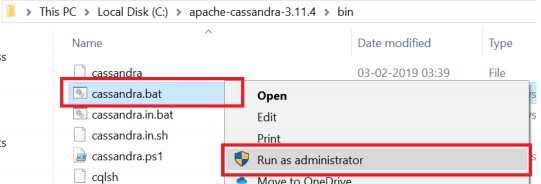
**Practical 1**

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

**Code:**

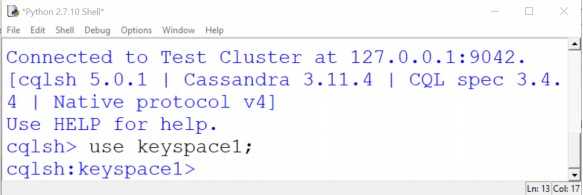
Go to Cassandra directory C:\apache-cassandra-3.11.4\bin

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');

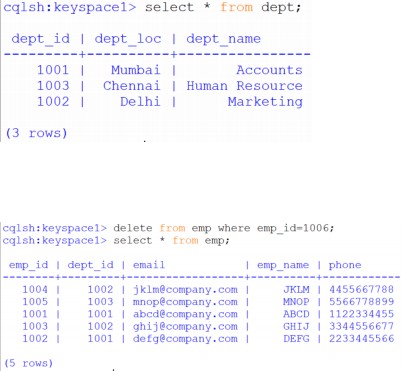
Insert into emp ( emp\_id, emp\_name, dept\_id, email, phone ) values (1004, 'JKLM', 1002, 'jklm@company.com', '4455667788');

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;



**Conclusion:** Thus implemented the experiment successfully

**Practical 2**

**Aim:- Conversion from different formats to HOURS format**

1. **Text delimited csv formats**

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

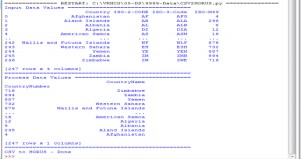
OutputData=ProcessData

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 :-**



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

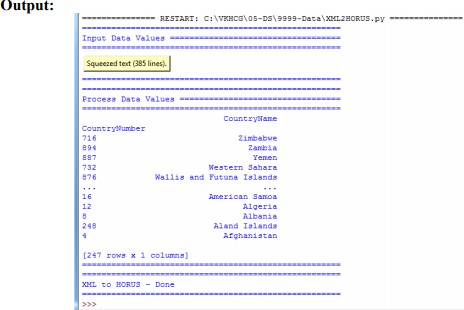
print(ProcessData) print('=====================================================')

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 :-**



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

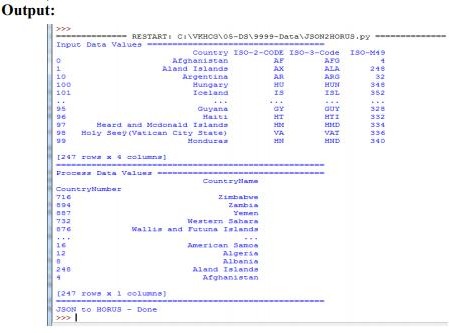
OutputData=ProcessData

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 :-**



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

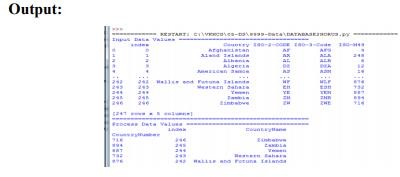
OutputData=ProcessData

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 :-**



### 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 :-**



### Video to HOURS Format Code:

**Movie to Frames**

# Utility Start Movie to HOURS (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 HOURS.

**Frames to Hours**

# 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])

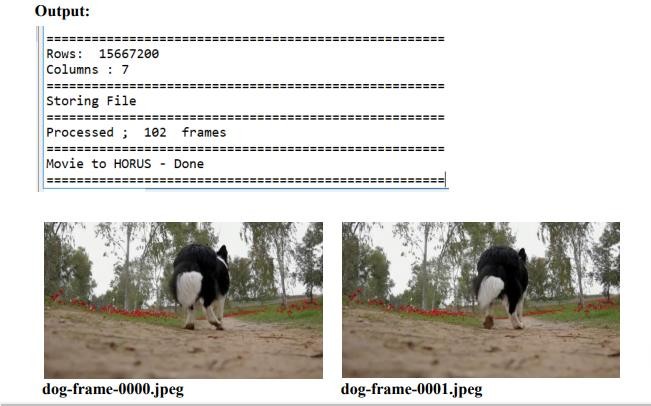
print('=====================================================') # Output Agreement ===========================================

