Python For RF Optimization
&
Planning Engineers

First Edition - Volume 1

Umer Saeed & Irfan Kareem

UNIVERSITY OF MANAGEMENT AND TECHNOLOGY, LAHORE, PAKISTAN
HTTPS://JOVIAN.AI/UMERSAEED81/PYTHONFORDATASCIENCE

First printing, June 2021

Preface

Python is an amazing language with a strong and friendly community of programmers. However, there is a lack of documentation on what to learn after getting the basics of Python down your throat. Through this book I aim to solve this problem. I would give you bits of information about some interesting topics which you can further explore.

The topics which are discussed in the book open up your mind toward some nice comers of Python language. This book is an outcome of my desire to have something like this when I was beginning to learn Python.

If you are beginner, intermediate or even an advanced programmer there is something for you in this book.

Please note that this book is not a tutorial and does not teach you Python. The topics are note explained in depth, instead only the minimum required information is given.

I love Python. Pandas New Era Excel!

Microsoft Excel is the industry leading spreadsheet software program, a powerful data visualization and analysis tool but it is not suitable for processing large amounts of data so I am sharing some common things a lot of people do in excel but using python's pandas package, for example vlookup, filtering data or pivot table.

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python.

This book is a continuous work in progress. If you find anything which you can further improve (I know you will find a lot of stuff) then kindly submit a pull request!

I am sure you are as excited as I am so let's start!

Contents

1. Merge GSM Worst Cells from *PRS*

1.1 Merge GSM Worst Cells

1.1.1 Input File Format

- Following *PRS Report* use to prepare the *Worst Cells* data;
  - WCL Central2 Today Worst Report
- Input File must be *.zip* and *.xlsx* Format

1.1.2 Import required Libraries

[1]:
```
import os
import zipfile
import pandas as pd
from glob import glob
```

1.1.3 Set Working Path

[2]:
```
working_directory = 'D:/DataSets/KPIs/2G_WC'
os.chdir(working_directory)
```

1.1.4 Unzip the Daily Worst Cell Files

[3]:
```
for file in os.listdir(working_directory):   # get the list of files
    if zipfile.is_zipfile(file): # if it is a zipfile, extract it
        with zipfile.ZipFile(file) as item: # treat the file as a zip
           item.extractall()  # extract it in the working directory
```

1.1.5 Import and Merge All the Files and Tabs

[4]:
```
all_files = glob('*.xlsx')
sheets = pd.ExcelFile(all_files[0]).sheet_names
dfs = {s: pd.concat(pd.read_excel(f, sheet_name=s,\
    converters={'Integrity': lambda value: '{:,.0f}%'.format(value * 100)})\
    for f in all_files) for s in sheets}
```

1.1.6 Delete Excel File

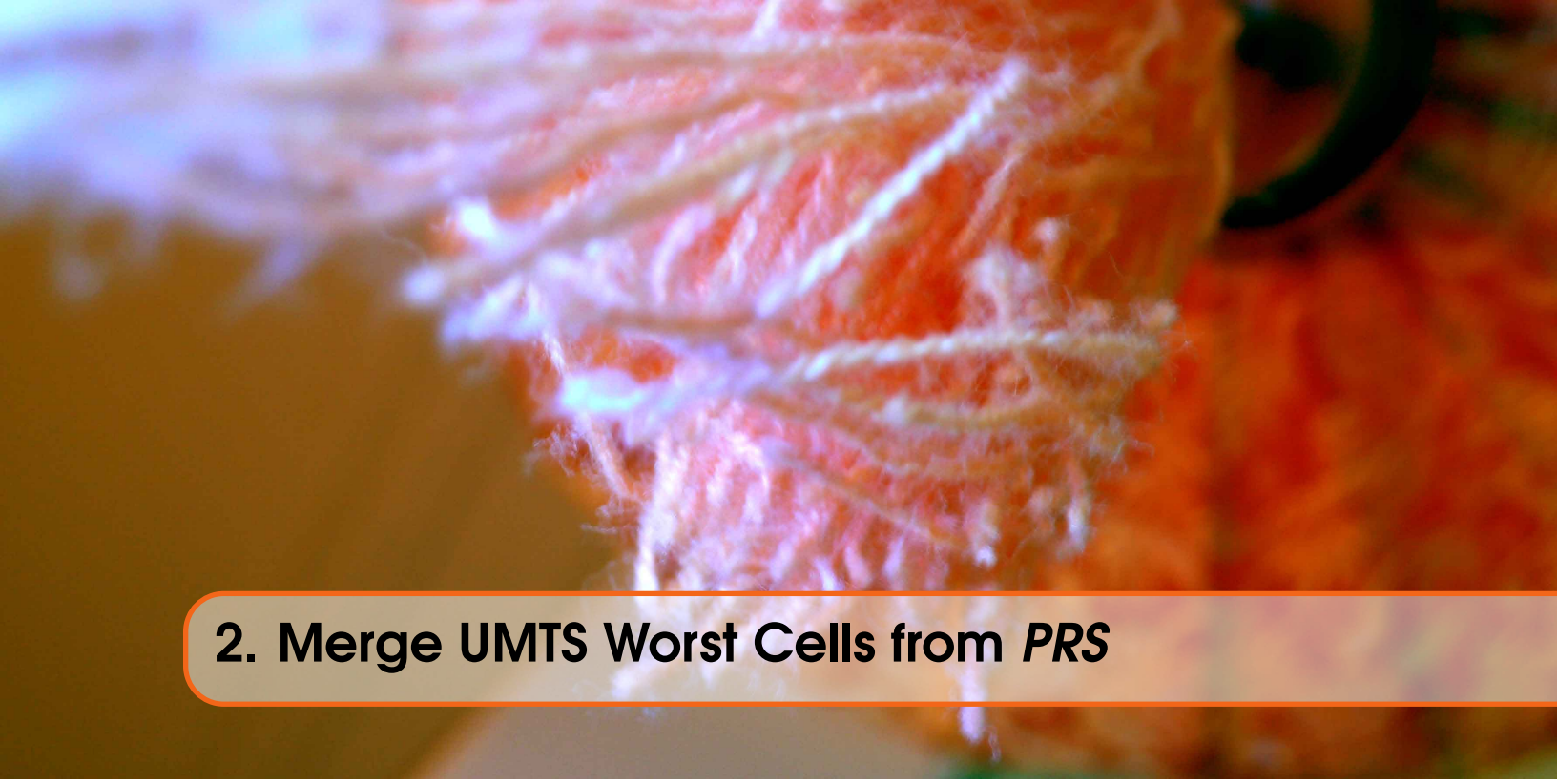
[5]:
```
for filename in os.listdir(working_directory):
    if filename.endswith('.xlsx'):
        os.unlink(os.path.join(working_directory, filename))
```

1.1.7 Export Final Data Set

[6]:
```
with pd.ExcelWriter('2G_WC.xlsx') as writer:
    dfs['CSSR'].\
    to_excel(writer,sheet_name="CSSR",engine='openpyxl',index=False)
    dfs['DCR'].\
    to_excel(writer,sheet_name="DCR",engine='openpyxl',index=False)
    dfs['Inc HSR'].\
    to_excel(writer,sheet_name="Inc HSR",engine='openpyxl',index=False)
```

1.1.8 Output File Format

- Output File Format



2. Merge UMTS Worst Cells from *PRS*

2.1 Merge UMTS Worst Cells

2.1.1 Input File Format

- Following *PRS Report* use to prepare the *Worst Cells* data;
  - WCL Central2 Today Worst Report
- Input File must be *.zip* and *.xlsx* Format

2.1.2 Import required Libraries

[1]:
```
import os
import zipfile
import pandas as pd
from glob import glob
```

2.1.3 Set Working Path

[2]:
```
working_directory = 'D:/DataSets/KPIs/3G_WC'
os.chdir(working_directory)
```

2.1.4 Unzip the Daily Worst Cell Files

[3]:
```
for file in os.listdir(working_directory):   # get the list of files
    if zipfile.is_zipfile(file): # if it is a zipfile, extract it
        with zipfile.ZipFile(file) as item: # treat the file as a zip
           item.extractall()  # extract it in the working directory
```

2.1.5 Import and Merge All the Files and Tabs

[4]:
```
all_files = glob('*.xlsx')
sheets = pd.ExcelFile(all_files[0]).sheet_names
dfs = {s: pd.concat(pd.read_excel(f, sheet_name=s,\
    converters={'Integrity': lambda value: '{:,.0f}%'.format(value * 100)}) \
        for f in all_files) for s in sheets}
```

2.1.6 Add Blank Columns

[5]:
```
dfs['CSSR (%)']["Comments"] = ''
dfs['CSSR (%)']["Bottleneck"] = ''
dfs['CSSR (%)']["Status"] = ''
dfs['RRC SSR (%)']["Comments"] = ''
dfs['RRC SSR (%)']["Bottleneck"] = ''
dfs['RRC SSR (%)']["Status"] = ''
```

2.1.7 Delete Excel File

[6]:
```
for filename in os.listdir(working_directory):
    if filename.endswith('.xlsx'):
        os.unlink(os.path.join(working_directory, filename))
```

2.1.8 Export Final Data Set

[7]:
```
writer = pd.ExcelWriter('3G_WCELL.xlsx')
for key in dfs:
        dfs[key].to_excel(writer, key,index=False)
writer.save()
```

2.1.9 Output File Format

- Output File Format

3. LAC TAC Convert Hexadecimal to Decimal

3.1 Convert Hexadecimal to Decimal

3.1.1 Input File Format

- LAC and TAC Values in Hexadecimal

3.1.2 Import required Libraries

[1]:
```
import os
import pandas as pd
```

3.1.3 Set Working Path

[2]:
```
folder_path = 'D:/DataSets/DSP/LAC_TAC'
os.chdir(folder_path)
```

3.1.4 Import Data Set

[3]:
```
df=pd.read_csv('TACLAC.txt')
```

3.1.5 LAC and TAC Convert Hex to Dec

[4]:
```
df['TAC'] = df['TAC   LAC'].str.split(' ').str[0].apply(lambda x: int(x, 16))
df['LAC'] = df['TAC   LAC'].str.split(' ').str[2].apply(lambda x: int(x, 16))
```

3.1.6 Export Final Data Set

[5]:
```
df.to_csv('Final_Values.csv',index=False)
```

3.1.7 Output File Format

- Output File Format

4. Cell on Cluster Busy Hour Filtering

4.1 GSM Cell on Cluster Busy Hour

4.1.1 Input File Format

Following *PRS Report* use to prepare the Cell On Cluster Busy Hour data;

- 2G Cell Hourly Counters
- Cluster BH Report
- Input File must be *.zip* and *.csv* Format, *Date and Time must be in different columns*

4.1.2 Import required Libraries

[1]:
```
import os
import zipfile
import pandas as pd
from glob import glob
import dask.dataframe as dd
```

