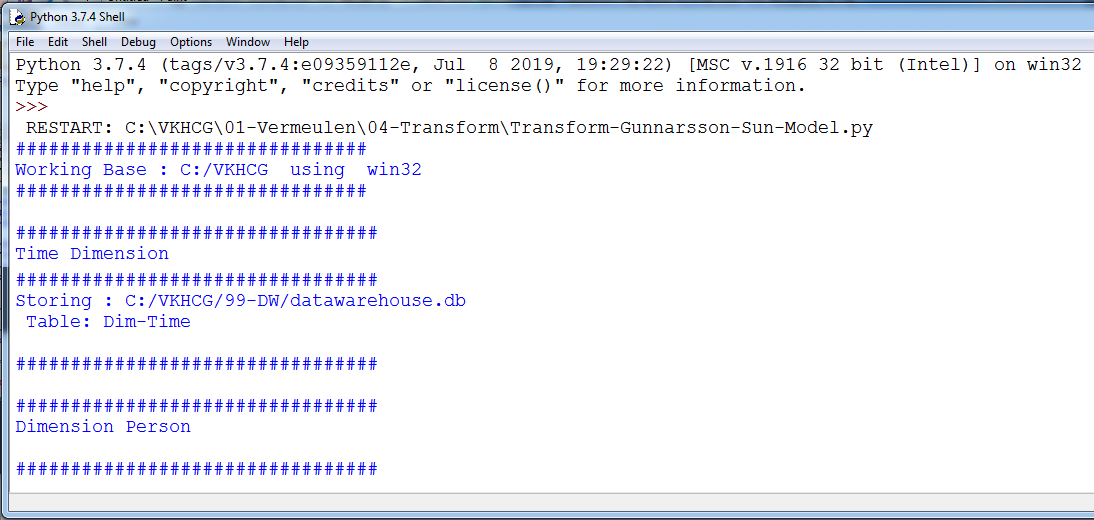
FctPersonTime=PersonTimeFrame FctPersonTimeIndex=FctPersonTime.set\_index(['IDNumber'],inplace=False) ################################################################

sTable = 'Fact-Person-Time' print('\n#################################')

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

FctPersonTimeIndex.to\_sql(sTable, conn1, if\_exists="replace") FctPersonTimeIndex.to\_sql(sTable, conn2, if\_exists="replace") ################################################################

### Output:



Building a Data Warehouse

Open the Transform-Sun-Models.py file from directory C:\VKHCG\01-Vermeulen\04- Transform. ################################################################

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

Linear regression is used if there is a relationship or significant association between the variables. This can be checked by scatterplots. If no linear association appears between the variables, fitting a linear regression model to the data will not provide a useful model. A linear regression line has equations in the following form:

Y = a + bX,

Where, X = explanatory variable and

Y = dependent variable b = slope of the line

a = intercept (the value of y when x = 0)

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

# -\*- coding: utf-8 -\*- ################################################################

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()

# Load the diabetes dataset diabetes = datasets.load\_diabetes()

# Use only one feature

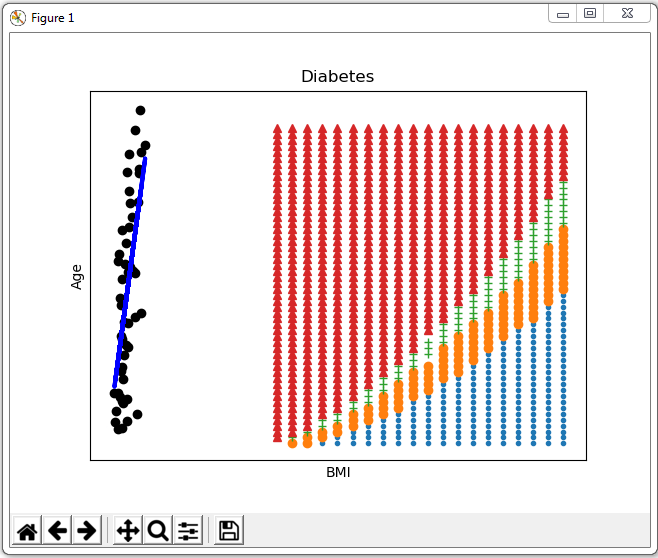
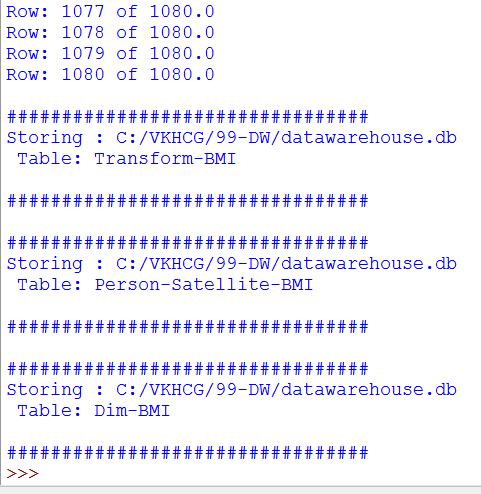
diabetes\_X = diabetes.data[:, np.newaxis, 2] diabetes\_X\_train = diabetes\_X[:-30] diabetes\_X\_test = diabetes\_X[-50:] diabetes\_y\_train = diabetes.target[:-30] diabetes\_y\_test = diabetes.target[-50:]

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 8:

**Organizing Data C:\VKHCG\01-Vermeulen\05-Organise\ Organize-Horizontal.py** ################################################################

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 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#################################')

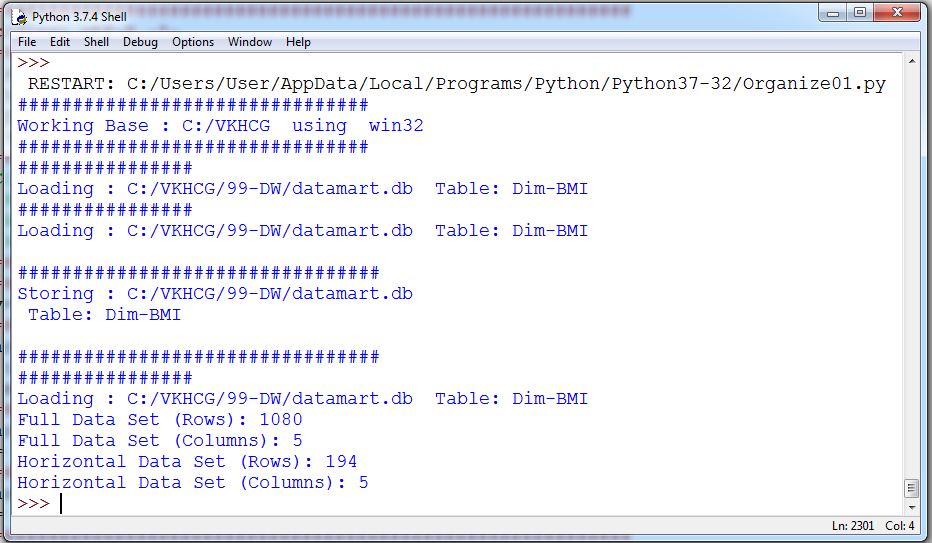
#DimPersonIndex.to\_sql(sTable, conn2, if\_exists="replace") ################################################################ print('################')

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

### C:\VKHCG\01-Vermeulen\05-Organise\ Organize-Vertical.py

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];"

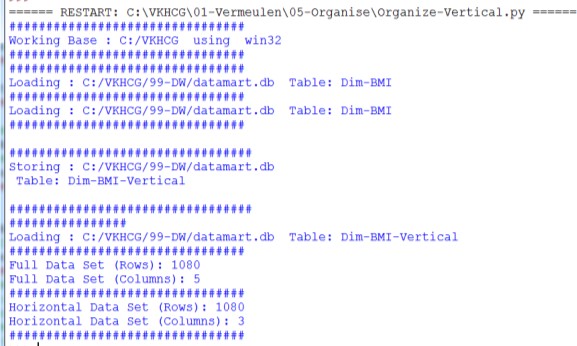
PersonFrame2=pd.read\_sql\_query(sSQL, conn2) ################################################################ print('################################')

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

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

**Output:**



Island Style

**C:\VKHCG\01-Vermeulen\05-Organise\ Organize-Island.py**

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\

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];"

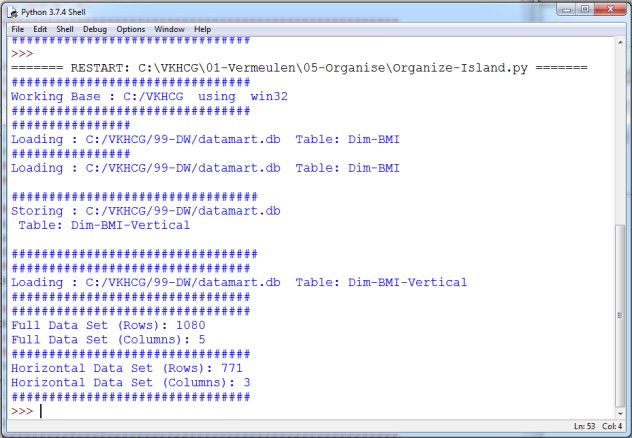
PersonFrame2=pd.read\_sql\_query(sSQL, conn2) ################################################################ print('################################')

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

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

### Output:



Secure Vault Style

**C:\VKHCG\01-Vermeulen\05-Organise\ Organize-Secure-Vault.py**

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

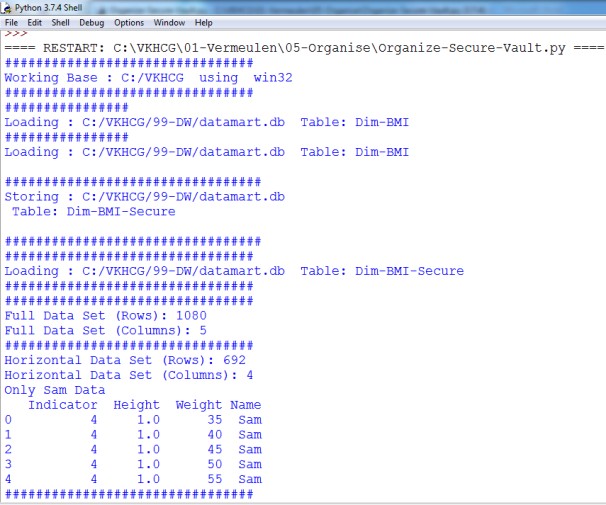
print('Full Data Set (Rows):', PersonFrame0.shape[0]) print('Full Data Set (Columns):', PersonFrame0.shape[1]) print('################################')

print('Horizontal Data Set (Rows):', PersonFrame2.shape[0]) print('Horizontal Data Set (Columns):', PersonFrame2.shape[1]) print('Only Sam Data')

print(PersonFrame2.head()) print('################################')