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



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

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

### Audio to HOURS Format Code:

# Utility Start Audio to HOURS ===============================

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

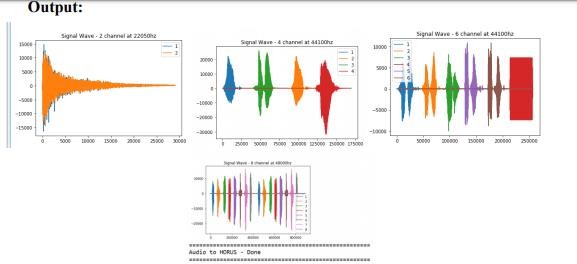
InputRate, InputData = wavfile.read(sInputFileName) show\_info("8 channel", InputData,InputRate) ProcessData=pd.DataFrame(InputData)

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 HOUS - Done')



**Practical 3.**

**Aim:-** Utilities and Auditing

**Code:**

#### 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)

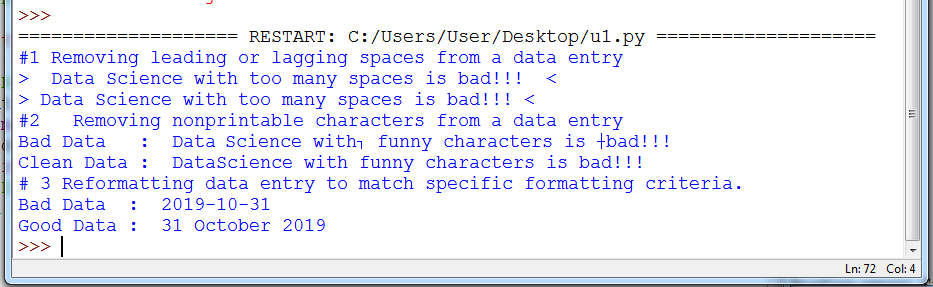
**# 3 Reformatting data entry to match specific formatting criteria**. # Convert YYYY/MM/DD to DD Month YYYY

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')

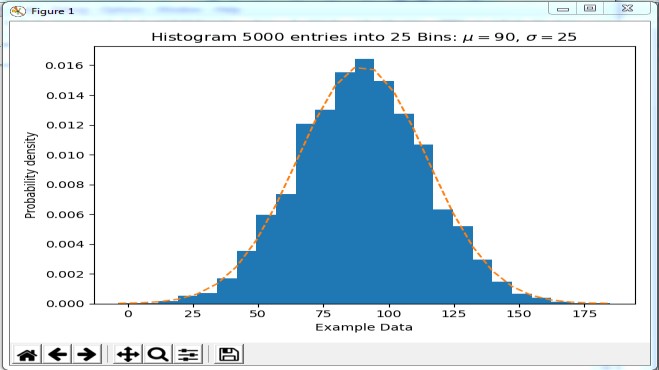
sTitle=r'Histogram ' + str(len(x)) + ' entries into ' + str(num\_bins) + ' Bins: $\mu=' + str(mu)

+ '$, $\sigma=' + str(sigma) + '$' 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:**



### 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 ')

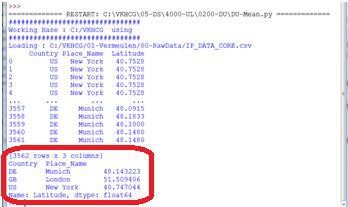
print('################################')

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

#### 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

### 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 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':

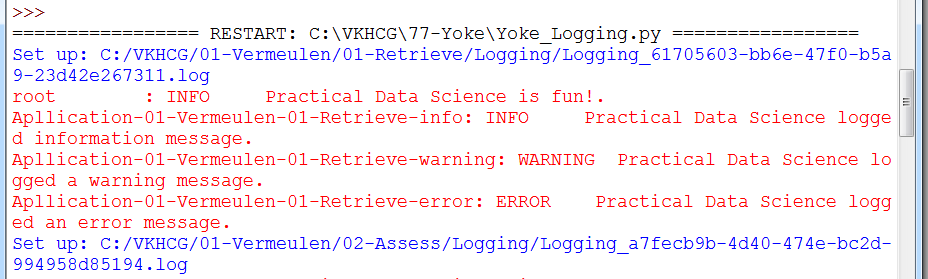
logger.info('Practical Data Science logged information message.') if sLevel == 'warning':

logger.warning('Practical Data Science logged a warning message.') if sLevel == 'error':