4.1.3 Set Working Path For Cell Hourly KPIs

[2]:
```
working_directory = 'D:/DataSets/KPIs/2G_Num_Dem'
os.chdir(working_directory)
```

4.1.4 Unzip the Cell Hourly Files

[3]:
```
%%time
for file in os.listdir(working_directory):   # get the list of files
    if zipfile.is_zipfile(file): # if it is a zipfile, extract it
        with zipfile.ZipFile(file) as item: # treat the file as a zip
           item.extractall()  # extract it in the working directory
```

```
Wall time: 44.3 s
```

4.1.5 Import Cell Hourly Data

[4]:
```
%%time
cell = dd.read_csv('*.csv',\
                   skiprows=[0,1,2,3,4,5],\
                   skipfooter=1,
                    engine='python',\
                    na_values=['NIL','/0'],
                    parse_dates=["Date"],assume_missing=True)
```

```
Wall time: 187 ms
```

4.1.6 Set Working Path For Cluster BH KPIs

[5]:
```
folder_path = 'D:/DataSets/Conformance/Quaterly Conformance Working'
os.chdir(folder_path)
```

4.1.7 Unzip the Cluster BH Files

[6]:
```
%%time
for file in os.listdir(folder_path):   # get the list of files
    if zipfile.is_zipfile(file): # if it is a zipfile, extract it
        with zipfile.ZipFile(file) as item: # treat the file as a zip
           item.extractall()  # extract it in the working directory
```

```
Wall time: 208 ms
```

4.1.8 Import Cluster BH KPIs

[7]:
```
busy_hour_files = sorted(glob('Umer Saeed_Cluster_BH*.csv'))
# concat all the Cluster DA Files
cluster=pd.concat((pd.read_csv(file,skiprows=[0,1,2,3,4],\
                           skipfooter=1,engine='python',\
                           usecols=['Date','Time','GCell Group'],\
    parse_dates=["Date"]) for file in busy_hour_files)).sort_values('Date')
cluster=cluster.rename(columns={"GCell Group":"Location"})
```

4.1.9 Merge Cell On Cluster BH

[8]:
```
%%time
ccbh = dd.merge(cell,cluster,on=['Date','Time','Location'])
```

```
Wall time: 40.1 ms
```

[9]:
```
%%time
ccbh = ccbh.compute()
```

```
Wall time: 16min 49s
```

4.1.10 Remove Duplicates

[10]:
```
ccbh = ccbh.drop_duplicates()
```

4.1.11 Cell On Cluster Busy Hour Output

[12]:
```
working_directory = 'D:/DataSets/KPIs/2G_Num_Dem/Output'
os.chdir(working_directory)
```

[13]:
```
for f in os.listdir(working_directory):
    os.remove(os.path.join(working_directory, f))
```

4.1.12 Export Final Data Set

[14]:
```
for i, g in ccbh.groupby('Location'):
    g.to_csv('Loc_{}.csv'.format(i), header=True,index=False)
```

4.1.13 Delete unzip Files

[15]:
```
working_directory = 'D:/DataSets/KPIs/2G_Num_Dem'
os.chdir(working_directory)
```

[16]:
```
for filename in os.listdir(working_directory):
    if filename.endswith('.csv'):
        os.unlink(os.path.join(working_directory, filename))
```

[17]:
```
folder_path = 'D:/DataSets/Conformance/Quaterly Conformance Working'
os.chdir(folder_path)
```

[18]:
```
for filename in os.listdir(folder_path):
    if filename.endswith('.csv'):
        os.unlink(os.path.join(folder_path, filename))
```

4.1.14 Output File Format

- Output File Format

5. UMTS IP-Pool KPIs

5.1 IP-Pool KPIs Summary

5.1.1 Input File Format

- UMTS IPPool Hourly KPIs

5.1.2 Import required Libraries

[1]:
```
import os
import pandas as pd
from glob import glob
```

5.1.3 Set Working Path

[2]:
```
working_directory = 'D:/DataSets/KPIs/3GIPPool'
os.chdir(working_directory)
```

5.1.4 SLA Values

[3]:
```
df=pd.DataFrame({
    'KPI':['VS.IPPOOL.ADJNODE.PING.MeanDELAY(ms)',\
           'VS.IPPOOL.ADJNODE.PING.MeanJITTER(ms)',\
           'VS.IPPOOL.ADJNODE.PING.MeanLOST(%)'],
                'Target Value':[20,2,0.1]})
```

5.1.5 Import IPPool Stats

[4]:
```
#3G IPP Files list to be merge
df0 = sorted(glob('*.csv'))
# concat all the Cluster DA Files
df1=pd.concat((pd.read_csv(file,header=3,skipfooter=1,engine='python',\
            na_values=['NIL','/0']\
            ,usecols=['Date', 'Time', 'RNC', \
            'Adjacent Node ID', \
            'VS.IPPOOL.ADJNODE.PING.MeanDELAY(ms)',\
            'VS.IPPOOL.ADJNODE.PING.MeanJITTER(ms)',\
            'VS.IPPOOL.ADJNODE.PING.MeanLOST(%)'])\
            for file in df0),ignore_index=True).fillna(0)
```

5.1.6 Re-Shape Data Set(IPPool Stats Stats)

[5]:
```
df2=pd.melt(df1,\
    id_vars=['Date', 'Time', 'RNC','Adjacent Node ID'],\
    var_name="KPI", value_name='KPI-Value')
```

5.1.7 Merge Re-Shape Data Set(IPPool Stats) with SLA Values

[6]:
```
df3 = pd.merge(df2,df,on=['KPI'])
```

5.1.8 Find Daily Adjacent Node IDMax Values

[7]:
```
df4 = df3.loc[df3.groupby(['Date','RNC','Adjacent Node ID',\
            'KPI','Target Value'])['KPI-Value'].idxmax()]
```

5.1.9 Re-Shape MaxID Values

[8]:
```
df5 = pd.DataFrame()
for u in df4['KPI'].unique():
    pivot = pd.pivot_table(df4[df4['KPI'] == u], \
            index=["RNC",'Adjacent Node ID','KPI','Target Value']\
            ,columns=["Date"],values = 'KPI-Value').reset_index()
    df5= df5.append(pivot)
```

5.1.10 Compare the value with Target Value

[9]:
```
df3['qe'] = (df3['KPI-Value']>=df3['Target Value']).astype(str)
```

5.1.11 Crosstab to Calculate the number of Interval

[10]:
```
df6=pd.crosstab([df3["RNC"],df3["Adjacent Node ID"],df3["KPI"]],\
            df3['qe']).reset_index().fillna(0).\
            rename(columns={"False":"#_of_Intervalus_Value_Below_Threshold",\
            "True":"#_of_Intervalus_Value_Above_Threshold"})
```

5.1.12 Merge Crosstab Data With Max Value Table

[11]:
```
df7 =pd.merge(df6,df5,on=['RNC','Adjacent Node ID','KPI'])
```

5.1.13 Calculate the Number of day breach the Target

[12]:
```
df7['Count']=df7.iloc[:,6:].ge(df7.iloc [:,5],axis=0).sum(axis=1)
```

5.1.14 Export Final Data Set

[13]:
```
df7.to_csv('Output.csv',index=False)
```

5.1.15 Output File Format

- Output File Format

6. UMTS IPPM KPIs

6.1 IPPM KPIs Summary

6.1.1 Input File Format

- UMTS IPPM Hourly KPIs

6.1.2 Import required Libraries

[1]:
```
import os
import pandas as pd
from glob import glob
```

6.1.3 Import IPPM Stats

[4]:
```
#3G IPP Files list to be merge
df0 = sorted(glob('*.csv'))
# concat all the Cluster DA Files
df1=pd.concat((pd.read_csv(file,header=3,skipfooter=1,engine='python',\
    na_values=['NIL','/0']\
    ,usecols=['Date', 'Time', 'RNC', \
    'Adjacent Node ID', \
    'VS.IPPM.Back.JitterStandardDeviation(ms)',\
    'VS.IPPM.Forward.JitterStandardDeviation(ms)',\
    'VS.IPPM.Forword.DropMeans(%)','VS.IPPM.Rtt.Means(ms)'])\
    for file in df0),ignore_index=True).fillna(0)
```

6.1.4 Re-Shape Data Set(IPPM Stats)

[5]:
```
df2=pd.melt(df1,\
    id_vars=['Date', 'Time', 'RNC','Adjacent Node ID'],\
    var_name="KPI", value_name='KPI-Value')
```

6.1.5 Merge Re-Shape Data Set(IPPM Stats) with SLA Values

[6]:
```
df3 = pd.merge(df2,df,on=['KPI'])
```

6.1.6 Find Daily Adjacent Node IDMax Values

[7]:
```
df4 = df3.loc[df3.groupby(['Date','RNC','Adjacent Node ID',\
            'KPI','Target Value'])\
              ['KPI-Value'].idxmax()]
```

6.1.7 Re-Shape MaxID Values

[8]:
```
df5 = pd.DataFrame()
for u in df4['KPI'].unique():
    pivot = pd.pivot_table(df4[df4['KPI'] == u], \
    index=["RNC",'Adjacent Node ID','KPI','Target Value'],\
    columns=["Date"],values = 'KPI-Value').reset_index()
    df5= df5.append(pivot)
```

6.1.8 Compare the value with Target Value

[9]:
```
df3['qe'] = (df3['KPI-Value']>=df3['Target Value']).astype(str)
```

6.1.9 Crosstab to Calculate the number of Interval

[10]:
```
df6=pd.crosstab([df3["RNC"],df3["Adjacent Node ID"],df3["KPI"]],\
    df3['qe']).reset_index().fillna(0).\
    rename(columns={"False":"#_of_Intervalus_Value_Below_Threshold",\
    "True":"#_of_Intervalus_Value_Above_Threshold"})
```