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

### Output:



Association Rule Mining

**C:\VKHCG\01-Vermeulen\05-Organise\ Organize-Association-Rule.py**

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'))

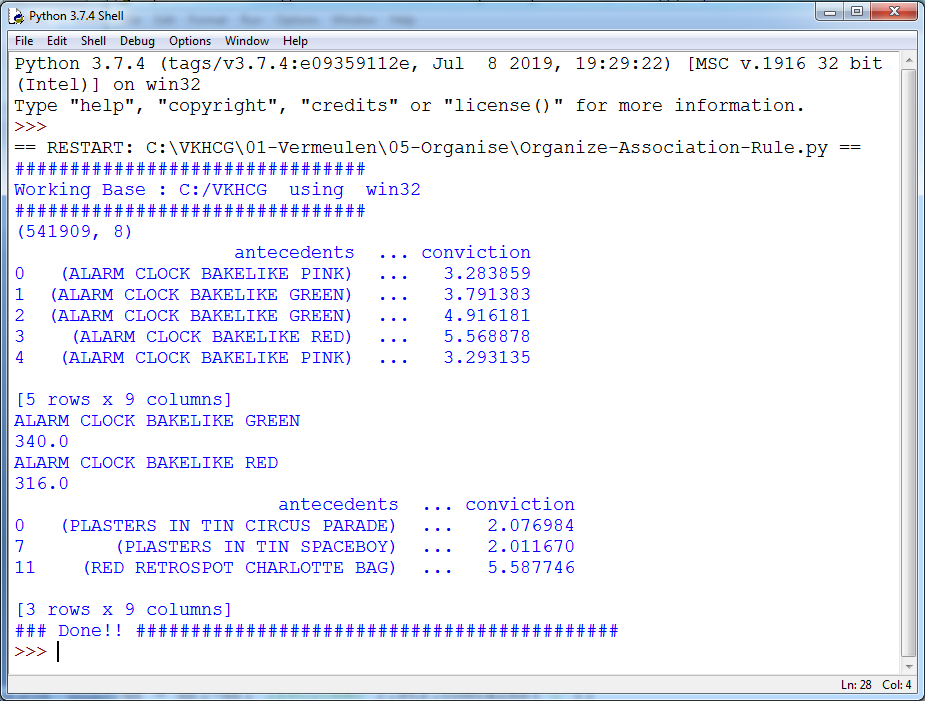
basket\_sets2 = basket2.applymap(encode\_units) basket\_sets2.drop('POSTAGE', inplace=True, axis=1)

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)])

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

### Output:



**Create a Network Routing Diagram**

I will guide you through a possible solution for the requirement, by constructing an island-style Organize superstep that uses a graph data model to reduce the records and the columns on the data set.

**C:\VKHCG\01-Vermeulen\05-Organise\ Organise-Network-Routing-Company.py**

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

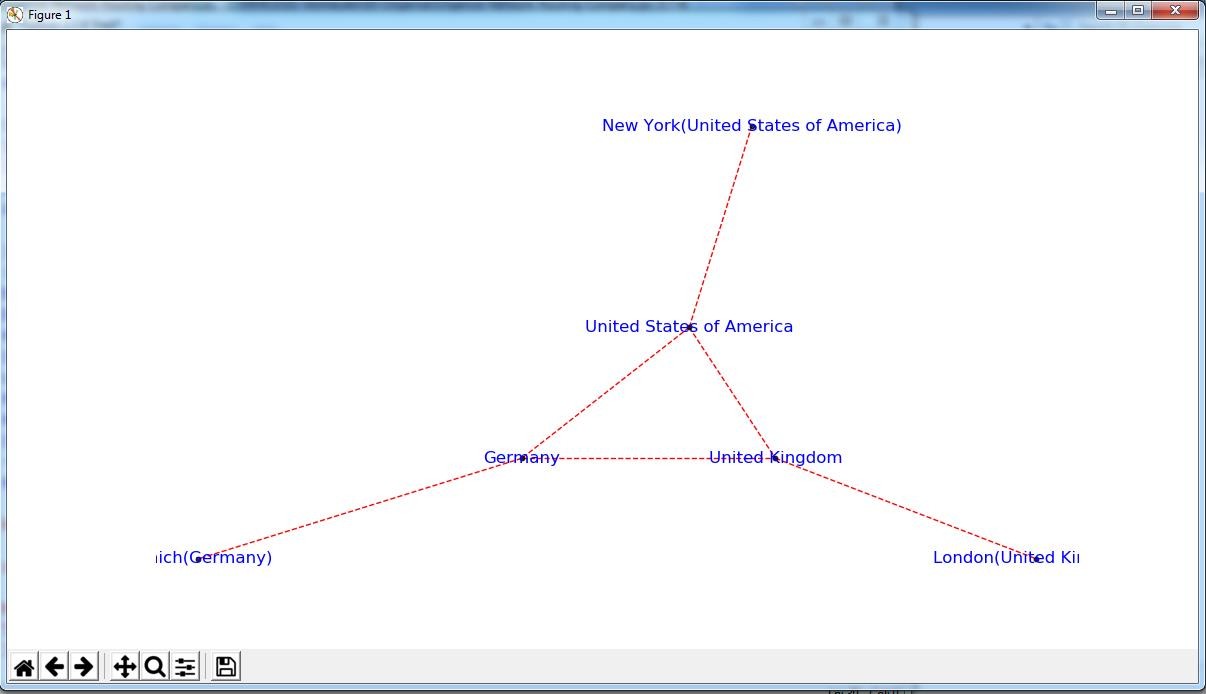
print('Storing Graph Image:',sFileName) print('################################')

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



## Picking Content for Billboards

### C:\VKHCG\02-Krennwallner\05-Organise\ Organise-billboards.py

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

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)

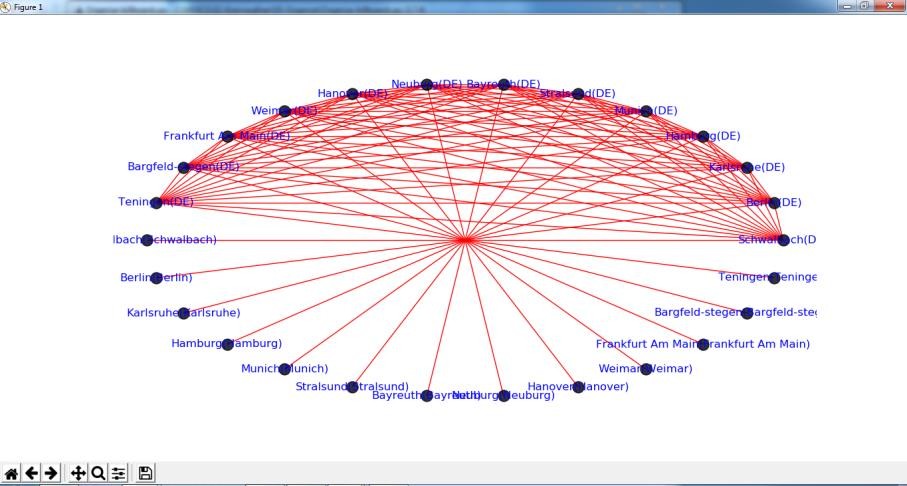
nx.draw\_networkx\_nodes(G,pos, node\_color='k', node\_size=150, alpha=0.8) nx.draw\_networkx\_edges(G, pos,edge\_color='r', arrows=False, style='solid') nx.draw\_networkx\_labels(G,pos,font\_size=12,font\_family='sans-serif',font\_color='b') plt.axis('off')

plt.savefig(sFileName,dpi=600) plt.show()

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

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

### Output :



**Create a Delivery Route**

**C:\VKHCG\03-Hillman\05-Organise\Organise-Routes.py**

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' ################################################################ ################################################################

### Import Routes Data ################################################################

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

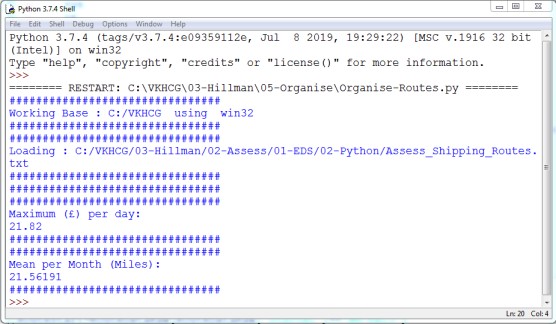
print('Maximum (£) per day:') print(RouteMaxCost) print('################################')

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

RouteMean=RouteStart["Measure"].mean() RouteMeanMonth=round((((RouteMean/1000)\*2\*30)),6) print('################################')

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

### Output:



Clark Ltd

Our financial services company has been tasked to investigate the options to convert1 million pounds sterling into extra income. Mr. Clark Junior suggests using the simplevariance in the daily rate between the British pound sterling and the US dollar, togenerate extra income from trading. Your chief financial officer wants to know if this isfeasible?

**Simple Forex Trading Planner**

Your challenge is to take 1 million US dollars or just over six hunderd thou sand pounds sterling and, by simply converting it between pounds sterling and US dollars, achieve a profit. Are you up to this challenge?

The Program will help you how to model this problem and achieve a positive outcome. The forex data has been collected on a daily basis by Clark’s accounting department, from previous overseas transactions.

**C:\VKHCG\04-Clark\05-Organise\Organise-Forex.py**

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.

Report Superstep

**Practical 9**

**Generating Data**

The Report superstep is the step in the ecosystem that enhances the data science findings with the art of storytelling and data visualization. You can perform the best data science, but if you cannot execute a respectable and trustworthy Report step by turning your data science into actionable business insights, you have achieved no advantage for your business.

Vermeulen PLC

Vermeulen requires a map of all their customers’ data links. Can you provide a report to deliver this? I will guide you through an example that delivers this requirement.