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

#

**Output:**



## Practical 4

### Aim: Program to retrieve different attributes of data.

### Code:

##### **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

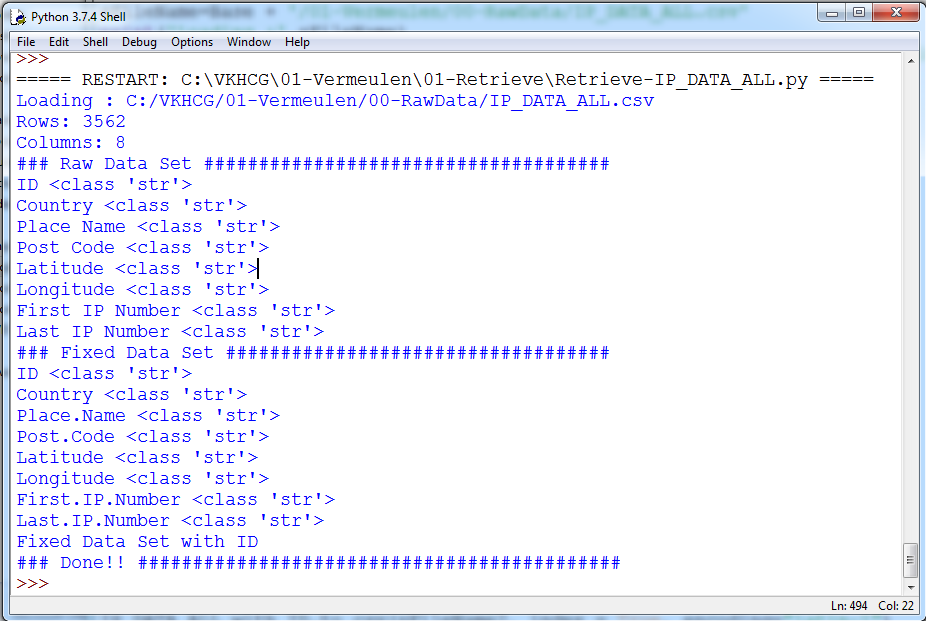
for i in range(0,len(IP\_DATA\_ALL.columns)): cNameOld=IP\_DATA\_ALL\_FIX.columns[i] + ' ' cNameNew=cNameOld.strip().replace(" ", ".") IP\_DATA\_ALL\_FIX.columns.values[i] = cNameNew print(IP\_DATA\_ALL.columns[i],type(IP\_DATA\_ALL.columns[i])) ################################################################

#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!! ############################################') ################################################################

**Output:**



**Conclusion**: Thus implemented the experiment successfully

**Practical 5.**

**Aim:-**  Assessing Data

**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])

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

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

1. 3.0 Good Better NaN 256.0 NaN 256.0 3
2. 4.0 Good Better Best NaN NaN 211.0 4
3. 5.0 Good Better NaN 64.0 NaN 6411.0 5
4. 6.0 Good NaN Best 32.0 NaN 32.0 6
5. 7.0 NaN Better Best 16.0 NaN 1611.0 7
6. 8.0 NaN NaN Best 8.0 NaN 8111.0 8
7. 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

1. 10.0 Good Better Best NaN NaN NaN 15
2. 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('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('################################')

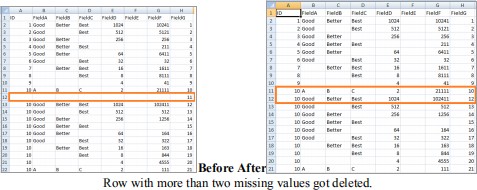
print(TestData) print('################################')

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

### Practical 6: Processing Data

#### A. 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('################')

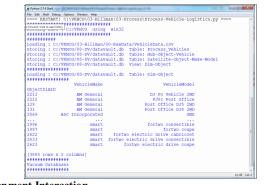
print('Vacuum Databases') sSQL="VACUUM;"

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)

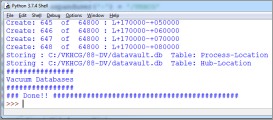
LocationHubIndex=LocationFrame.set\_index(['IDNumber'],inplace=False) sTable = 'Process-Location'

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.