6.1.10 Merge Crosstab Data With Max Value Table

[11]:
```
df7 =pd.merge(df6,df5,on=['RNC','Adjacent Node ID','KPI'])
```

6.1.11 Calculate the Number of day breach the Target

[12]:
```
df7['Count']=df7.iloc[:,6:].ge(df7.iloc [:,5],axis=0).sum(axis=1)
```

6.1.12 Export Final Data Set

[13]:
```
df7.to_csv('Output.csv',index=False)
```

6.1.13 Output File Format

- Output File Format

7. GSM IoI KPI

7.1 GSM IOI KPI Summary

7.1.1 Input File Format

- GSM External Interference DA and Hourly KPIs

7.1.2 Import required Libraries

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
from glob import glob
```

7.1.3 Set Working Path For 2G(DA) IOI KPIs

[2]:
```
folder_path = 'D:/DataSets/KPIs/IOI_KPIs/2G/DA'
os.chdir(folder_path)
```

7.1.4 2G DA - IOI and calculate Average

[3]:
```
da = sorted(glob('*.csv'))
df_2g_da=pd.concat((pd.read_csv(file,header=3,\
    skipfooter=1,engine='python',na_values=['NIL','/0'],\
    parse_dates=["Date"]) for file in da))\
    .sort_values('Date').set_index(['Date']).last('10D').reset_index()
```

[4]:
```
#calucalate average
df_2g_da_avg=df_2g_da.groupby(['GBSC','Cell CI']).\
    apply(lambda x: np.average(x['Interference Band Proportion (4~5)(%)']))\
    .reset_index(name='2G DA IoI')
```

7.1.5 Set Working Path For 2G(Hourly) IOI KPIs

[5]:
```
folder_path = 'D:/DataSets/KPIs/IOI_KPIs/2G/Hourly'
os.chdir(folder_path)
```

7.1.6 2G Hourly - IOI :Calculate Max IOI Per Cell Per Day

[6]:
```
bh= sorted(glob('*.csv'))
df_2g_hr=pd.concat((pd.read_csv(file,header=3,\
    skipfooter=1,engine='python',na_values=['NIL','/0'],\
    parse_dates=["Date"]) for file in bh))\
    .fillna(-6666).sort_values('Date')\
    .set_index(['Date']).last('10D').reset_index()
```

[7]:
```
# assign Date to Date Number as category
df_2g_hr['Day']='Day'+'-'+df_2g_hr.Date.astype("category")\
                .cat.codes.astype(str)+'-'+'MaxIOI'
```

[8]:
```
# idendtify the max interference index
df_2g_hr_max = df_2g_hr.loc[df_2g_hr.groupby(['Date','GBSC','Cell CI'])\
            ['Interference Band Proportion (4~5)(%)'].idxmax()]
```

[9]:
```
# re-shapre data frame
df_2g_hr_rs= df_2g_hr_max.pivot_table(index=['GBSC','Cell CI'],\
            columns="Day",values='Interference Band Proportion (4~5)(%)')\
            .reset_index()
```

7.1.7 2G Hourly - IOI :Calculate Number of Interval Ioi >=10

[10]:
```
# re-shapre
df_2g_hr_gt_10_interval=df_2g_hr.groupby(['GBSC','Cell CI'])\
        ['Interference Band Proportion (4~5)(%)'].\
        apply(lambda x: (x.ge(10)).sum())\
        .reset_index(name='Total Interval IOI>10')
```

7.1.8 2G IOI - Final Data Set

[11]:
```
# IOI stats Output
df_2g_fds = df_2g_da_avg.\
    merge(df_2g_hr_gt_10_interval,on=['GBSC','Cell CI'],how='left').\
    merge(df_2g_hr_rs,on=['GBSC','Cell CI'],how='left')
```

[12]:
```
#row count
df_2g_fds["#of Day IOI>10"] = df_2g_fds.where(df_2g_fds.iloc[:,4:] >= 10).
 ↪count(1)
```

7.1.9 Set Working Path For Output

[13]:
```
folder_path = 'D:/DataSets/KPIs/IOI_KPIs/2G'
os.chdir(folder_path)
```

7.1.10 Formatting Data Set

[14]:
```
# conditional colour
df12= df_2g_fds.style.\
    applymap(lambda x: 'color: black' \
             if pd.isnull(x) \
             else 'background-color: %s' % 'red' \
             if x>=10 else 'background-color: %s' % 'green',\
             subset=[('2G DA IoI'),('Day-0-MaxIOI'),\
                     ('Day-1-MaxIOI'),('Day-2-MaxIOI'),\
                     ('Day-3-MaxIOI'),('Day-4-MaxIOI'),\
                     ('Day-5-MaxIOI'),('Day-6-MaxIOI'),\
                     ('Day-7-MaxIOI'),('Day-8-MaxIOI'),\
                     ('Day-9-MaxIOI')])
```

7.1.11 Export Final Data Set

[15]:
```
#exmport data set
with pd.ExcelWriter('2G_IOI_KPIs.xlsx') as writer:
    df12.to_excel(writer,sheet_name="2G_IOI_KPIs_Summary",\
                  engine='openpyxl',\
                  na_rep='N/A',index=False,float_format='%.2f')
```

[16]:
```
df_2g_da.Date.unique()
```

[16]:
```
array(['2021-06-03T00:00:00.000000000', '2021-06-04T00:00:00.000000000',
       '2021-06-05T00:00:00.000000000', '2021-06-06T00:00:00.000000000',
       '2021-06-07T00:00:00.000000000', '2021-06-08T00:00:00.000000000',
       '2021-06-09T00:00:00.000000000', '2021-06-10T00:00:00.000000000',
       '2021-06-11T00:00:00.000000000', '2021-06-12T00:00:00.000000000'],
      dtype='datetime64[ns]')
```

[17]:
```
df_2g_hr.Date.unique()
```

```
[17]: array(['2021-06-03T00:00:00.000000000', '2021-06-04T00:00:00.000000000',
              '2021-06-05T00:00:00.000000000', '2021-06-06T00:00:00.000000000',
              '2021-06-07T00:00:00.000000000', '2021-06-08T00:00:00.000000000',
              '2021-06-09T00:00:00.000000000', '2021-06-10T00:00:00.000000000',
              '2021-06-11T00:00:00.000000000', '2021-06-12T00:00:00.000000000'],
             dtype='datetime64[ns]')
```

7.1.12 Output File Format

- Output File Format

8. UMTS RTWP KPI

8.1 UMT RTWP KPI Summary

8.1.1 Input File Format

- UMTs External Interference DA and Hourly KPIs

8.1.2 Import required Libraries

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
from glob import glob
```

8.1.3 Set Working Path For 3G(DA) RTWP KPIs

[2]:
```
folder_path = 'D:/DataSets/KPIs/IOI_KPIs/3G/DA'
os.chdir(folder_path)
```

8.1.4 3G DA - RTWP and CalculatevAverage

[3]:
```
da = sorted(glob('*.csv'))
df_3g_da=pd.concat((pd.read_csv(file,header=3,\
    skipfooter=1,engine='python',na_values=['NIL','/0'],\
    parse_dates=["Date"]) for file in da)).\
    sort_values('Date').set_index(['Date']).\
    last('10D').reset_index()
```

[4]:
```
# calucalate average
df_3g_da_avg=df_3g_da.groupby(['RNC','Cell ID']).\
    apply(lambda x: np.average(x['VS.MeanRTWP(dBm)']))\
    .reset_index(name='3G DA IoI')
```

8.1.5 Set Working Path For 3G(Hourly) RTWP KPIs

[5]:
```
folder_path = 'D:/DataSets/KPIs/IOI_KPIs/3G/Hourly'
os.chdir(folder_path)
```

8.1.6 3G Hourly - RTWP :Calculate Max RTWP Per Cell Per Day

[6]:
```
bh= sorted(glob('*.csv'))
df_3g_hr=pd.concat((pd.read_csv(file,header=3,\
        skipfooter=1,engine='python',na_values=['NIL','/0'],\
        parse_dates=["Date"]) for file in bh)).\
        fillna(-6666).sort_values('Date').\
        set_index(['Date']).last('10D').reset_index()
```

[7]:
```
# assign Date to Date Number as category
df_3g_hr['Day']='Day'+'-'+df_3g_hr.Date.astype("category").\
                    cat.codes.astype(str)+'-'+'MaxIOI'
```

[8]:
```
# identdtify the max interference index
df_3g_hr_max = df_3g_hr.loc[df_3g_hr.\
        groupby(['Date','RNC','Cell ID','Cell Name'])\
        'VS.MeanRTWP(dBm)'].idxmax()]
```

[9]:
```
# re-shapre data frame
df_3g_hr_rs= df_3g_hr_max.pivot_table(index=['RNC','Cell ID'],\
     columns="Day",values='VS.MeanRTWP(dBm)').reset_index()
```

8.1.7 3G Hourly - RTWP :Calculate Number of Interval RTWP >=-95(U900) and >=98(U2100)

[10]:
```
# re-shapre (for u900)
df_3g_hr_gt_n95_interval_u900=df_3g_hr.\
        groupby(['RNC','Cell ID','Cell Name'])['VS.MeanRTWP(dBm)'].\
        apply(lambda x: (x.ge(-95)).sum()).\
        reset_index(name='Total Interval IOI>=-95')
```

[11]:
```
# re-shapre (for u2100)
df_3g_hr_gt_n98_interval_u2100=df_3g_hr.\
        groupby(['RNC','Cell ID','Cell Name'])['VS.MeanRTWP(dBm)'].\
        apply(lambda x: (x.ge(-98)).sum()).\
        reset_index(name='Total Interval IOI>=-98')
```

8.1.8 3G RTWP - Final Data Set

[12]:
```
# IOI stats Output
df_3g_fds = df_3g_da_avg.\
    merge(df_3g_hr_gt_n95_interval_u900,on=['RNC','Cell ID'],how='left').\
    merge(df_3g_hr_gt_n98_interval_u2100,on=['RNC','Cell ID'],how='left').\
    merge(df_3g_hr_rs,on=['RNC','Cell ID'],how='left')
```

8.1.9 Set Working Path For Output

[13]:
```
folder_path = 'D:/DataSets/KPIs/IOI_KPIs/3G'
os.chdir(folder_path)
```

8.1.10 Formatting Data Set

[14]:
```
# conditional colour
df12= df_3g_fds.style.applymap\
            (lambda x: 'color: black' if pd.isnull(x) \
             else 'background-color: %s' % 'red' \
             if x>=-98 else 'background-color: %s' % 'green',\
             subset=[('3G DA IoI'),('Day-0-MaxIOI'),\
                     ('Day-1-MaxIOI'),('Day-2-MaxIOI'),\
                     ('Day-3-MaxIOI'),('Day-4-MaxIOI'),\
                     ('Day-5-MaxIOI'),('Day-6-MaxIOI'),\
                     ('Day-7-MaxIOI'),('Day-8-MaxIOI'),\
                     ('Day-9-MaxIOI')])
```

8.1.11 Export Final Data Set

[15]:
```
#export data set
with pd.ExcelWriter('3G_IOI_KPIs.xlsx') as writer:
    df12.to_excel(writer,sheet_name="3G_IOI_KPIs_Summary",\
                engine='openpyxl',na_rep='N/A',\
                  index=False,float_format='%.2f')
```

[16]:
```
df_3g_da.Date.unique()
```

[16]:
```
array(['2021-06-03T00:00:00.000000000', '2021-06-04T00:00:00.000000000',
       '2021-06-05T00:00:00.000000000', '2021-06-06T00:00:00.000000000',
       '2021-06-07T00:00:00.000000000', '2021-06-08T00:00:00.000000000',
       '2021-06-09T00:00:00.000000000', '2021-06-10T00:00:00.000000000',
       '2021-06-11T00:00:00.000000000', '2021-06-12T00:00:00.000000000'],
      dtype='datetime64[ns]')
```