**C:\VKHCG\01-Vermeulen\06-Report\Raport-Network-Routing-Customer.py**

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

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' ################################################################ ################################################################

### Import Country Data ################################################################

sFileName=Base + '/' + Company + '/' + sInputFileName print('################################')

print('Loading :',sFileName) print('################################')

CustomerDataRaw=pd.read\_csv(sFileName,header=0,low\_memory=False, encoding="latin-1") CustomerData=CustomerDataRaw.head(100)

print('Loaded Country:',CustomerData.columns.values) print('################################') ################################################################

print(CustomerData.head()) 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) ################################################################

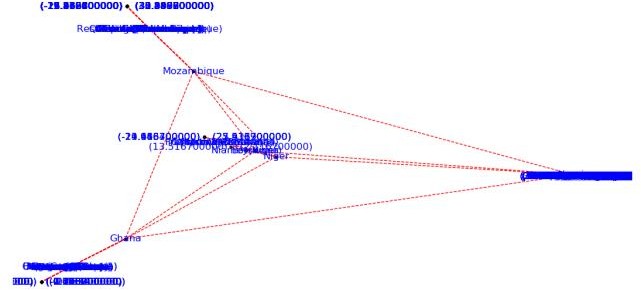
sFileName=Base + '/' + Company + '/' + sOutputFileName2 print('################################')

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

Krennwallner AG

The Krennwallner marketing department wants to deploy the locations of the billboards onto the company web server. Can you prepare three versions of the locations’ web pages?

* Locations clustered into bubbles when you zoom out
* Locations as pins
* Locations as heat map

Picking Content for Billboards

**C:\VKHCG\02-Krennwallner\06-Report\Report\_Billboard.py**

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)

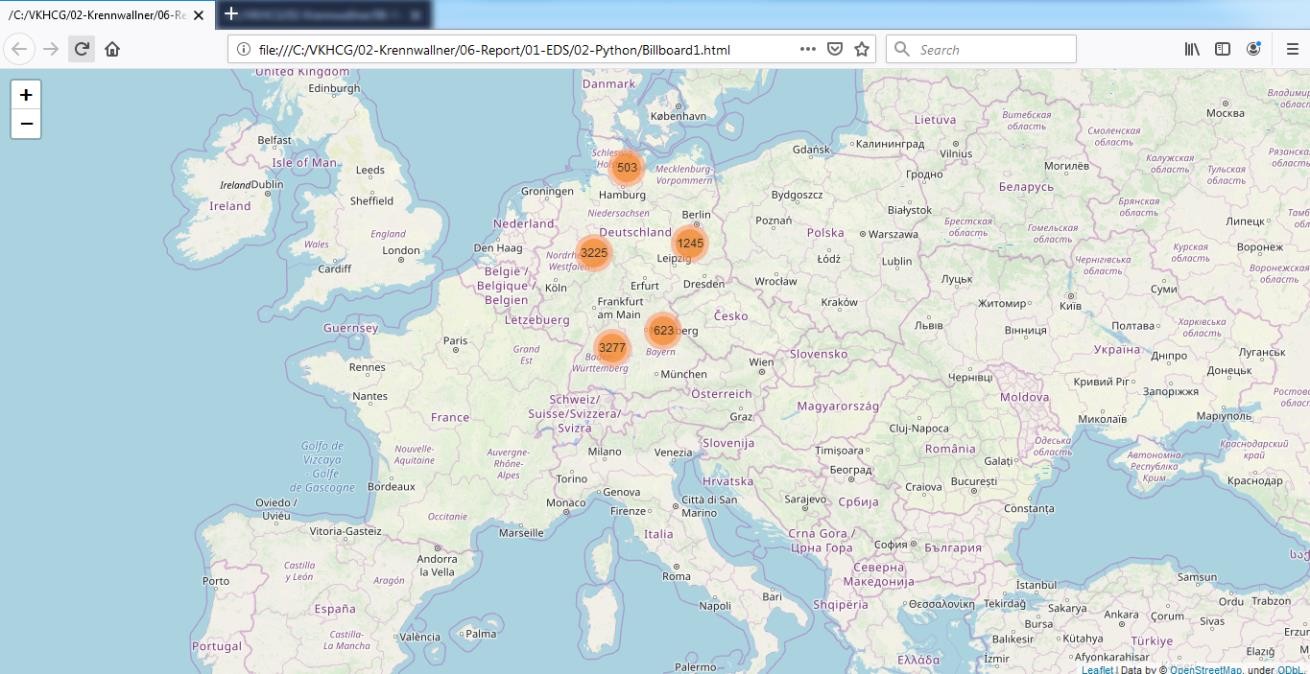
webbrowser.open('file://' + os.path.realpath(sFileNameHtml)) ################################################################

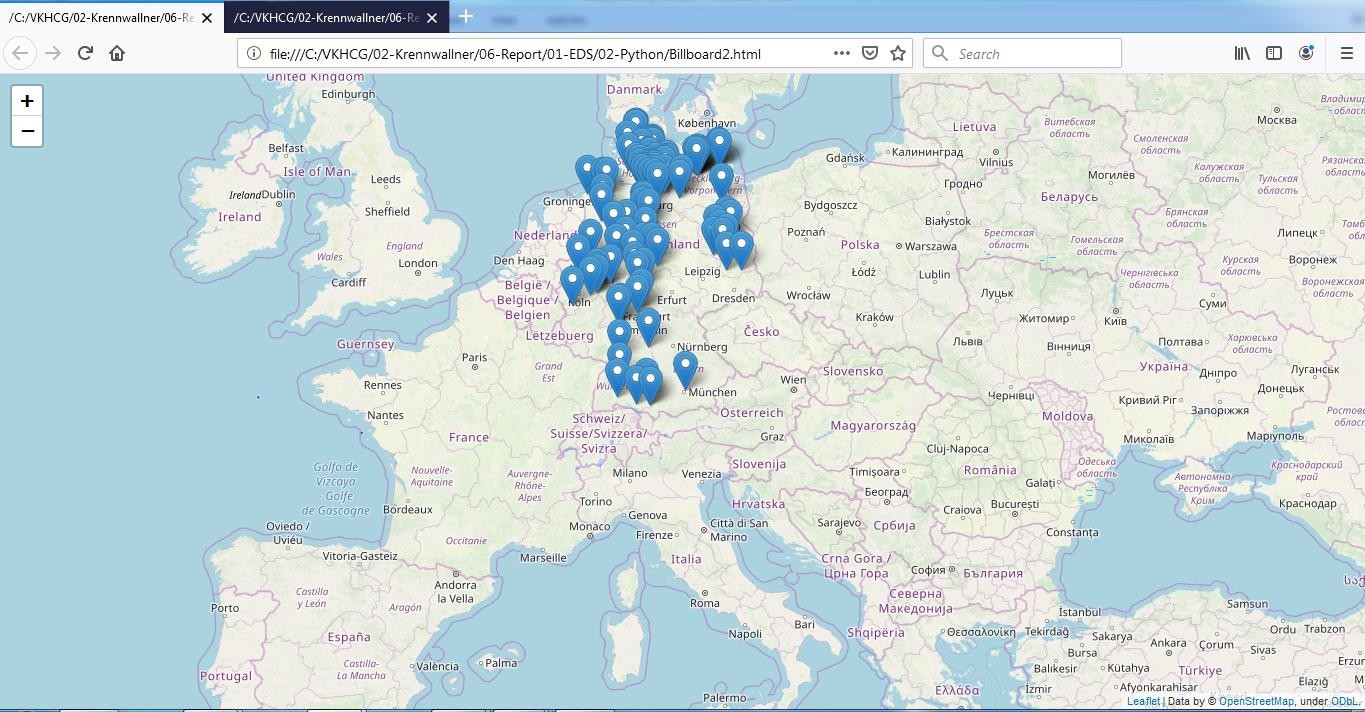
stops\_heatmap = Map(location=[48.1459806, 11.4985484], zoom\_start=5) stops\_heatmap.add\_child(HeatMap([[row["Latitude"], row["Longitude"]] for name, row in pins.iloc[:100].iterrows()]))

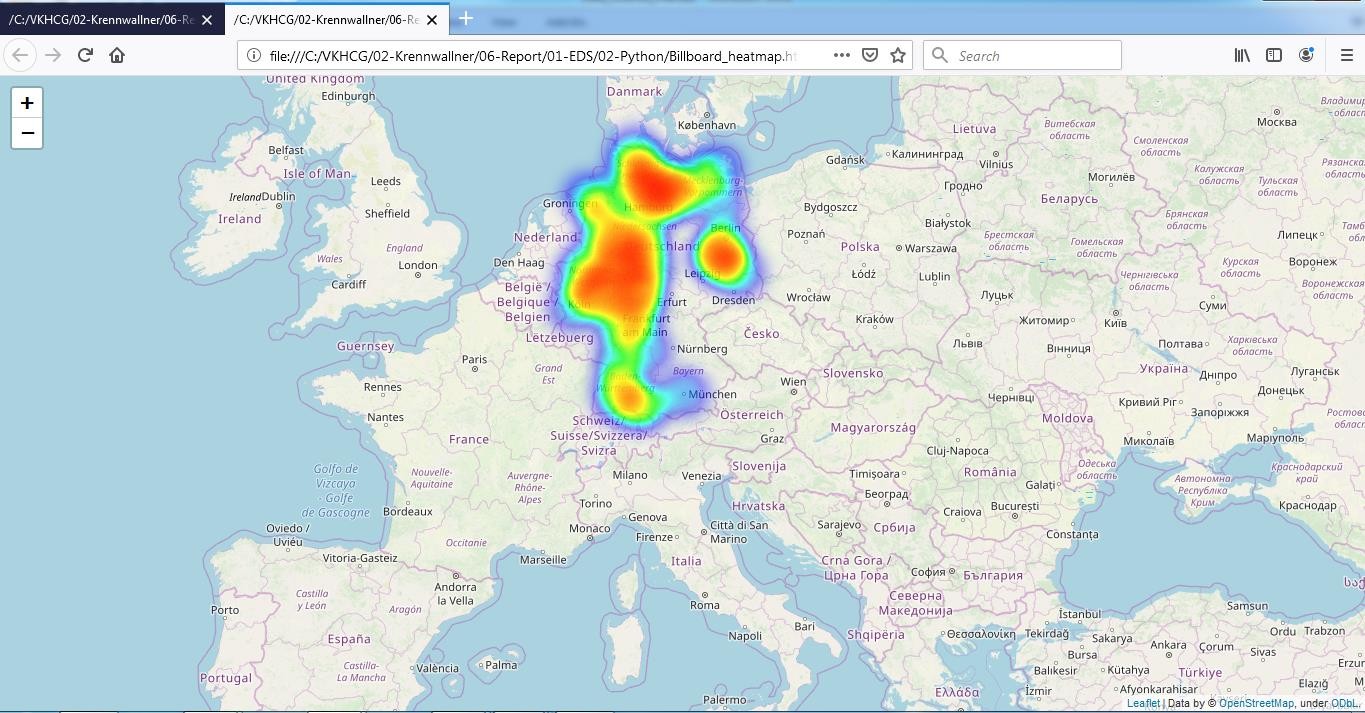
sFileNameHtml=Base+'/02-Krennwallner/06-Report/01-EDS/02-Python/Billboard\_heatmap.html' stops\_heatmap.save(sFileNameHtml)

webbrowser.open('file://' + os.path.realpath(sFileNameHtml)) ################################################################ print('### Done!! ############################################') ################################################################

## Output:







Hillman Ltd

Dr. Hillman Sr. has just installed a camera system that enables the company to capture video and, therefore, indirectly, images of all containers that enter or leave the warehouse. Can you convert the number on the side of the containers into digits?