# 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 :**



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

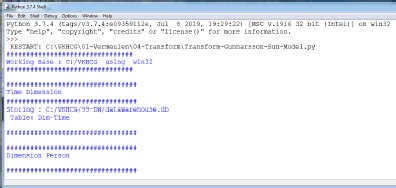
FctPersonTime=PersonTimeFrame FctPersonTimeIndex=FctPersonTime.set\_index(['IDNumber'],inplace=False) ################################################################

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)

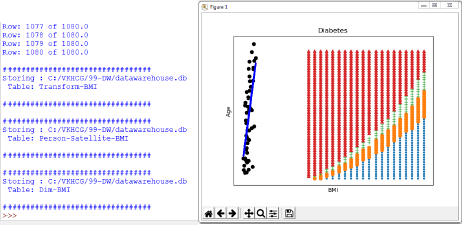
print('Coefficients: \n', regr.coef\_) print("Mean squared error: %.2f"

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

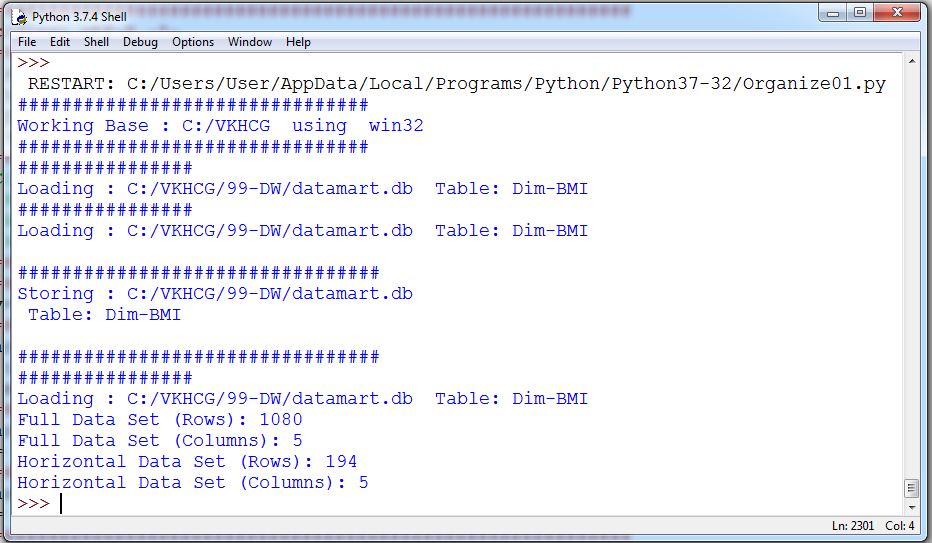
sTable = 'Dim-BMI'

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])



# 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];"

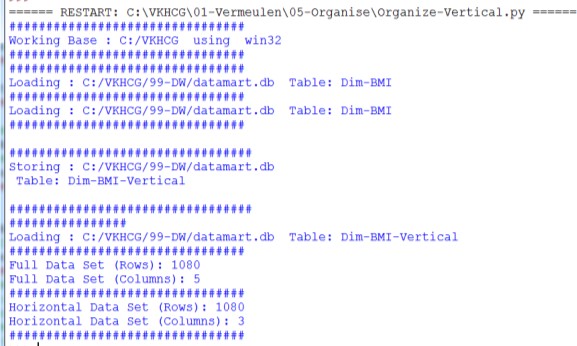
PersonFrame2=pd.read\_sql\_query(sSQL, conn2) ################################################################ print('################################')

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];"

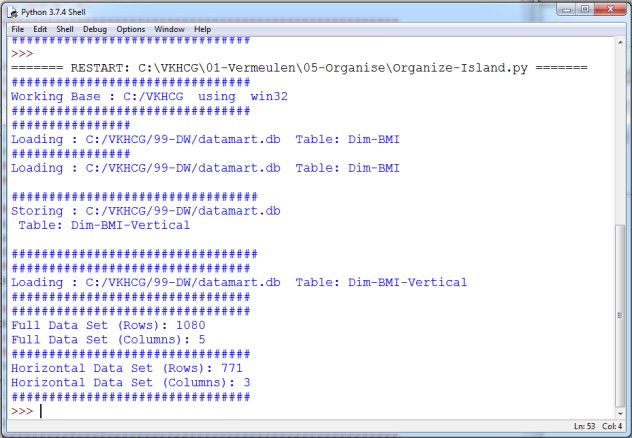
PersonFrame2=pd.read\_sql\_query(sSQL, conn2) ################################################################ print('################################')