[17]:
```
df_3g_hr.Date.unique()
```

[17]:
```
array(['2021-06-03T00:00:00.000000000', '2021-06-04T00:00:00.000000000',
       '2021-06-05T00:00:00.000000000', '2021-06-06T00:00:00.000000000',
       '2021-06-07T00:00:00.000000000', '2021-06-08T00:00:00.000000000',
       '2021-06-09T00:00:00.000000000', '2021-06-10T00:00:00.000000000',
       '2021-06-11T00:00:00.000000000', '2021-06-12T00:00:00.000000000'],
      dtype='datetime64[ns]')
```

8.1.12 Output File Format

- Output File Format

9. LTE UL Interference KPI

9.1 LTE UL Interference KPI Summary

9.1.1 Input File Format

- LTE External Interference DA and Hourly KPIs

9.1.2 Import required Libraries

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
from glob import glob
```

9.1.3 Set Working Path For 4G(DA) UL Interference KPIs

[2]:
```
folder_path = 'D:/DataSets/KPIs/IOI_KPIs/4G/DA'
os.chdir(folder_path)
```

9.1.4 4G DA - UL Interference and calculate Average

[3]:
```
da = sorted(glob('*.csv'))
df_4g_da=pd.concat((pd.read_csv(file,header=3,\
        skipfooter=1,engine='python',na_values=['NIL','/0'],\
        parse_dates=["Date"]) for file in da)).\
        sort_values('Date').set_index(['Date']).\
        last('10D').reset_index()
```

[4]:
```
# calucalate average
df_4g_da_avg=df_4g_da.groupby(['Cell Name']).\
            apply(lambda x: np.average\
            (x['Average UL Interference per Non Shared PRB for GL6MHz (dBm)']))\
            .reset_index(name='4G DA IoI')
```

9.1.5 Set Working Path For 4G(Hourly) UL Interference KPIs

[5]:
```
folder_path = 'D:/DataSets/KPIs/IOI_KPIs/4G/Hourly'
os.chdir(folder_path)
```

9.1.6 4G Hourly - Interference :Calculate Max UL Interference Per Cell Per Day

[6]:
```
bh= sorted(glob('*.csv'))
df_4g_hr=pd.concat((pd.read_csv(file,header=3,\
        skipfooter=1,engine='python',na_values=['NIL','/0'],\
        parse_dates=["Date"]) for file in bh)).\
        fillna(-6666).sort_values('Date').\
        set_index(['Date']).last('10D').reset_index()
```

[7]:
```
# assign Date to Date Number as category
df_4g_hr['Day']='Day'+'-'+df_4g_hr.Date.astype("category").\
                cat.codes.astype(str)+'-'+'MaxIOI'
```

[8]:
```
# idendtify the max interference index
df_4g_hr_max = df_4g_hr.loc[df_4g_hr.groupby(['Date','Cell Name'])\
        ['Average UL Interference per Non Shared PRB for GL6MHz (dBm)']\
                            .idxmax()]
```

[9]:
```
# re-shapre data frame
df_4g_hr_rs= df_4g_hr_max.pivot_table(index=['Cell Name'],\
    columns="Day",\
    values='Average UL Interference per Non Shared PRB for GL6MHz (dBm)')\
    .reset_index()
```

9.1.7 4G Hourly - Interference : Calculate Number of Interval UL Interference >=-108

[10]:
```
# re-shapre
df_4g_hr_gt_n108_interval=df_4g_hr.groupby(['Cell Name'])\
        ['Average UL Interference per Non Shared PRB for GL6MHz (dBm)'].\
        apply(lambda x: (x.ge(-108)).sum()).\
        reset_index(name='Total Interval IOI>-100')
```

9.1.8 4G Interference - Final Data Set

[11]:
```
# IOI stats Output
df_4g_fds = df_4g_da_avg.\
    merge(df_4g_hr_gt_n108_interval,on=['Cell Name'],how='left').\
    merge(df_4g_hr_rs,on=['Cell Name'],how='left')
```

[12]:
```
#row count
df_4g_fds["#of Day IOI>-108"] = df_4g_fds.\
        where(df_4g_fds.iloc[:,3:] >= -108)\
        .count(1)
```

9.1.9 Set Working Path For Output

[13]:
```
folder_path = 'D:/DataSets/KPIs/IOI_KPIs/4G'
os.chdir(folder_path)
```

9.1.10 Formatting Data Set

[14]:
```
# conditional colour
df12= df_4g_fds.style.applymap\
            (lambda x: 'color: black' if pd.isnull(x) \
             else 'background-color: %s' % 'red' \
             if x>=-98 else 'background-color: %s' % 'green',\
             subset=[('4G DA IoI'),('Day-0-MaxIOI'),\
                     ('Day-1-MaxIOI'),('Day-2-MaxIOI'),\
                     ('Day-3-MaxIOI'),('Day-4-MaxIOI'),\
                     ('Day-5-MaxIOI'),('Day-6-MaxIOI'),\
                     ('Day-7-MaxIOI'),('Day-8-MaxIOI'),\
                     ('Day-9-MaxIOI')])
```

9.1.11 Export Final Data Set

[15]:
```
#export data set
with pd.ExcelWriter('4G_IOI_KPIs.xlsx') as writer:
    df12.to_excel(writer,sheet_name="4G_IOI_KPIs_Summary",\
            engine='openpyxl',na_rep='N/A',\
            index=False,float_format='%.2f')
```

[16]:
```
df_4g_da.Date.unique()
```

[16]:
```
array(['2021-06-03T00:00:00.000000000', '2021-06-04T00:00:00.000000000',
       '2021-06-05T00:00:00.000000000', '2021-06-06T00:00:00.000000000',
       '2021-06-07T00:00:00.000000000', '2021-06-08T00:00:00.000000000',
       '2021-06-09T00:00:00.000000000', '2021-06-10T00:00:00.000000000',
       '2021-06-11T00:00:00.000000000', '2021-06-12T00:00:00.000000000'],
      dtype='datetime64[ns]')
```

[17]:
```
df_4g_hr.Date.unique()
```

[17]:
```
array(['2021-06-03T00:00:00.000000000', '2021-06-04T00:00:00.000000000',
       '2021-06-05T00:00:00.000000000', '2021-06-06T00:00:00.000000000',
       '2021-06-07T00:00:00.000000000', '2021-06-08T00:00:00.000000000',
       '2021-06-09T00:00:00.000000000', '2021-06-10T00:00:00.000000000',
       '2021-06-11T00:00:00.000000000', '2021-06-12T00:00:00.000000000'],
      dtype='datetime64[ns]')
```

9.1.12 Output File Format

- Output File Format

10. GSM BSS Issues

10.1 BSS Drops and TCH Availability Rate

10.1.1 Input File Format

- BSS Issues DA KPIs

10.1.2 Import required Libraries

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
from glob import glob
```

10.1.3 working path

[2]:
```
working_directory = 'D:/DataSets/KPIs/2G_BSS_Issues'
os.chdir(working_directory)
```

10.1.4 Unzip the Files

[3]:
```
for file in os.listdir(working_directory):   # get the list of files
    if zipfile.is_zipfile(file): # if it is a zipfile, extract it
        with zipfile.ZipFile(file) as item: # treat the file as a zip
           item.extractall()  # extract it in the working directory
```

10.1.5 Import BSS Data

[4]:
```
bss_files = sorted(glob('*.csv'))
# concat all the Cluster DA Files
cell_da=pd.concat((pd.read_csv(file,skiprows=[0,1,2,3,4],\
    skipfooter=1,engine='python',
    usecols=['Date','Cell CI','GBSC','TCH Availability Rate',\
'CM333:Call Drops due to Abis Terrestrial Link Failure (Traffic Channel)',\
'CM334:Call Drops due to Equipment Failure (Traffic Channel)','Cell Name'],\
    parse_dates=["Date"],na_values=['NIL','/0']) for file in bss_files))\
    .sort_values('Date').set_index(['Date']).last('5D').reset_index()
```

10.1.6 Delete csv Files

[5]:
```
for filename in os.listdir(working_directory):
    if filename.endswith('.csv'):
        os.unlink(os.path.join(working_directory, filename))
```

10.1.7 Calculate BSS Drops

[6]:
```
cell_da['BSS_Drops']= cell_da['CM333:Call Drops due to Abis Terrestrial Link␣
 ↪Failure (Traffic Channel)']\
+cell_da['CM334:Call Drops due to Equipment Failure (Traffic Channel)']
```

10.1.8 Select Requied Columns

[7]:
```
cell_da = cell_da\
        [['Date', 'Cell CI', 'GBSC',\
        'TCH Availability Rate','BSS_Drops',\
        'Cell Name']]
```

10.1.9 pivot_table design

[8]:
```
df5=cell_da.pivot_table\
    (index=["GBSC",'Cell CI','Cell Name'],\
    columns="Date").reset_index()
```

10.1.10 Count of Days TCH Availability Rate <100

[9]:
```
df5["count"] = df5.where(df5.iloc[:,8:] <100).count(1)
```

10.1.11 Export Final Data Set

[10]:
```
df5.to_excel('BSS_Issues.xlsx',sheet_name='BSS Issues-Center Region')
```

[11]:
```
cell_da.Date.unique()
```

[11]:
```
array(['2021-06-09T00:00:00.000000000', '2021-06-10T00:00:00.000000000',
       '2021-06-11T00:00:00.000000000', '2021-06-12T00:00:00.000000000',
       '2021-06-13T00:00:00.000000000'], dtype='datetime64[ns]')
```

10.1.12 Output File Format

- Output File Format

11. Calculate Cluster Busy Hour

Calculate Cluster Busy Hour

11.1 Case-1: If Date and Time in Seprate Column

11.1.1 Input File Format

- Cluster Hourly KPIs

11.1.2 Import required Libraries

[1]:
```
import os
import numpy as np
import pandas as pd
from glob import glob
```

11.1.3 Set Working Path

[2]:
```
working_directory = 'D:/DataSets/KPIs/Calculate_BH/Date_Time_Diff'
os.chdir(working_directory)
```

11.1.4 Import Files Date and Time in Different Column

[3]:
```
dtd = sorted(glob('*.csv'))
df_d=pd.concat((pd.read_csv(file,header=3,\
    skipfooter=1,engine='python',na_values=['NIL','/0'],\
    parse_dates=["Date"]) for file in dtd)).sort_values('Date')
```

11.1.5 Filter Max Traffic with groupby

[4]:
```
df_d_bh=df_d.loc[df_d.groupby(['Date','GCell Group'])\
                 ['GlobelTraffic'].idxmax()]
```

11.1.6 Export

[5]:
```
df_d_bh.to_csv('cluster_bh.csv',index=False)
```

11.2 Case-2: If Date and Time in Same Column

11.2.1 Input File Format

- Cluster Hourly KPIs

11.2.2 Set Working Path

[6]:
```
working_directory = 'D:/DataSets/KPIs/Calculate_BH/Date_Time_Same'
os.chdir(working_directory)
```

11.2.3 Import Files Date and Time in same Column

[7]:
```
dts = sorted(glob('*.csv'))
df_s=pd.concat((pd.read_csv(file,header=3,\
    skipfooter=1,engine='python',na_values=['NIL','/0'],\
    parse_dates=["Time"]) for file in dts)).sort_values('Time')
```