Reading the Containers

**C:\VKHCG\03-Hillman\06-Report\ Report\_Reading\_Container.py**

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()

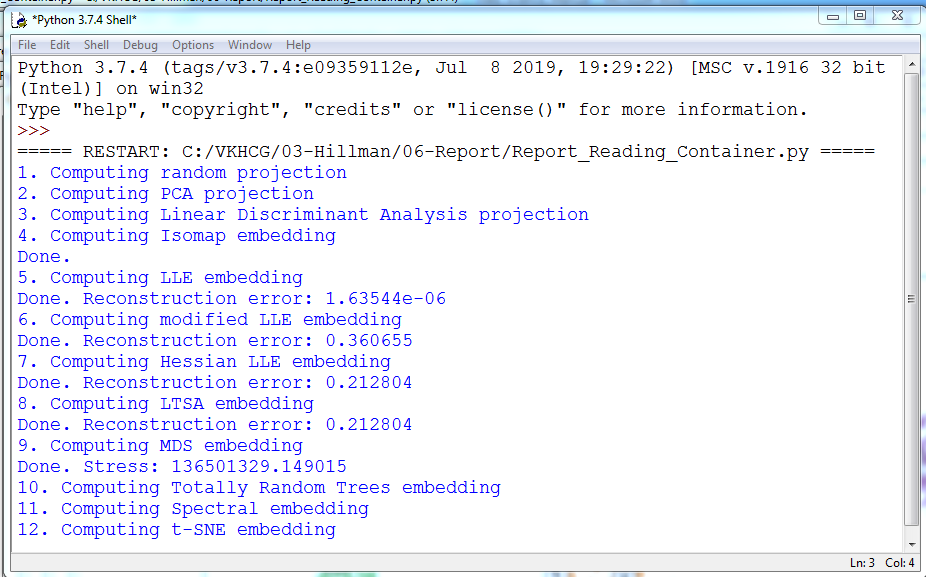
X\_se = embedder.fit\_transform(X)

plot\_embedding(X\_se,"Spectral embedding of the digits (time %.2fs)" %(time() - t0)) print("Computing t-SNE embedding")

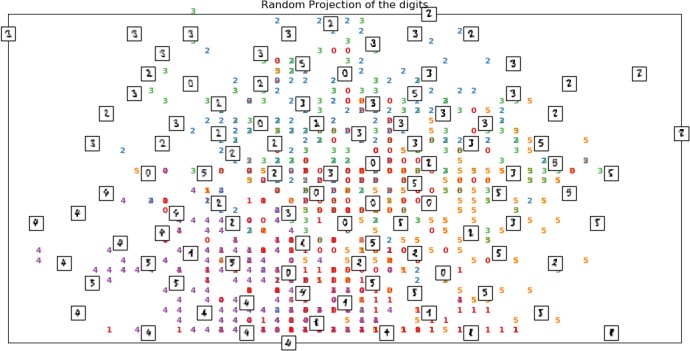
tsne = manifold.TSNE(n\_components=2, init='pca', random\_state=0) t0 = time()

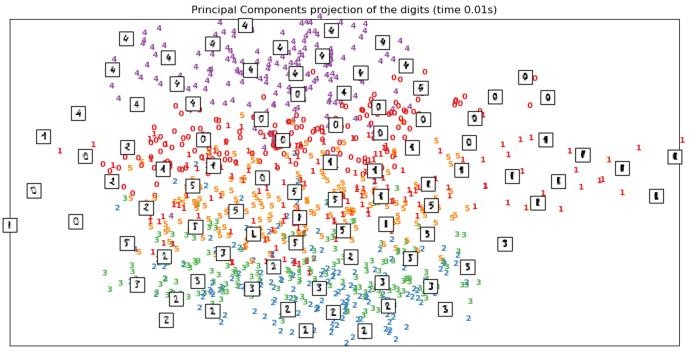
X\_tsne = tsne.fit\_transform(X)

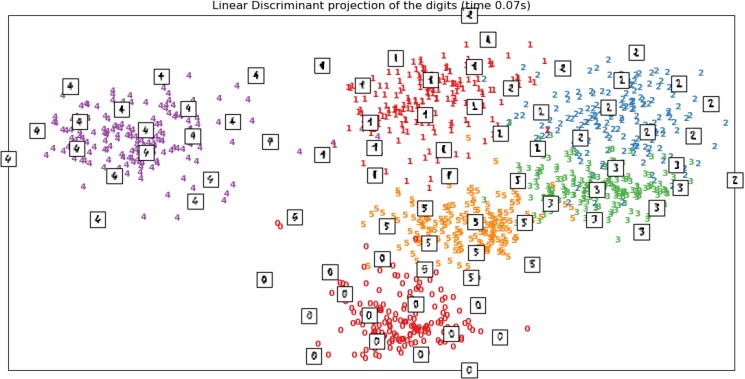
plot\_embedding(X\_tsne,"t-SNE embedding of the digits (time %.2fs)" %(time() - t0)) plt.show()

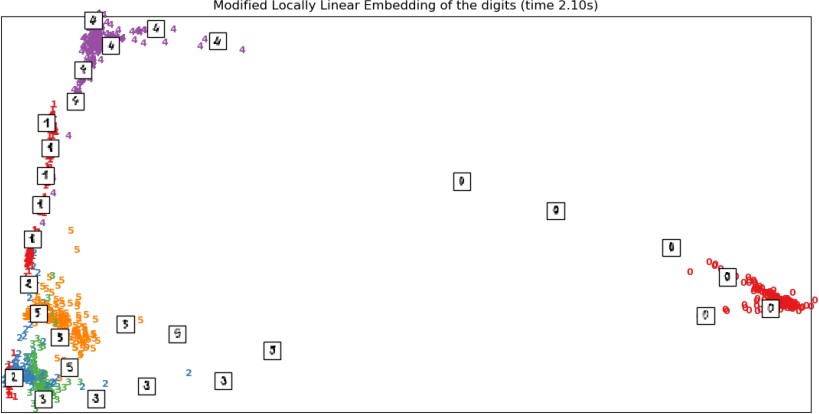
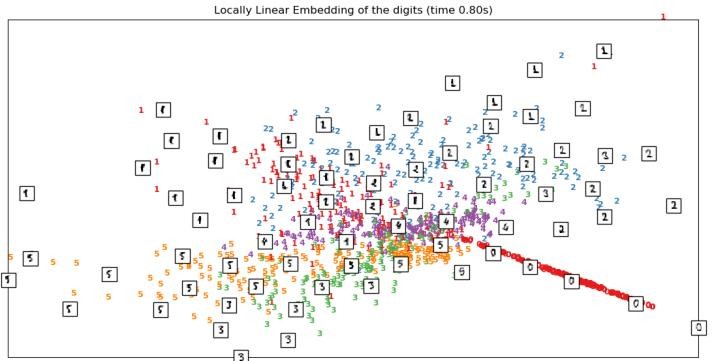


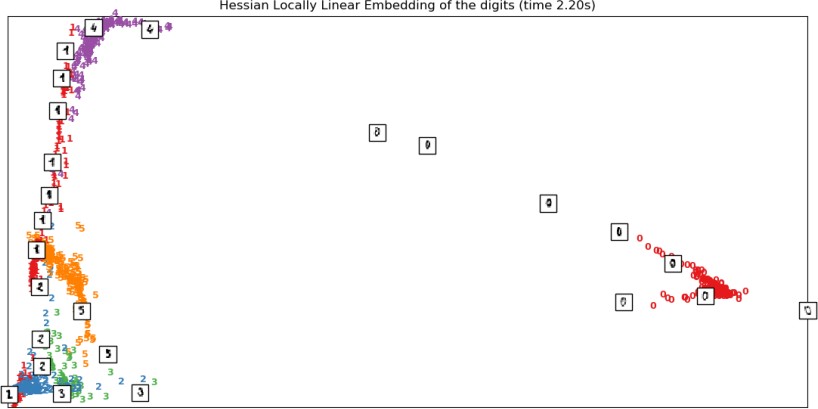


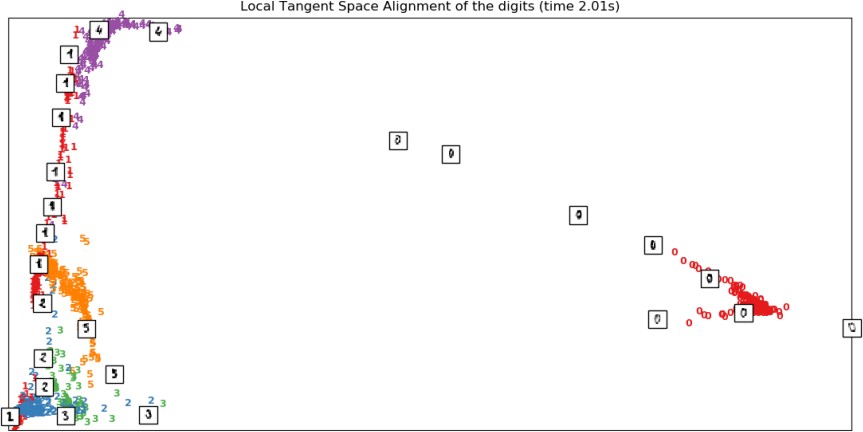












Clark Ltd

The financial company in VKHCG is the Clark accounting firm that VKHCG owns with a 60% stake. The accountants are the financial advisers to the group and handle everything to do with the complex work of international accounting.

Financials

The VKHCG companies did well last year, and the teams at Clark must prepare a balance sheet for each company in the group. The companies require a balance sheet for each company, to be produced using the template (Balance-Sheet-Template.xlsx) that can be found in the example directory (..\VKHCG\04-Clark\00- RawData).

The Program will guide you through a process that will enable you to merge the data science with preformatted Microsoft Excel template, to produce a balance sheet for each of the VKHCG companies.

## C:\VKHCG\04-Clark\06-Report\Report-Balance-Sheet.py

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:

Graphics

This section will now guide you through a number of visualizations that particularly useful in presenting data to my customers.