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

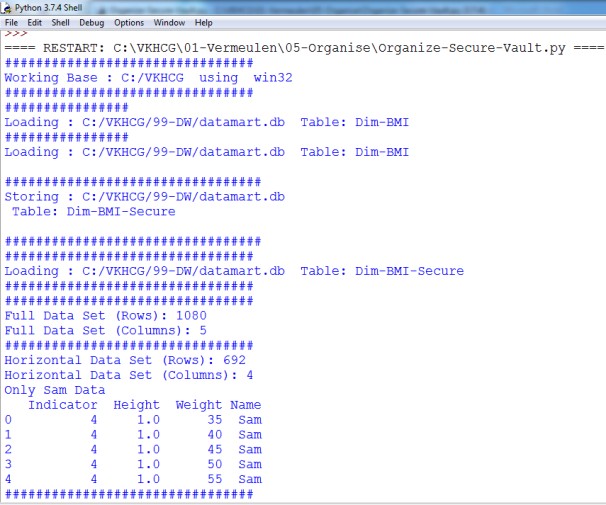
print('Full Data Set (Rows):', PersonFrame0.shape[0]) print('Full Data Set (Columns):', PersonFrame0.shape[1]) print('################################')

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'))

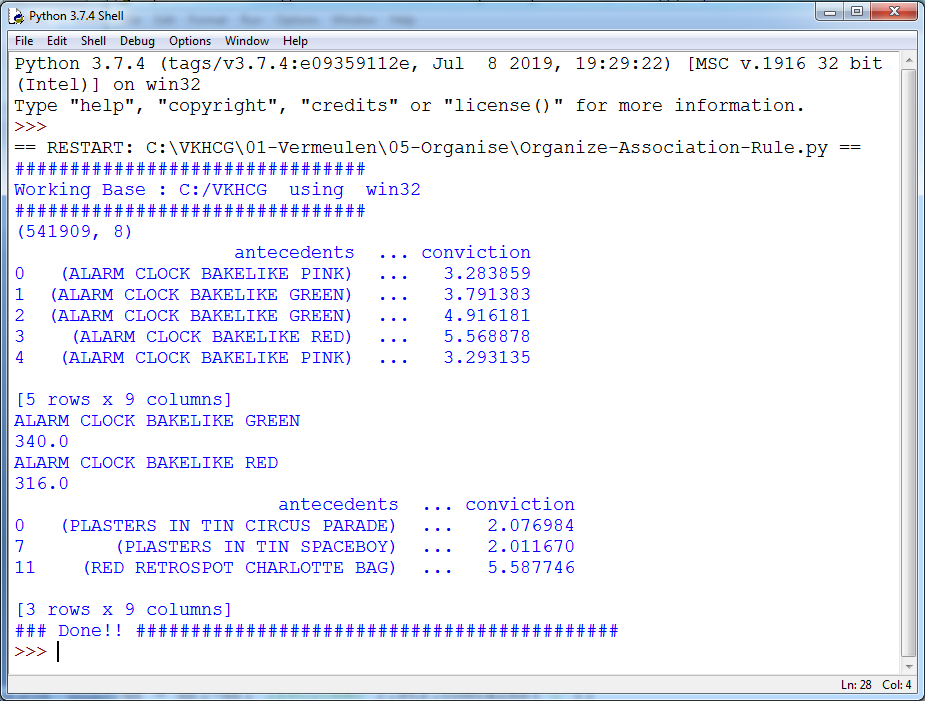
basket\_sets2 = basket2.applymap(encode\_units) basket\_sets2.drop('POSTAGE', inplace=True, axis=1)