11.2.4 Date and Time split in Different Columns

[8]:
```
df_s['Date']= pd.to_datetime(df_s['Time']).dt.date
df_s['Tim']= pd.to_datetime(df_s['Time']).dt.time
```

11.2.5 Calculate Cluster BH

[9]:
```
df_d_bh=df_s.loc[df_s.groupby(['Date','GCell Group'])\
                 ['GlobelTraffic'].idxmax()]
```

11.2.6 Remove Unwanted Columns

[10]:
```
df_d_bh=df_d_bh.iloc[:,:-2]
```

11.2.7 Export Final Data Set

[11]:
```
df_d_bh.to_csv('cluster_bh.csv',index=False)
```

11.2.8 Output File Format

- Output File Format

12. Daily SLA Target Identification

12.1 Daily Conformance

12.1.1 Input File Format

- Daily Conformance Working Input Files

12.1.2 Import required Libraries

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
from glob import glob
from collections import ChainMap
```

12.1.3 Set Working Path

[2]:
```
folder_path = 'D:/DataSets/Conformance/Quaterly Conformance Working'
os.chdir(folder_path)
```

12.1.4 Unzip the Cluster BH KPIs

[3]:
```
for file in os.listdir(folder_path):    # get the list of files
    if zipfile.is_zipfile(file): # if it is a zipfile, extract it
        with zipfile.ZipFile(file) as item: # treat the file as a zip
           item.extractall()  # extract it in the working directory
```

12.1.5 Import Cluster BH KPIs

[4]:
```
busy_hour_files = sorted(glob('Umer Saeed_Cluster_BH*.csv'))
# concat all the Cluster DA Files
cluster_bh=pd.concat((pd.read_csv(file,skiprows=[0,1,2,3,4],\
        skipfooter=1,engine='python',\
        usecols=['Date','Time','GCell Group',\
        'CSSR_Non Blocking','HSR (Incoming & Outgoing)',\
        'DCR','RxQual Index DL(%)',\
        'RxQual Index UL(%)','GOS-SDCCH(%)',\
        'CallSetup TCH GOS(%)','Mobility TCH GOS(%)'],\
        parse_dates=["Date"],na_values=['NIL','/0'])\
        for file in busy_hour_files)).sort_values('Date').\
        set_index(['Date']).last('1Q')
```

12.1.6 Select only Clusters

[5]:
```
# select only cluster , remove city, region, sub region
cluster_bh=cluster_bh[cluster_bh['GCell Group'].\
    str.contains('|'.join(['_Rural','_Urban']))].\
    reset_index()
```

12.1.7 Modification Cluster Name as per requirement

[6]:
```
# Cluster is Urban or Rural, get from GCell Group
cluster_bh['Cluster Type'] = cluster_bh['GCell Group'].str.strip().str[-5:]
# Remove Urban and Rural from GCell Group
cluster_bh['Location']=cluster_bh['GCell Group'].map(lambda x: str(x)[:-6])
```

12.1.8 Sub Region Defination

[7]:
```
d = ChainMap(dict.fromkeys(['GUJRANWALA_CLUSTER_01_Rural',
                            'GUJRANWALA_CLUSTER_01_Urban',
'GUJRANWALA_CLUSTER_02_Rural','GUJRANWALA_CLUSTER_02_Urban',
'GUJRANWALA_CLUSTER_03_Rural','GUJRANWALA_CLUSTER_03_Urban',
'GUJRANWALA_CLUSTER_04_Rural','GUJRANWALA_CLUSTER_04_Urban',
'GUJRANWALA_CLUSTER_05_Rural','GUJRANWALA_CLUSTER_05_Urban',
'GUJRANWALA_CLUSTER_06_Rural','KASUR_CLUSTER_01_Rural',
'KASUR_CLUSTER_02_Rural','KASUR_CLUSTER_03_Rural',
'KASUR_CLUSTER_03_Urban','LAHORE_CLUSTER_01_Rural',
'LAHORE_CLUSTER_01_Urban',
'LAHORE_CLUSTER_02_Rural','LAHORE_CLUSTER_02_Urban',
'LAHORE_CLUSTER_03_Rural','LAHORE_CLUSTER_03_Urban',
'LAHORE_CLUSTER_04_Urban','LAHORE_CLUSTER_05_Rural',
'LAHORE_CLUSTER_05_Urban',
'LAHORE_CLUSTER_06_Rural','LAHORE_CLUSTER_06_Urban',
```

```
'LAHORE_CLUSTER_07_Rural','LAHORE_CLUSTER_07_Urban',
'LAHORE_CLUSTER_08_Rural','LAHORE_CLUSTER_08_Urban',
'LAHORE_CLUSTER_09_Rural','LAHORE_CLUSTER_09_Urban',
'LAHORE_CLUSTER_10_Urban','LAHORE_CLUSTER_11_Rural',
'LAHORE_CLUSTER_11_Urban','LAHORE_CLUSTER_12_Urban',
'LAHORE_CLUSTER_13_Urban','LAHORE_CLUSTER_14_Urban',
'SIALKOT_CLUSTER_01_Rural','SIALKOT_CLUSTER_01_Urban',
'SIALKOT_CLUSTER_02_Rural','SIALKOT_CLUSTER_02_Urban',
'SIALKOT_CLUSTER_03_Rural','SIALKOT_CLUSTER_03_Urban',
'SIALKOT_CLUSTER_04_Rural','SIALKOT_CLUSTER_05_Rural',
'SIALKOT_CLUSTER_05_Urban','SIALKOT_CLUSTER_06_Rural',
'SIALKOT_CLUSTER_06_Urban','SIALKOT_CLUSTER_07_Rural',
'SIALKOT_CLUSTER_07_Urban'], 'Center-1'),
dict.fromkeys(['DG_KHAN_CLUSTER_01_Rural',
'DG_KHAN_CLUSTER_02_Rural','DG_KHAN_CLUSTER_02_Urban',
'DI_KHAN_CLUSTER_01_Rural','DI_KHAN_CLUSTER_01_Urban',
'DI_KHAN_CLUSTER_02_Rural',
'DI_KHAN_CLUSTER_02_Urban','DI_KHAN_CLUSTER_03_Rural',
'FAISALABAD_CLUSTER_01_Rural',
'FAISALABAD_CLUSTER_02_Rural','FAISALABAD_CLUSTER_03_Rural',
'FAISALABAD_CLUSTER_04_Rural',
'FAISALABAD_CLUSTER_04_Urban','FAISALABAD_CLUSTER_05_Rural',
'FAISALABAD_CLUSTER_05_Urban',
'FAISALABAD_CLUSTER_06_Rural','FAISALABAD_CLUSTER_06_Urban',
'JHUNG_CLUSTER_01_Rural',
'JHUNG_CLUSTER_01_Urban','JHUNG_CLUSTER_02_Rural',
'JHUNG_CLUSTER_02_Urban',
'JHUNG_CLUSTER_03_Rural','JHUNG_CLUSTER_03_Urban',
'JHUNG_CLUSTER_04_Rural',
'JHUNG_CLUSTER_04_Urban','JHUNG_CLUSTER_05_Rural',
'JHUNG_CLUSTER_05_Urban',
'SAHIWAL_CLUSTER_01_Rural','SAHIWAL_CLUSTER_01_Urban',
'SAHIWAL_CLUSTER_02_Rural',
'SAHIWAL_CLUSTER_02_Urban'], 'Center-2'),
dict.fromkeys(['JAMPUR_CLUSTER_01_Urban',
'RAJANPUR_CLUSTER_01_Rural','RAJANPUR_CLUSTER_01_Urban',
'JAMPUR_CLUSTER_01_Rural','DG_KHAN_CLUSTER_03_Rural',
               'DG_KHAN_CLUSTER_03_Urban',
'DG_KHAN_CLUSTER_04_Rural','DG_KHAN_CLUSTER_04_Urban',
               'SAHIWAL_CLUSTER_03_Rural',
'SAHIWAL_CLUSTER_03_Urban','KHANPUR_CLUSTER_01_Rural',
               'KHANPUR_CLUSTER_01_Urban',
'RAHIMYARKHAN_CLUSTER_01_Rural','RAHIMYARKHAN_CLUSTER_01_Urban',
'AHMEDPUREAST_CLUSTER_01_Rural','AHMEDPUREAST_CLUSTER_01_Urban',
'ALIPUR_CLUSTER_01_Rural','ALIPUR_CLUSTER_01_Urban',
'BAHAWALPUR_CLUSTER_01_Rural','BAHAWALPUR_CLUSTER_01_Urban',
'BAHAWALPUR_CLUSTER_02_Rural','SAHIWAL_CLUSTER_04_Rural',
```

```
'SAHIWAL_CLUSTER_04_Urban','MULTAN_CLUSTER_01_Rural',
'MULTAN_CLUSTER_01_Urban','MULTAN_CLUSTER_02_Rural',
               'MULTAN_CLUSTER_02_Urban',
'MULTAN_CLUSTER_03_Rural','MULTAN_CLUSTER_03_Urban',
               'RYK DESERT_Cluster_Rural',
'SADIQABAD_CLUSTER_01_Rural','SAHIWAL_CLUSTER_05_Rural',
               'SAHIWAL_CLUSTER_05_Urban',
'SAHIWAL_CLUSTER_06_Rural','SAHIWAL_CLUSTER_06_Urban',
               'SAHIWAL_CLUSTER_07_Rural',
'SAHIWAL_CLUSTER_07_Urban'], 'Center-3'))

cluster_bh['Region'] = cluster_bh['GCell Group'].map(d.get)
```

12.1.9 Data Re-shape

[8]:
```
qformat=cluster_bh.pivot(index=['Date','Region','Location'],\
        columns='Cluster Type',\
        values=['CSSR_Non Blocking',\
        'HSR (Incoming & Outgoing)', 'DCR', 'GOS-SDCCH(%)',\
        'CallSetup TCH GOS(%)', 'Mobility TCH GOS(%)', 'RxQual Index DL(%)',\
        'RxQual Index UL(%)']).\
         fillna('N/A').\
        sort_index(level=[0,1],\
        axis=1,ascending=[True,False])
```

[9]:
```
qformat=qformat.reset_index()
```

[10]:
```
# Export
qformat.to_excel("SLA Target.xlsx",engine='openpyxl',na_rep='N/A')
```

[11]:
```
#import
aa=pd.read_excel('SLA Target.xlsx',header=[0,1])
```

12.1.10 Formatting and Export Final Data Set

[12]:
```
bb=aa.style\
.applymap(lambda x: 'color: black' if pd.isnull(x) else
          'background-color: %s' % 'green'
          if x>=99.50 else 'background-color: %s' % 'red'
          ,subset=[('CSSR_Non Blocking','Urban')])\
.applymap(lambda x: 'color: black' if pd.isnull(x) else
          'background-color: %s' % 'green'
          if x>=99.00 else 'background-color: %s' % 'red',
          subset=[('CSSR_Non Blocking','Rural')])\
.applymap(lambda x: 'color: black' if pd.isnull(x)
          else 'background-color: %s' % 'green'
```