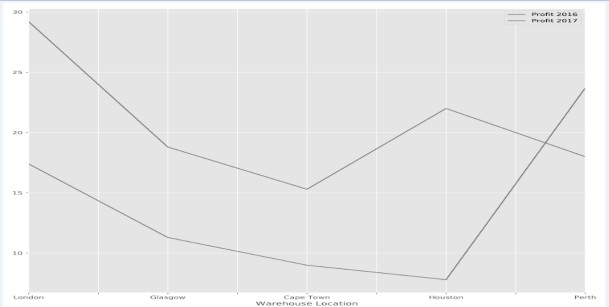
Pie Graph Double Pie

C:\VKHCG\01-Vermeulen\06-Report\Report\_Graph\_A.py



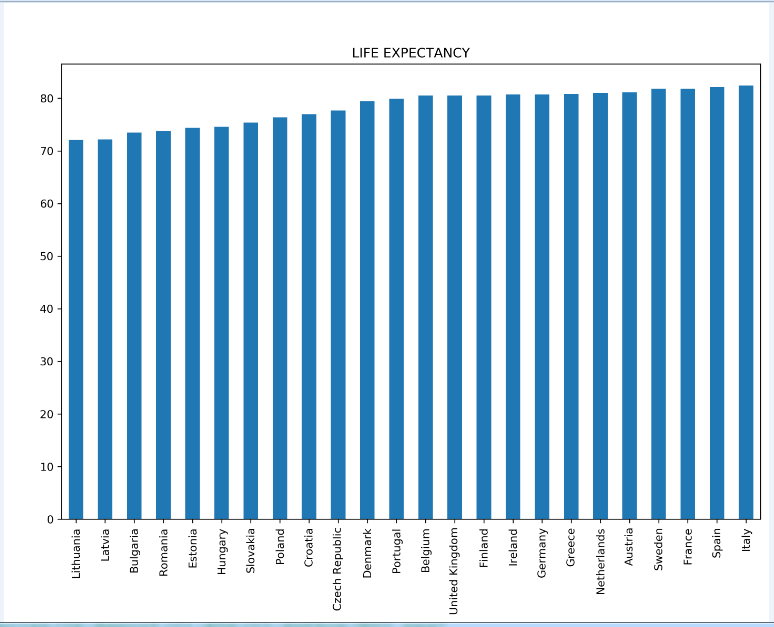
Line Graph

C:/VKHCG/01-Vermeulen/06-Report/Report\_Graph\_A.py



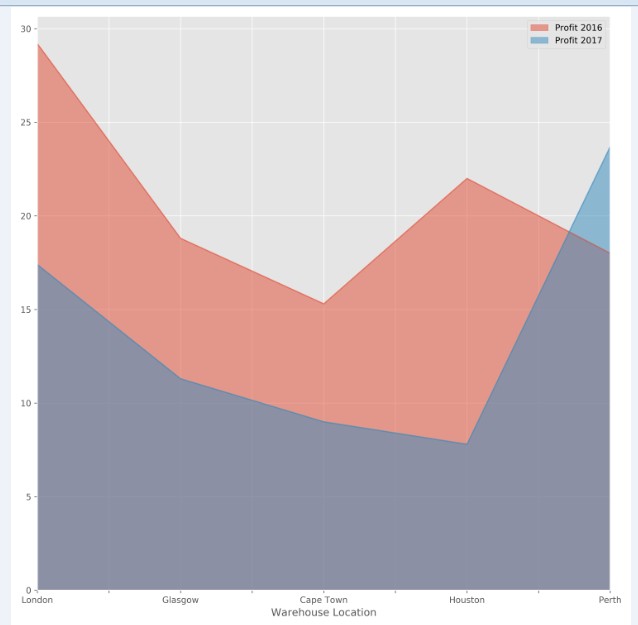
Bar Graph / Horizontal Bar Graph

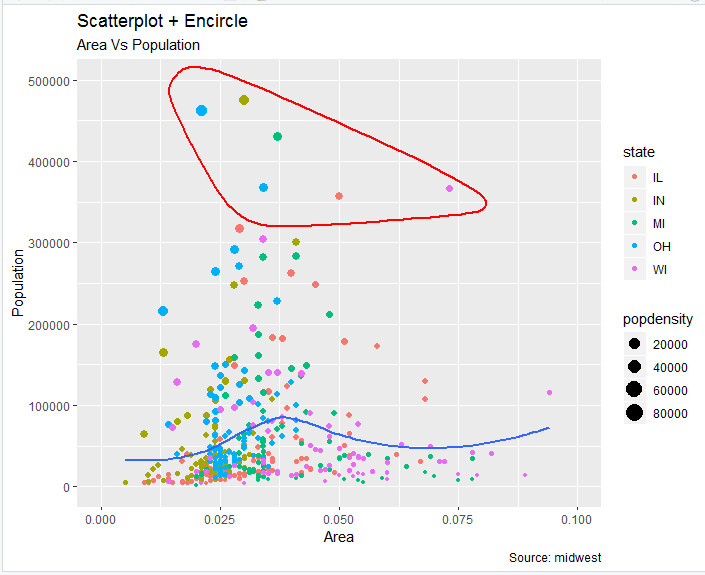
C:/VKHCG/01-Vermeulen/06-Report/Report\_Graph\_A.py



Area Graph

C:/VKHCG/01-Vermeulen/06-Report/Report\_Graph\_A.py





SCATTER GRAPH : [VKHCG](https://github.com/Apress/practical-data-science/tree/master/VKHCG)/[03-HILLMAN](https://github.com/Apress/practical-data-science/tree/master/VKHCG/03-Hillman)/[06-REPORT](https://github.com/Apress/practical-data-science/tree/master/VKHCG/03-Hillman/06-Report)/REPORT- SCATTERPLOT-WITH-ENCIRCLING.R

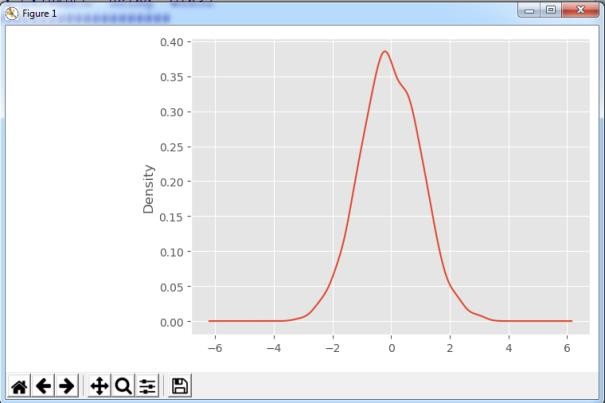
Hexbin:

Program : C:\VKHCG\01-Vermeulen\06-Report\Report\_Graph\_A.py



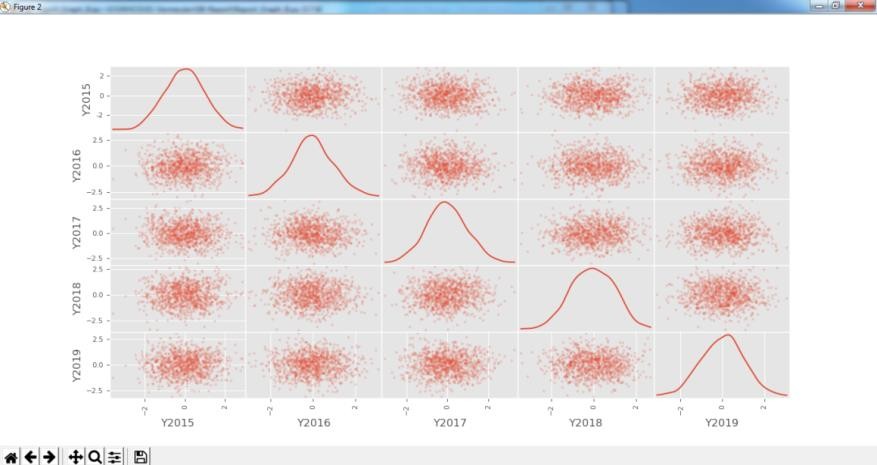
Kernel Density Estimation (KDE) Graph

C:\VKHCG\01-Vermeulen\06-Report\Report\_Graph\_B.py



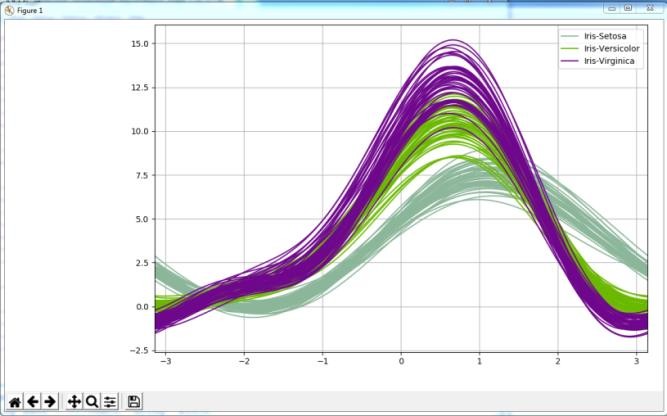
Scatter Matrix Graph

C:\VKHCG\01-Vermeulen\06-Report\Report\_Graph\_B.py



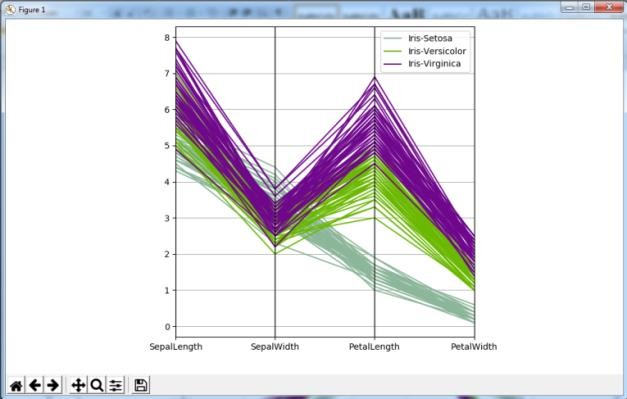
Andrews’ Curves

C:\VKHCG\01-Vermeulen\06-Report\Report\_Graph\_C.py



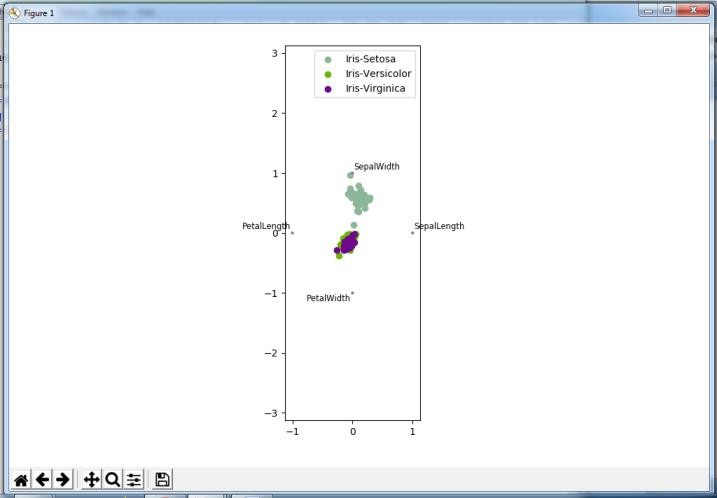
Parallel Coordinates

C:\VKHCG\01-Vermeulen\06-Report\Report\_Graph\_C.py



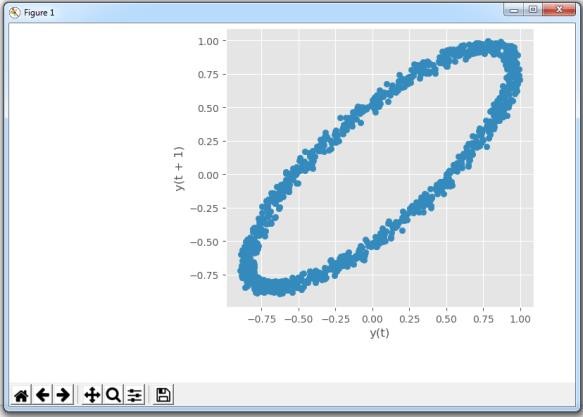
RADVIZ Method

C:\VKHCG\01-Vermeulen\06-Report\Report\_Graph\_C.py



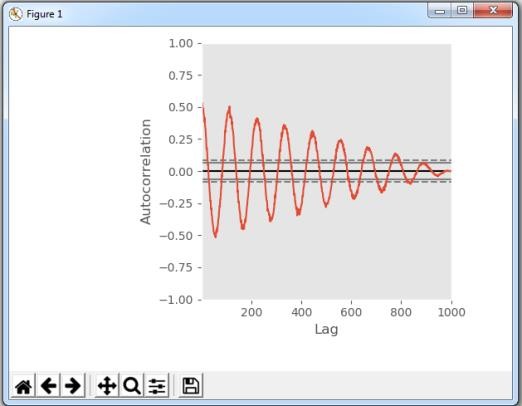
Lag Plot

C:\VKHCG\01-Vermeulen\06-Report\Report\_Graph\_D.py



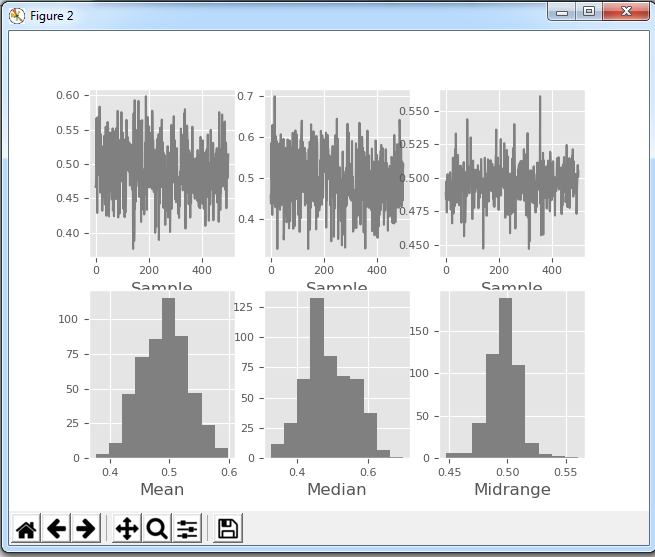
Autocorrelation Plot

C:\VKHCG\01-Vermeulen\06-Report\Report\_Graph\_D.py



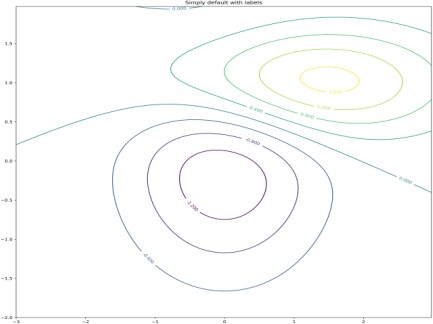
Bootstrap Plot

C:\VKHCG\01-Vermeulen\06-Report\Report\_Graph\_D.py



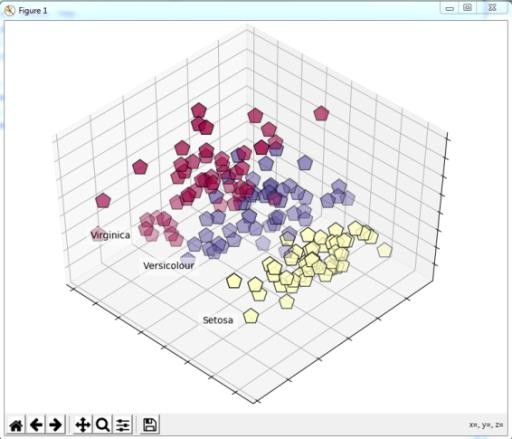
Contour Graphs

C:\VKHCG\01-Vermeulen\06-Report\Report\_Graph\_G.py



3D Graphs

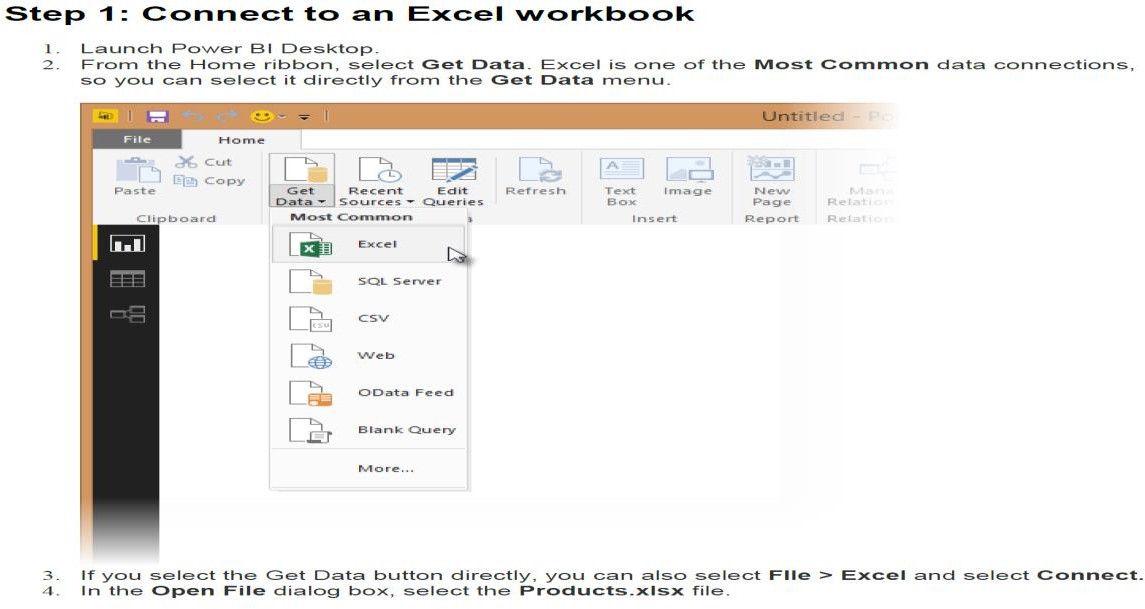
C:\VKHCG\01-Vermeulen\06-Report\Report\_PCA\_IRIS.py