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

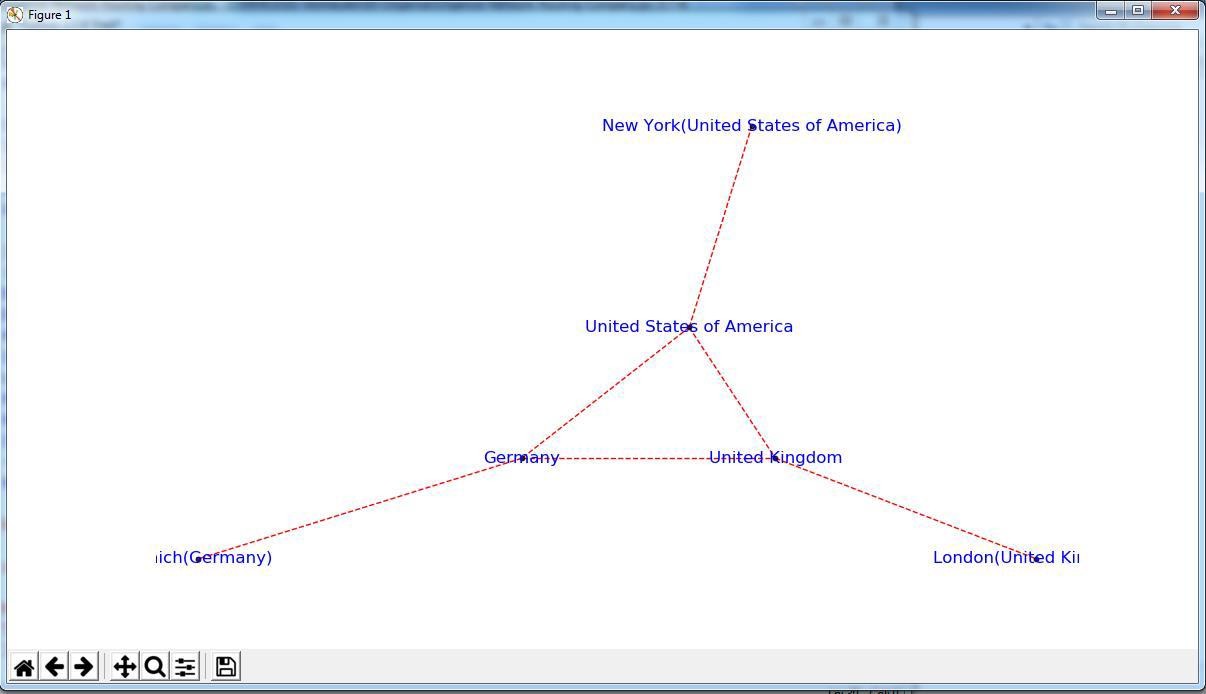
plt.figure(figsize=(15, 15)) pos=nx.spectral\_layout(G,dim=2)

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)

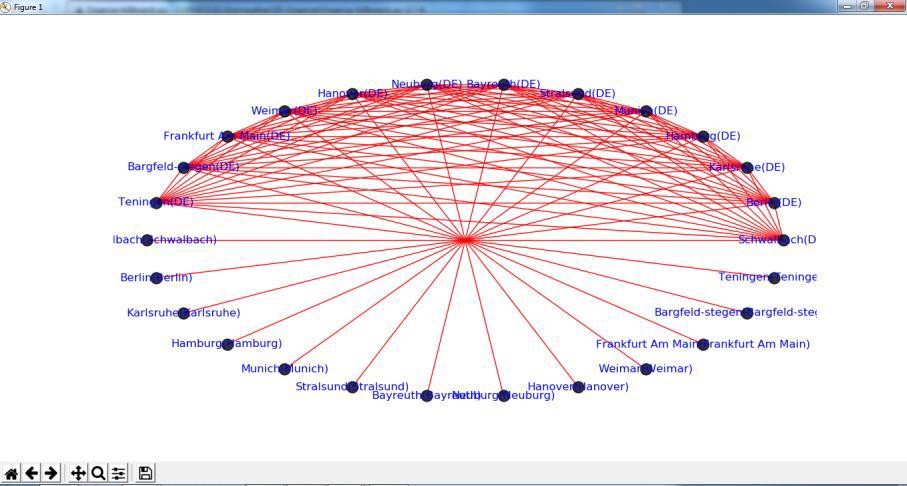
nx.draw\_networkx\_nodes(G,pos, node\_color='k', node\_size=150, alpha=0.8) nx.draw\_networkx\_edges(G, pos,edge\_color='r', arrows=False, style='solid') nx.draw\_networkx\_labels(G,pos,font\_size=12,font\_family='sans-serif',font\_color='b') plt.axis('off')

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:')

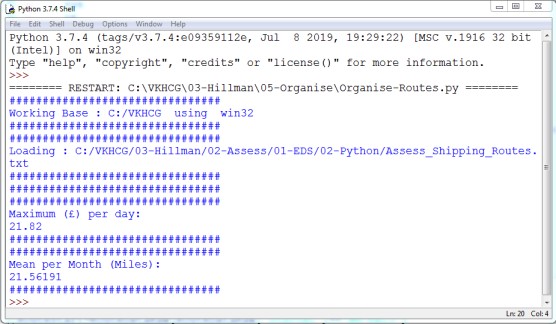
print(RouteMaxCost) print('################################')

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

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.