```
          if x<=0.60  else 'background-color: %s' % 'red'\
          ,subset=[('DCR','Urban')])\
.applymap(lambda x: 'color: black' if pd.isnull(x) else
          'background-color: %s' % 'green'
          if x<=1.00  else 'background-color: %s' % 'red'
          ,subset=[('DCR','Rural')])\
.applymap(lambda x: 'color: black' if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x>=97.50 else 'background-color: %s' % 'red'
          ,subset=[('HSR (Incoming & Outgoing)','Urban')])\
.applymap(lambda x: 'color: black' if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x>=96.00 else 'background-color: %s' % 'red'
          ,subset=[('HSR (Incoming & Outgoing)','Rural')])\
.applymap(lambda x: 'color: black' if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x<=0.10  else 'background-color: %s' % 'red'
          ,subset=[('GOS-SDCCH(%)','Urban')])\
.applymap(lambda x: 'color: black' if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x<=0.10  else 'background-color: %s' % 'red'
          ,subset=[('GOS-SDCCH(%)','Rural')])\
.applymap(lambda x: 'color: black' if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x<=2.00  else 'background-color: %s' % 'red'
          ,subset=[('CallSetup TCH GOS(%)','Urban')])\
.applymap(lambda x: 'color: black' if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x<=2.00  else 'background-color: %s' % 'red'
          ,subset=[('CallSetup TCH GOS(%)','Rural')])\
.applymap(lambda x: 'color: black' if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x<=4.00  else 'background-color: %s' % 'red'
          ,subset=[('Mobility TCH GOS(%)','Urban')])\
.applymap(lambda x: 'color: black' if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x<=4.00  else 'background-color: %s' % 'red'
          ,subset=[('Mobility TCH GOS(%)','Rural')])\
.applymap(lambda x: 'color: black' if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x>=98.40 else 'background-color: %s' % 'red'
          ,subset=[('RxQual Index DL(%)','Urban')])\
.applymap(lambda x: 'color: black' if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x>=97.00 else 'background-color: %s' % 'red'
          ,subset=[('RxQual Index DL(%)','Rural')])\
.applymap(lambda x: 'color: black' if pd.isnull(x)
```

```
          else 'background-color: %s' % 'green'
          if x>=98.20 else 'background-color: %s' % 'red'
          ,subset=[('RxQual Index UL(%)','Urban')])\
.applymap(lambda x: 'color: black' if pd.isnull(x) else
          'background-color: %s' % 'green'
          if x>=97.70 else 'background-color: %s' % 'red'
          ,subset=[('RxQual Index UL(%)','Rural')])\
.to_excel('SLA Target.xlsx',engine='openpyxl',na_rep='N/A')
```

12.1.11 Delete Excel File

[13]:
```
for filename in os.listdir(folder_path):
    if filename.endswith('.csv'):
        os.unlink(os.path.join(folder_path, filename))
```

12.1.12 Output File Format

- Output File Format

13. Quarterly SLA Target Identification

13.1 Quarterly Conformance

13.1.1 Input File Format

- Quarterly Conformance Working Input Files

13.1.2 Import required Libraries

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
from glob import glob
from collections import ChainMap
```

13.1.3 Set Working Path

[2]:
```
folder_path = 'D:/DataSets/Conformance/Quaterly Conformance Working'
os.chdir(folder_path)
```

13.1.4 Unzip the Cluster BH KPIs

[3]:
```
for file in os.listdir(folder_path):   # get the list of files
    if zipfile.is_zipfile(file): # if it is a zipfile, extract it
        with zipfile.ZipFile(file) as item: # treat the file as a zip
           item.extractall()  # extract it in the working directory
```

13.1.5 Import Cluster BH Counters

[4]:
```
busy_hour_files = sorted(glob('Umer Saeed_Cluster_BH*.csv'))
# concat all the Cluster DA Files
cluster_bh=pd.concat((pd.read_csv(file,skiprows=[0,1,2,3,4],\
        skipfooter=1,engine='python',\
        usecols=['Date','Time','GCell Group',\
        '_CallSetup TCH GOS(%)_D','_CallSetup TCH GOS(%)_N',\
        '_GOS-SDCCH(%)_D','_GOS-SDCCH(%)_N',\
        '_Mobility TCH GOS(%)_D','_Mobility TCH GOS(%)_N',\
        '_DCR_D','_DCR_N',\
        '_RxQual Index DL_1','_RxQual Index DL_2',\
        '_RxQual Index UL_1','_RxQual Index UL_2',\
        '_HSR%_D','_HSR%_N',\
        'CSSR_Non Blocking_1_N','CSSR_Non Blocking_1_D',\
        'CSSR_Non Blocking_2_N','CSSR_Non Blocking_2_D'],\
        parse_dates=["Date"]) for file in busy_hour_files)).\
        sort_values('Date')
```

13.1.6 Select only Clusters

[5]:
```
# select only cluster , remove city, region, sub region
cluster_bh=cluster_bh[cluster_bh['GCell Group']\
        .str.contains('|'.join(['_Rural','_Urban']))]
```

13.1.7 Sub Region Defination

[6]:
```
d = ChainMap(dict.fromkeys(['GUJRANWALA_CLUSTER_01_Rural',
                            'GUJRANWALA_CLUSTER_01_Urban',
'GUJRANWALA_CLUSTER_02_Rural','GUJRANWALA_CLUSTER_02_Urban',
'GUJRANWALA_CLUSTER_03_Rural','GUJRANWALA_CLUSTER_03_Urban',
'GUJRANWALA_CLUSTER_04_Rural','GUJRANWALA_CLUSTER_04_Urban',
'GUJRANWALA_CLUSTER_05_Rural','GUJRANWALA_CLUSTER_05_Urban',
'GUJRANWALA_CLUSTER_06_Rural','KASUR_CLUSTER_01_Rural',
'KASUR_CLUSTER_02_Rural','KASUR_CLUSTER_03_Rural',
'KASUR_CLUSTER_03_Urban','LAHORE_CLUSTER_01_Rural',
                            'LAHORE_CLUSTER_01_Urban',
'LAHORE_CLUSTER_02_Rural','LAHORE_CLUSTER_02_Urban',
'LAHORE_CLUSTER_03_Rural','LAHORE_CLUSTER_03_Urban',
'LAHORE_CLUSTER_04_Urban','LAHORE_CLUSTER_05_Rural',
                            'LAHORE_CLUSTER_05_Urban',
```

```
'LAHORE_CLUSTER_06_Rural','LAHORE_CLUSTER_06_Urban',
'LAHORE_CLUSTER_07_Rural','LAHORE_CLUSTER_07_Urban',
'LAHORE_CLUSTER_08_Rural','LAHORE_CLUSTER_08_Urban',
'LAHORE_CLUSTER_09_Rural','LAHORE_CLUSTER_09_Urban',
'LAHORE_CLUSTER_10_Urban','LAHORE_CLUSTER_11_Rural',
'LAHORE_CLUSTER_11_Urban','LAHORE_CLUSTER_12_Urban',
'LAHORE_CLUSTER_13_Urban','LAHORE_CLUSTER_14_Urban',
'SIALKOT_CLUSTER_01_Rural','SIALKOT_CLUSTER_01_Urban',
'SIALKOT_CLUSTER_02_Rural','SIALKOT_CLUSTER_02_Urban',
'SIALKOT_CLUSTER_03_Rural','SIALKOT_CLUSTER_03_Urban',
'SIALKOT_CLUSTER_04_Rural','SIALKOT_CLUSTER_05_Rural',
'SIALKOT_CLUSTER_05_Urban','SIALKOT_CLUSTER_06_Rural',
'SIALKOT_CLUSTER_06_Urban','SIALKOT_CLUSTER_07_Rural',
'SIALKOT_CLUSTER_07_Urban'], 'Center-1'),
dict.fromkeys(['DG_KHAN_CLUSTER_01_Rural',
'DG_KHAN_CLUSTER_02_Rural','DG_KHAN_CLUSTER_02_Urban',
'DI_KHAN_CLUSTER_01_Rural','DI_KHAN_CLUSTER_01_Urban',
               'DI_KHAN_CLUSTER_02_Rural',
'DI_KHAN_CLUSTER_02_Urban','DI_KHAN_CLUSTER_03_Rural',
               'FAISALABAD_CLUSTER_01_Rural',
'FAISALABAD_CLUSTER_02_Rural','FAISALABAD_CLUSTER_03_Rural',
               'FAISALABAD_CLUSTER_04_Rural',
'FAISALABAD_CLUSTER_04_Urban','FAISALABAD_CLUSTER_05_Rural',
               'FAISALABAD_CLUSTER_05_Urban',
'FAISALABAD_CLUSTER_06_Rural','FAISALABAD_CLUSTER_06_Urban',
               'JHUNG_CLUSTER_01_Rural',
'JHUNG_CLUSTER_01_Urban','JHUNG_CLUSTER_02_Rural',
               'JHUNG_CLUSTER_02_Urban',
'JHUNG_CLUSTER_03_Rural','JHUNG_CLUSTER_03_Urban',
               'JHUNG_CLUSTER_04_Rural',
'JHUNG_CLUSTER_04_Urban','JHUNG_CLUSTER_05_Rural',
               'JHUNG_CLUSTER_05_Urban',
'SAHIWAL_CLUSTER_01_Rural','SAHIWAL_CLUSTER_01_Urban',
               'SAHIWAL_CLUSTER_02_Rural',
'SAHIWAL_CLUSTER_02_Urban'], 'Center-2'),
dict.fromkeys(['JAMPUR_CLUSTER_01_Urban','RAJANPUR_CLUSTER_01_Rural',
               'RAJANPUR_CLUSTER_01_Urban',
'JAMPUR_CLUSTER_01_Rural','DG_KHAN_CLUSTER_03_Rural',
               'DG_KHAN_CLUSTER_03_Urban',
'DG_KHAN_CLUSTER_04_Rural','DG_KHAN_CLUSTER_04_Urban',
               'SAHIWAL_CLUSTER_03_Rural',
'SAHIWAL_CLUSTER_03_Urban','KHANPUR_CLUSTER_01_Rural',
               'KHANPUR_CLUSTER_01_Urban',
'RAHIMYARKHAN_CLUSTER_01_Rural','RAHIMYARKHAN_CLUSTER_01_Urban',
'AHMEDPUREAST_CLUSTER_01_Rural','AHMEDPUREAST_CLUSTER_01_Urban',
'ALIPUR_CLUSTER_01_Rural','ALIPUR_CLUSTER_01_Urban',
'BAHAWALPUR_CLUSTER_01_Rural','BAHAWALPUR_CLUSTER_01_Urban',
```

```
'BAHAWALPUR_CLUSTER_02_Rural','SAHIWAL_CLUSTER_04_Rural',
'SAHIWAL_CLUSTER_04_Urban','MULTAN_CLUSTER_01_Rural',
'MULTAN_CLUSTER_01_Urban','MULTAN_CLUSTER_02_Rural',
               'MULTAN_CLUSTER_02_Urban',
'MULTAN_CLUSTER_03_Rural','MULTAN_CLUSTER_03_Urban',
               'RYK DESERT_Cluster_Rural',
'SADIQABAD_CLUSTER_01_Rural','SAHIWAL_CLUSTER_05_Rural',
               'SAHIWAL_CLUSTER_05_Urban',
'SAHIWAL_CLUSTER_06_Rural','SAHIWAL_CLUSTER_06_Urban',
               'SAHIWAL_CLUSTER_07_Rural',
 'SAHIWAL_CLUSTER_07_Urban'], 'Center-3'))

cluster_bh['Region'] = cluster_bh['GCell Group'].map(d.get)
```