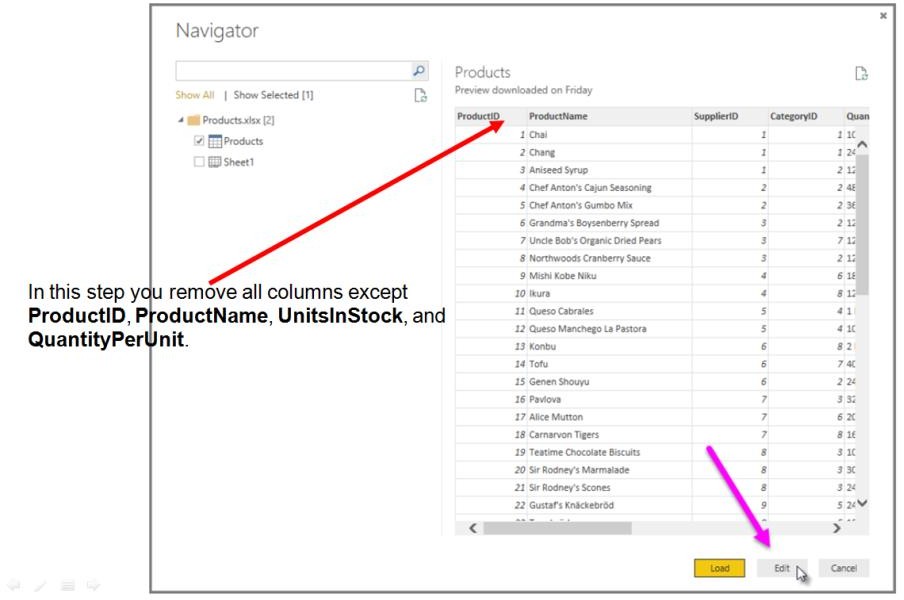


**Case Study : Sales Data**

**Practical 10**

**Data Visualization with Power BI**



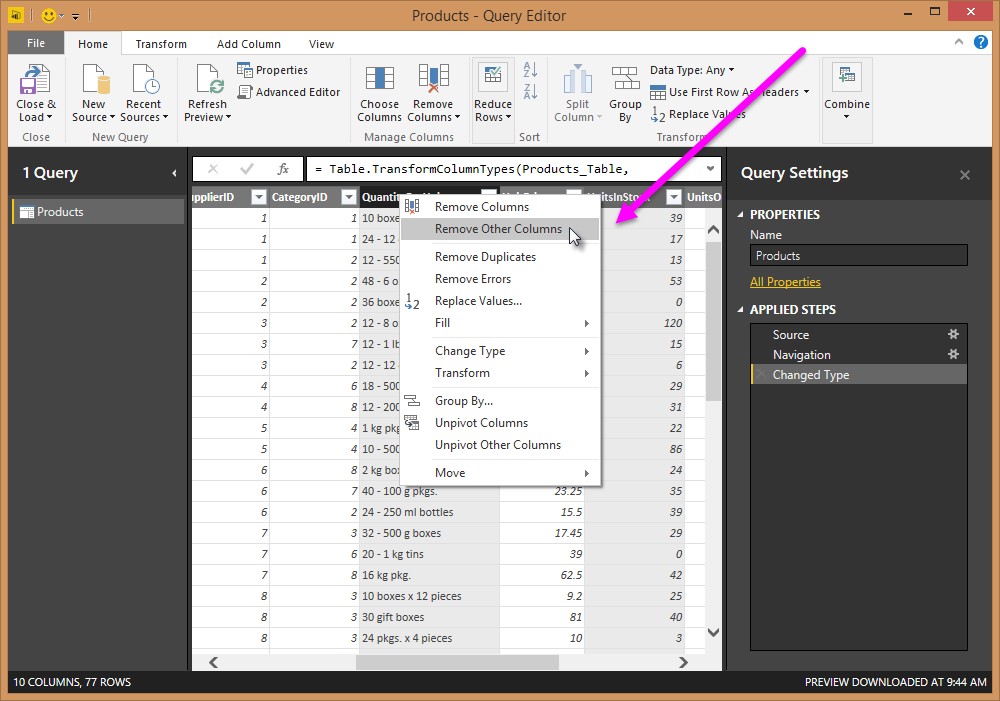


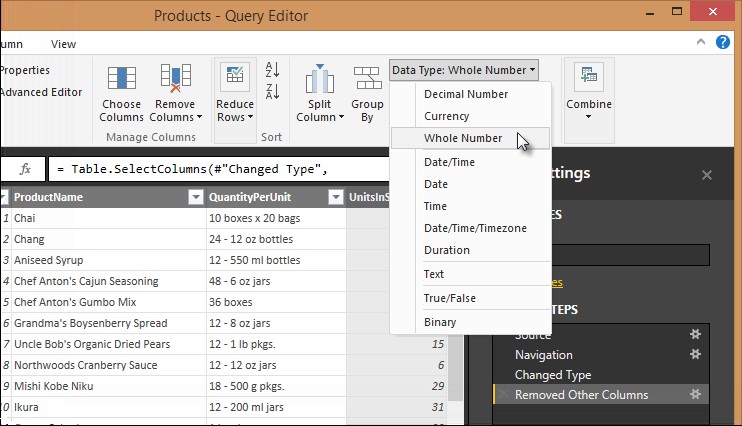
You can also open the Query Editor by selecting Edit Queries from the Home ribbon in Power BI Desktop. The following steps are performed in Query Editor.

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)*

1. 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.

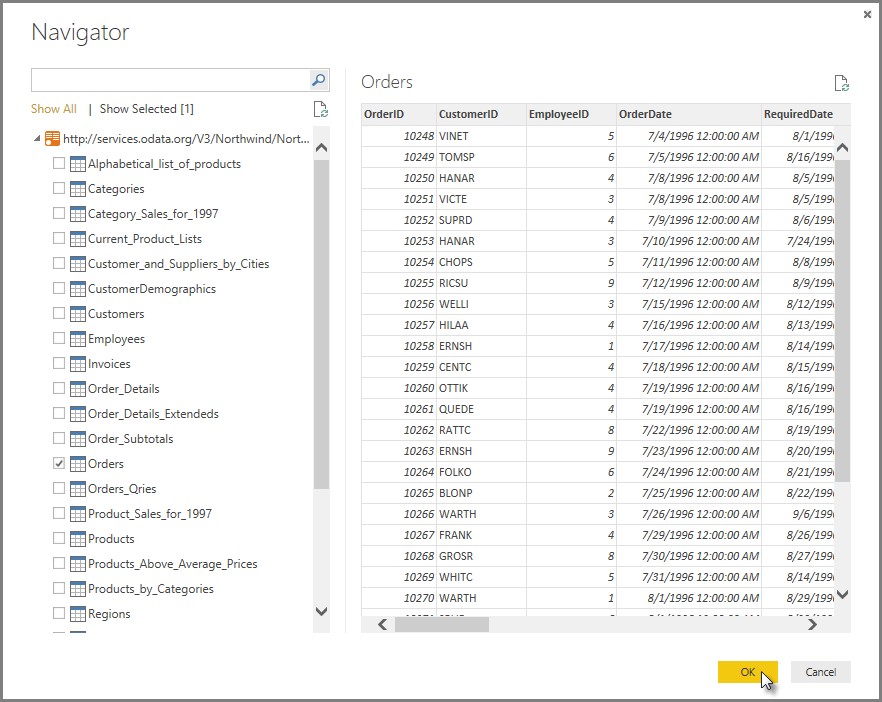
* 1. Select the UnitsInStock column.
  2. 2. Select the Data Type drop-down button in the Home ribbon.
  3. If not already a Whole Number, select Whole Number for data type from the drop down *(the Data Type: button also displays the data type for the current selection).*

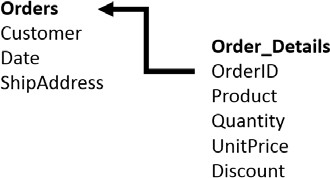
## Task 2: Import order data from an OData feed

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

**Step 1: Connect to an OData feed**

1. From the **Home** ribbon tab in Query Editor, select **Get Data.**
2. Browse to the **OData Feed** data source.
3. In the **OData Feed** dialog box, paste the **URL** for the Northwind OData feed.
4. Select **OK**.



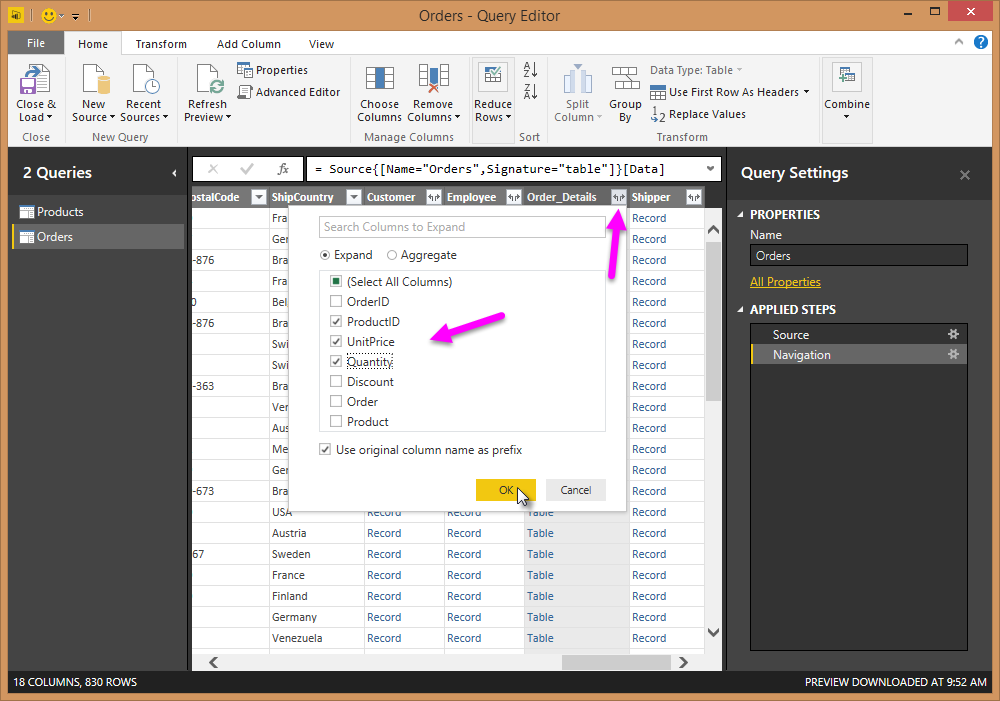
**Step 2: Expand the Order\_Details table**

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.

* 1. Click the first column (**OrderID**).
  2. Shift+Click the last column (**Shipper**).
  3. 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**

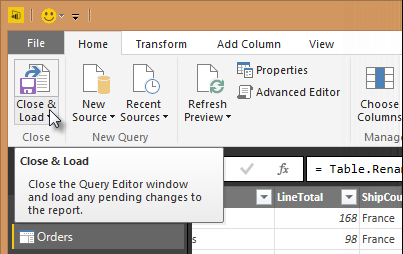
Power BI Desktop lets you to create calculations based on the columns you are importing, so you can enrich the data that you connect to. In this step, you create a **Custom Column** to calculate the line total for each **Order\_Details** row.

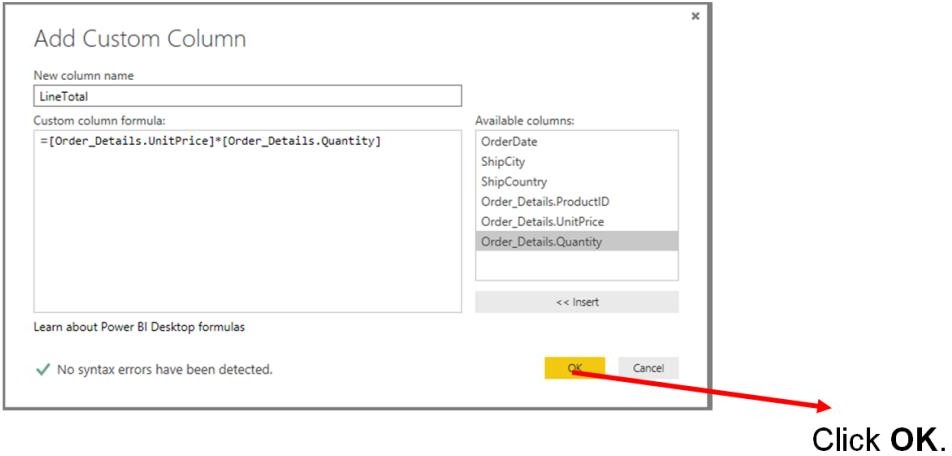
Calculate the line total for each **Order\_Details** row:

* + 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].**

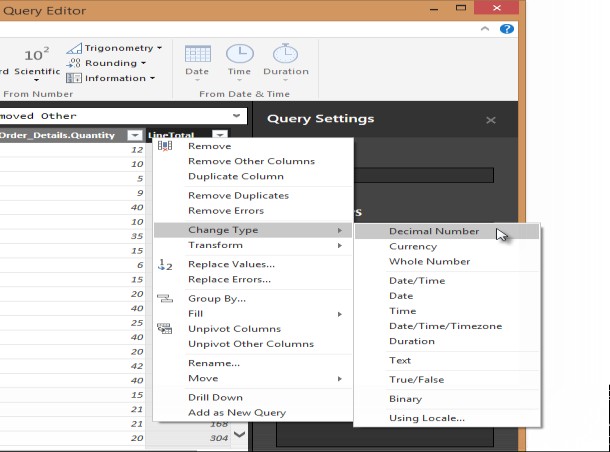
* + 1. In the New column name textbox, enter LineTotal.