13.1.8 Modification Cluster Name as per requirement

[7]:
```
# Cluster is Urban or Rural, get from GCell Group
cluster_bh['Cluster Type'] = cluster_bh['GCell Group'].str.strip().str[-5:]
# Remove Urban and Rural from GCell Group
cluster_bh['Location']=cluster_bh['GCell Group'].map(lambda x: str(x)[:-6])
```

13.1.9 Select Quater

[8]:
```
# Identify the Quater Step-1
cluster_bh['Quater'] = pd.PeriodIndex(pd.to_datetime(cluster_bh.Date), freq='Q')
# Select Required Quater
cluster_bh_rq=cluster_bh[cluster_bh.Quater=='2021Q2']
```

13.1.10 Step-1 days

[9]:
```
cluster_bh_rq=cluster_bh_rq.assign(Comments="", Step1="")
cluster_bh_rq_s1=cluster_bh_rq\
    [['Region','Cluster Type','Date',\
      'GCell Group','Comments','Step1']]
```

13.1.11 Sum of Counters

[10]:
```
cluster_bh_rq_cs=cluster_bh_rq.groupby(['Region','Location','Cluster Type'])\
    [['_CallSetup TCH GOS(%)_D','_CallSetup TCH GOS(%)_N',\
    '_GOS-SDCCH(%)_D','_GOS-SDCCH(%)_N',\
    '_Mobility TCH GOS(%)_D', '_Mobility TCH GOS(%)_N',\
      '_DCR_D','_DCR_N',\
     '_RxQual Index DL_1','_RxQual Index DL_2',\
      '_RxQual Index UL_1','_RxQual Index UL_2',\
       '_HSR%_D','_HSR%_N',\
        'CSSR_Non Blocking_1_N','CSSR_Non Blocking_1_D',\
          'CSSR_Non Blocking_2_N','CSSR_Non Blocking_2_D']]\
            .sum().reset_index()
```

13.1.12 Calculate Quarter KPIs Value

[11]:
```
# calculation for the KPIs
cluster_bh_rq_cs['CSSR']=(1-(cluster_bh_rq_cs['CSSR_Non Blocking_1_N']/
                    cluster_bh_rq_cs['CSSR_Non Blocking_1_D']))*\
             (1-(cluster_bh_rq_cs['CSSR_Non Blocking_2_N']/
                 cluster_bh_rq_cs['CSSR_Non Blocking_2_D']))*100
```

[12]:
```
cluster_bh_rq_cs['DCR']=(cluster_bh_rq_cs['_DCR_N']/
                         cluster_bh_rq_cs['_DCR_D'])*100
```

[13]:
```
cluster_bh_rq_cs['HSR']=(cluster_bh_rq_cs['_HSR%_N']/
                        cluster_bh_rq_cs['_HSR%_D'])*100
```

[14]:
```
cluster_bh_rq_cs['SDCCH GoS']=(cluster_bh_rq_cs['_GOS-SDCCH(%)_N']/
                               cluster_bh_rq_cs['_GOS-SDCCH(%)_D'])*100
```

[15]:
```
cluster_bh_rq_cs['TCH GoS']=(cluster_bh_rq_cs['_CallSetup TCH GOS(%)_N']/
                             cluster_bh_rq_cs['_CallSetup TCH GOS(%)_D'])*100
```

[16]:
```
cluster_bh_rq_cs['MoB GoS']=(cluster_bh_rq_cs['_Mobility TCH GOS(%)_N']/
                              cluster_bh_rq_cs['_Mobility TCH GOS(%)_D'])*100
```

[17]:
```
cluster_bh_rq_cs['DL RQI']=(cluster_bh_rq_cs['_RxQual Index DL_1']/
                             cluster_bh_rq_cs['_RxQual Index DL_2'])*100
```

[18]:
```
cluster_bh_rq_cs['UL RQI']=(cluster_bh_rq_cs['_RxQual Index UL_1']/
                             cluster_bh_rq_cs['_RxQual Index UL_2'])*100
```

[19]:
```
#select KPIs only
cluster_bh_rq_cs=cluster_bh_rq_cs[['Region','Location',\
        'Cluster Type','CSSR','DCR','HSR','SDCCH GoS',\
           'TCH GoS','MoB GoS','DL RQI','UL RQI']]
```

13.1.13 Re-Shape Data Set

[20]:
```
cluster_bh_rq_cs_rs=pd.DataFrame(pd.
 ↪melt(cluster_bh_rq_cs,id_vars=['Region','Location','Cluster Type'],\
                        var_name='KPI', value_name='KPI Value')).dropna()
```

13.1.14 SLA Target

[21]:
```
sla=pd.DataFrame({
'KPI':['CSSR','CSSR','DCR','DCR','HSR','HSR','SDCCH GoS','SDCCH GoS','TCH GoS',\
       'TCH GoS','MoB GoS','MoB GoS','DL RQI','DL RQI','UL RQI','UL RQI'],
'Cluster Type':['Urban','Rural','Urban','Rural','Urban','Rural','Urban','Rural',\
                'Urban','Rural','Urban','Rural','Urban','Rural','Urban','Rural'],
'Target Value':[99.5,99,0.6,1,97.5,96,0.1,0.1,2,2,4,4,98.4,97,98.2,97.7]
})
# Transpose SLA Target
sla1 = sla.set_index(['KPI','Cluster Type']).T
```

13.1.15 Merge Re-Shape Data Set & SLA Target

[22]:
```
cluster_bh_rq_cs_rs_t= pd.merge(cluster_bh_rq_cs_rs,sla\
                [['KPI','Cluster Type','Target Value']],\
                on=['KPI','Cluster Type'])
```

13.1.16 Compare with SLA Target

[23]:
```
# Cell Name in the requied format
cluster_bh_rq_cs_rs_t['Comments'] = np.where(
            (((cluster_bh_rq_cs_rs_t['KPI']=='CSSR')|\
              (cluster_bh_rq_cs_rs_t['KPI']=='HSR') |\
              (cluster_bh_rq_cs_rs_t['KPI']=='DL RQI') |\
              (cluster_bh_rq_cs_rs_t['KPI']=='UL RQI'))& \
             (cluster_bh_rq_cs_rs_t['KPI Value'] >= \
            cluster_bh_rq_cs_rs_t['Target Value'])),'Conformance',
    np.where(
            (((cluster_bh_rq_cs_rs_t['KPI']=='DCR')|\
              (cluster_bh_rq_cs_rs_t['KPI']=='SDCCH GoS')|\
              (cluster_bh_rq_cs_rs_t['KPI']=='TCH GoS') |\
              (cluster_bh_rq_cs_rs_t['KPI']=='MoB GoS'))& \
             (cluster_bh_rq_cs_rs_t['KPI Value'] <= \
              cluster_bh_rq_cs_rs_t['Target Value'])),'Conformance',
         'Non Conformance'))
```

13.1.17 Non-SLA KPIs Sheet

[24]:
```
non_sla_kpis=cluster_bh_rq_cs_rs_t\
    [cluster_bh_rq_cs_rs_t.Comments=='Non Conformance']
```

13.1.18 Summary

[25]:
```
kp3=non_sla_kpis.pivot_table(index=['KPI','Cluster Type'],\
        columns='Region',values=['KPI Value'],\
        aggfunc='count',margins=True,margins_name='Region Center').\
        fillna(0).apply(np.int64)
```

[26]:
```
kp3=kp3.iloc[:-1,:]
```

[27]:
```
gg3=pd.DataFrame(kp3.stack()).reset_index()
gg4 = gg3.pivot_table(index=['Region','Cluster Type'],\
                columns='KPI', aggfunc='sum').fillna(0)
#sub total
gg4['Total NC KPIs']= gg4.sum(level=0, axis=1)
```

[28]:
```
#sub total for each region
gg4 = gg4.unstack(0)
mask = gg4.columns.get_level_values('Region') != 'All'
gg4.loc['subtotal'] = gg4.loc[:, mask].sum()
gg4 = gg4.stack().swaplevel(0,1).sort_index()
```

13.1.19 HQ Format

[29]:
```
qformat=cluster_bh_rq_cs.pivot_table(index=['Region','Location'],\
        columns='Cluster Type',\
        values=['CSSR','DCR','HSR',\
        'SDCCH GoS','TCH GoS','MoB GoS','DL RQI','UL RQI'],\
        aggfunc=sum).\
        fillna('N/A').\
        sort_index(level=[0,1],axis=1,ascending=[True,False])
```

[30]:
```
#Rrequired sequence
qformat=qformat[[('CSSR', 'Urban'),\
                 ('CSSR', 'Rural'),\
                 ('DCR', 'Urban'),\
                 ('DCR', 'Rural'),\
                 ('HSR', 'Urban'),\
                 ('HSR', 'Rural'),\
                ('SDCCH GoS', 'Urban'),\
                 ('SDCCH GoS', 'Rural'),\
                 ('TCH GoS', 'Urban'),\
                 ('TCH GoS', 'Rural'),\
                ('MoB GoS', 'Urban'),\
                 ('MoB GoS', 'Rural'),\
                ('DL RQI', 'Urban'),\
                 ('DL RQI', 'Rural'),\
                 ('UL RQI', 'Urban'),\
                 ('UL RQI', 'Rural')]]
```

[31]:
```
qformat=qformat.reset_index()
```

[32]:
```
#export
qformat.to_excel('Quarter_Conformanc.xlsx',engine='openpyxl',na_rep='N/A')
```