**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

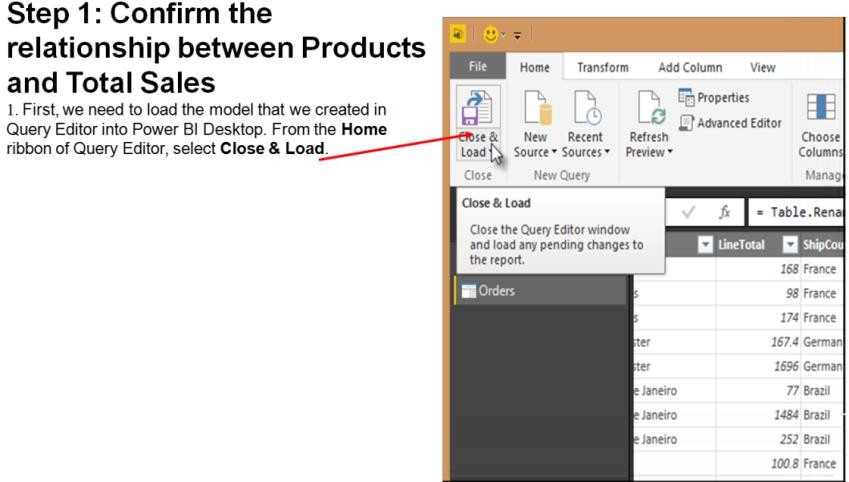
1. In **Query Editor**, drag the **LineTotal** column to the left, after **ShipCountry**.
2. Remove



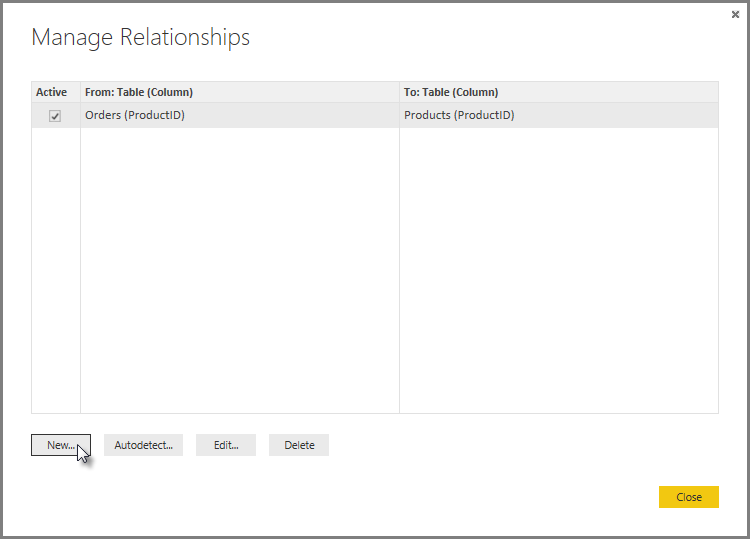
1. Remove the *Order\_Details.* prefix from the **Order\_Details.ProductID**, **Order\_Details.UnitPrice** and **Order\_Details.Quantity** columns, by double-clicking on each column header, and then deleting that text from the column name.

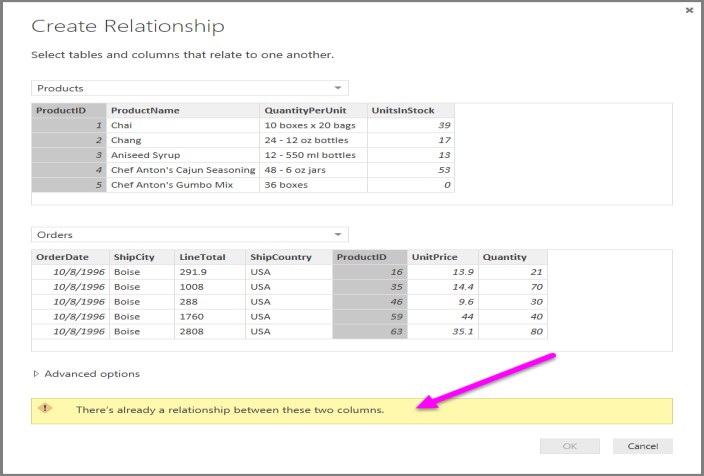


## Task 3: Combine the Products and Total Sales queries

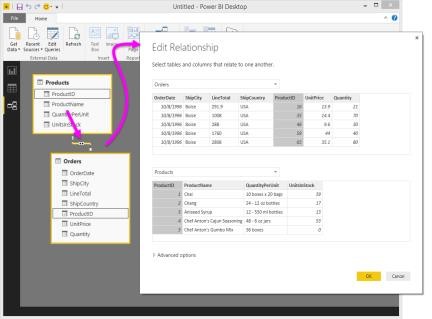
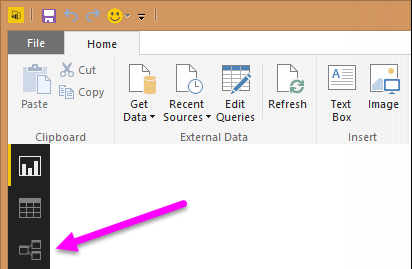


* 1. Power BI Desktop loads the data from the two queries
  2. Once the data is loaded, select the Manage Relationships button Home ribbon
  3. Select the New… button
  4. When we attempt to create the relationship, we see that one already exists! As shown in the Create Relationship dialog (by the shaded columns), the ProductsID fields in each query already have an established relationship.



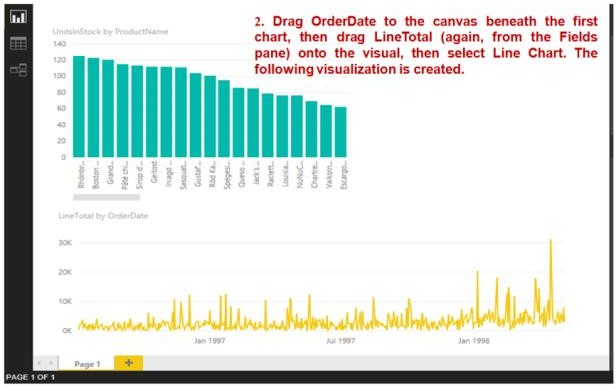
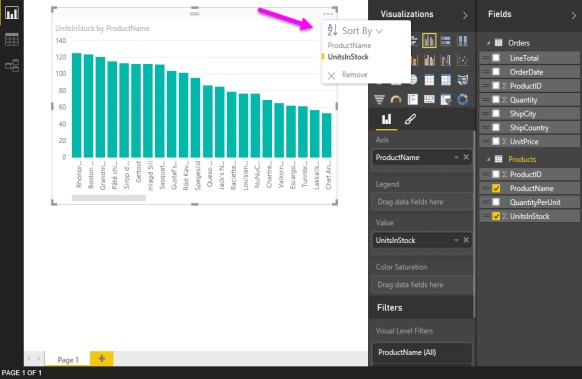


* 1. **Select Cancel, and then select Relationship view in Power BI Desktop.**

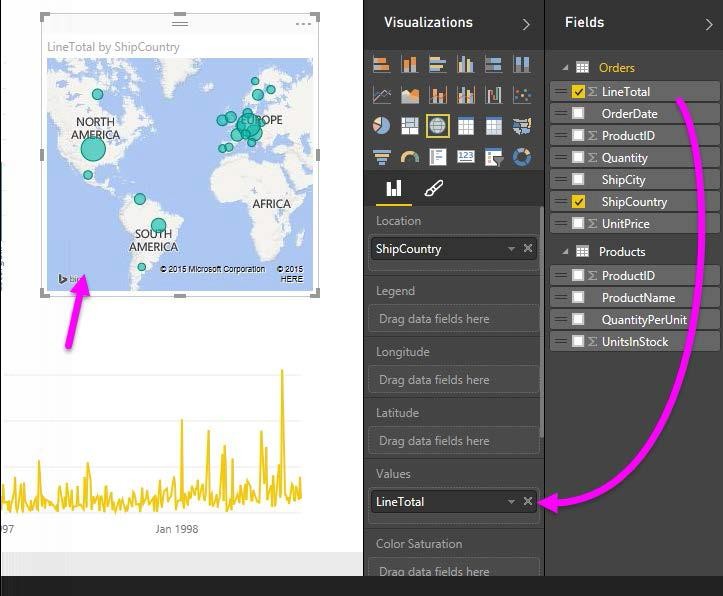


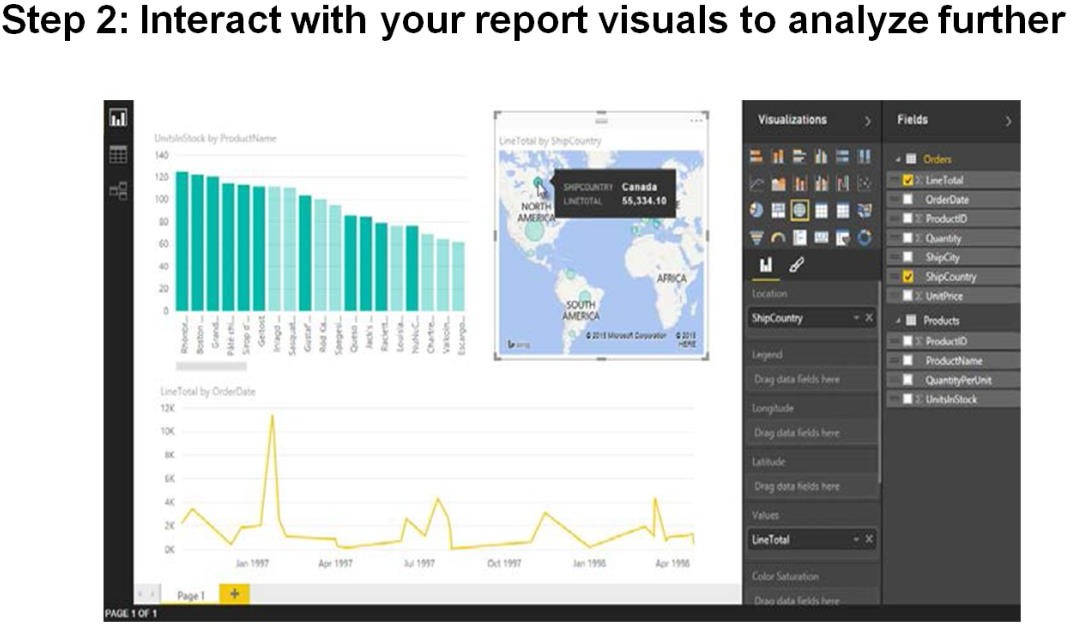
**Task 4: Build visuals using your data**

**Step 1: Create charts showing Units in Stock by Product and Total Sales by Year**



**3. 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.**





**~~~~~\*\*\*\*\*~~~~~**