[33]:
```
#import
aa=pd.read_excel('Quarter_Conformanc.xlsx',header=[0,1])
```

13.1.20 Formatting Final Data Set

[34]:
```
# Formatting
bb=aa.style\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x>=99.50 else 'background-color: %s' % 'red'
          ,subset=[('CSSR','Urban')])\
.applymap(lambda x: 'color: black' if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x>=99.00
          else 'background-color: %s' % 'red'
          ,subset=[('CSSR','Rural')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x<=0.60  else 'background-color: %s' % 'red'
          ,subset=[('DCR','Urban')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
```

```
          else 'background-color: %s' % 'green'
          if x<=1.00  else 'background-color: %s' % 'red'
          ,subset=[('DCR','Rural')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x>=97.50 else 'background-color: %s' % 'red'
          ,subset=[('HSR','Urban')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x>=96.00 else 'background-color: %s' % 'red'
          ,subset=[('HSR','Rural')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x<=0.10  else 'background-color: %s' % 'red'
          ,subset=[('SDCCH GoS','Urban')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x<=0.10  else 'background-color: %s' % 'red'
          ,subset=[('SDCCH GoS','Rural')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x<=2.00  else 'background-color: %s' % 'red'
          ,subset=[('TCH GoS','Urban')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x<=2.00  else 'background-color: %s' % 'red'
          ,subset=[('TCH GoS','Rural')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x<=4.00  else 'background-color: %s' % 'red'
          ,subset=[('MoB GoS','Urban')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x<=4.00  else 'background-color: %s' % 'red'
          ,subset=[('MoB GoS','Rural')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x>=98.40 else 'background-color: %s' % 'red'
```

```
          ,subset=[('DL RQI','Urban')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x>=97.00 else 'background-color: %s' % 'red'
          ,subset=[('DL RQI','Rural')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x>=98.20 else 'background-color: %s' % 'red'
          ,subset=[('UL RQI','Urban')])\
.applymap(lambda x: 'color: black'
          if pd.isnull(x)
          else 'background-color: %s' % 'green'
          if x>=97.70 else 'background-color: %s' % 'red'
          ,subset=[('UL RQI','Rural')])
```

13.1.21 Export Final Data Set

[35]:
```
with pd.ExcelWriter('Quarter_Conformanc.xlsx') as writer:
    sla1.to_excel(writer,sheet_name="SLA Target",\
                  engine='openpyxl',na_rep='N/A')
    gg4.to_excel(writer,sheet_name="Summary",\
                 engine='openpyxl',na_rep='N/A')
    bb.to_excel(writer,sheet_name="2021Q2",\
                engine='openpyxl',na_rep='N/A')
    non_sla_kpis.to_excel(writer,sheet_name="Non-Conformance",\
                engine='openpyxl',na_rep='N/A',index=False)
    cluster_bh_rq_s1.to_excel(writer,sheet_name="Step-1",\
                engine='openpyxl',na_rep='N/A',index=False)
    cluster_bh.to_excel(writer,sheet_name='Cluster_Num_Dem',\
                engine='openpyxl',na_rep='N/A',index=False)
```

13.1.22 Delete csv File

[36]:
```
for filename in os.listdir(folder_path):
    if filename.endswith('.csv'):
        os.unlink(os.path.join(folder_path, filename))
```

13.1.23 Output File Format

- Output File Format

14. GSM Quatrly Data Reshape

14.1 Quarterly Conformance Reshape

14.1.1 Input File Format

- Quarterly Conformance Re-Shape

14.1.2 Import required Libraries

[1]:
```
import os
import pandas as pd
from glob import glob
```

14.1.3 Set Working Path

[2]:
```
folder_path = 'D:/DataSets/Conformance/Quaterly Conformance Data Re-Shape'
os.chdir(folder_path)
```

14.1.4 Concat all Quarterly Conformance

[3]:
```
all_files = glob('*.XLSX')
df_from_each_file = (pd.read_excel(f,header=[0,1]).\
    assign(Quatr=os.path.basename(f).\
           split('.')[0]) for f in all_files)
concatdf = pd.concat(df_from_each_file, ignore_index=True)
```

14.1.5 Melt (re-shape) Data Set

[4]:
```
df=pd.melt(concatdf,\
   id_vars=[('Region', 'Unnamed: 0_level_1'),\
            ('Cell Group', 'Unnamed: 1_level_1'),\
            ('Quatr','')],
        var_name=["KPI-Name",'Cluster-Sub-Zone'],\
           value_name='KPI-Value')
```

14.1.6 Data Pre-Processing

[5]:
```
# re-name columns name
df = df.rename(columns = {('Region', 'Unnamed: 0_level_1') : 'Region',
                          ('Cell Group', 'Unnamed: 1_level_1'): 'Cluster',
                           ('Quatr',''): 'Quatr'})
# trim the column value
df['Quatr']=  df['Quatr'].str.replace(' NPM GSM KPIs Summary Step 0', '')
# ignore the NaN rows
df = df[df['KPI-Value'].notnull()]
```

14.1.7 Pivot (re-shape) Data Set in Required Format

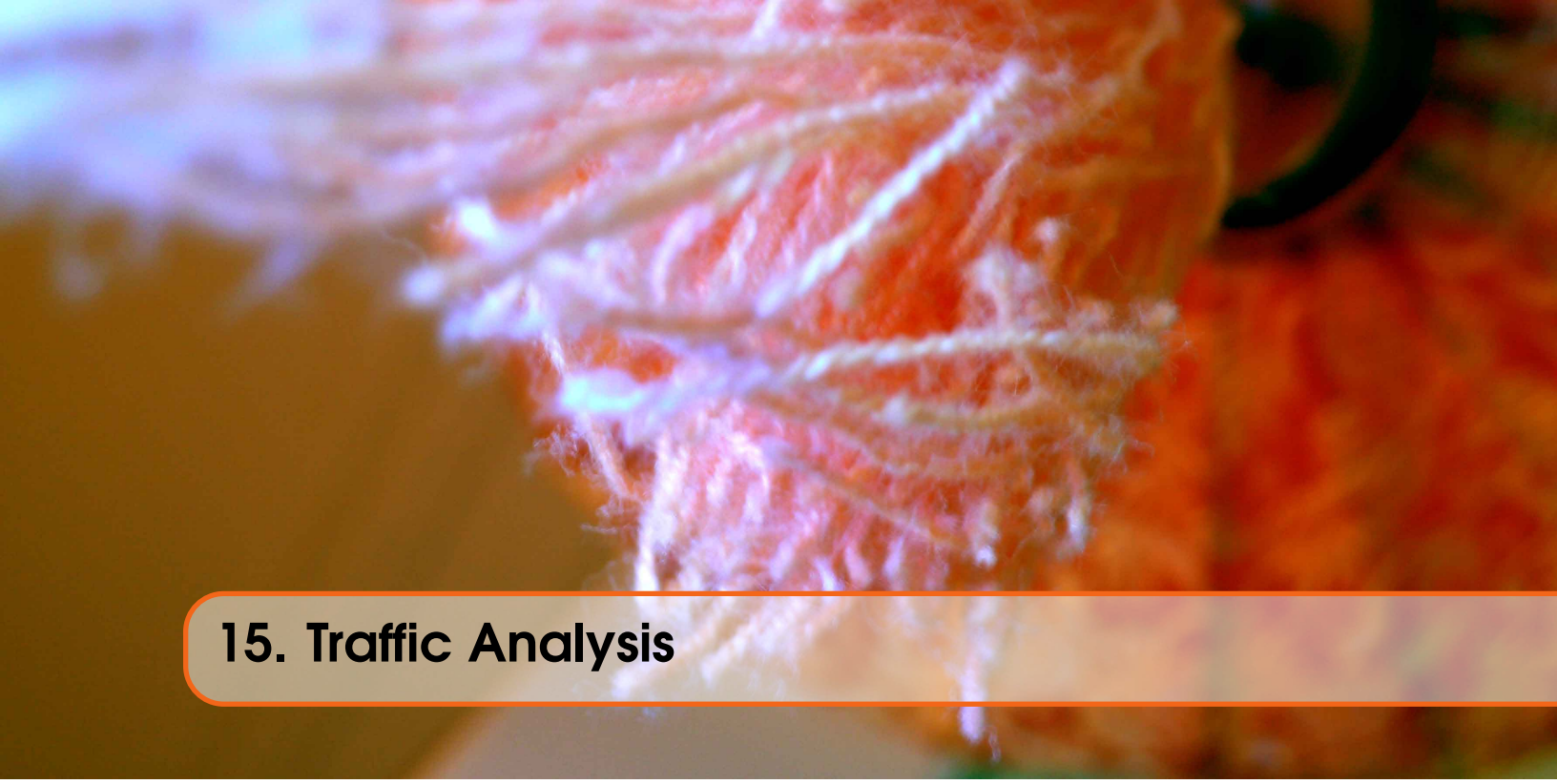
[6]:
```
df1 = df.pivot(index=['Region','Cluster','Cluster-Sub-Zone','Quatr'],\
        columns="KPI-Name",values='KPI-Value')\
        .reset_index()
```

14.1.8 Export Final Data Set

[7]:
```
df1.to_csv('2G_Quatrly_Data_Reshape.csv',index=False)
```

14.1.9 Output File Format

- Output File Format



15. Traffic Analysis

15.1 Traffic Analysis

15.1.1 Input File Format

- PTML Center Region Traffic

15.1.2 Import required Libraries

[1]:
```
import os
import numpy as np
import pandas as pd
```

15.1.3 Working Path

[2]:
```
working_directory = 'D:/DataSets/KPIs/Traffic'
os.chdir(working_directory)
```

15.1.4 Import Excel Sheets

[3]:
```
df = pd.read_excel('Center_Traffic.xlsx',\
    sheet_name='2G DA',\
    converters={'Integrity': lambda value: '{:,.0f}%'.format(value * 100)},\
    parse_dates=['Date'])\
    .rename(columns={'GCell Group':'Cluster',\
    'Global Traffic':'CS Traffic',\
    'Payload(GB)':'PS Traffic'})
```

[4]:
```
df0= pd.read_excel('Center_Traffic.xlsx',\
    sheet_name='3G DA',\
    converters={'Integrity': lambda value: '{:,.0f}%'.format(value * 100)},\
    parse_dates=['Date'])\
    .rename(columns={'UCell Group':'Cluster',\
                     'AMR CS Traffic (Erl)(Erl)':\
                  'CS Traffic','Data Volume (GB) Incl.VP':'PS Traffic'})
```

[5]:
```
df1= pd.read_excel('Center_Traffic.xlsx',\
    sheet_name='4G DA',\
    converters={'Integrity': lambda value: '{:,.0f}%'.format(value * 100)},\
    parse_dates=['Date'])\
    .rename(columns={'LTE Cell Group':'Cluster',\
            'Data Volume (GB)':'PS Traffic'})
```

15.1.5 Add additional columns

[6]:
```
df['Tech']  = '2G'
df0['Tech'] = '3G'
df1['Tech'] = '4G'
#Insert a column in s specific location
df1.insert(3,'CS Traffic',0)
```

15.1.6 Concat data frames

[7]:
```
df2=pd.concat([df,df0,df1])
```

15.1.7 map cluster name to Region

[8]:
```
df2['Region'] = df2.Cluster.map({'REGIONAL_CENTERAL':'Center', \
                                 'NPM_REGIONAL_CENTRAL':'Center',\
                                'NPM_LTE_REGIONAL_CENTRAL':'Center',\
                                 'REGIONAL_CENTER01_215':'Center1',\
                                'NPM_REGIONAL_CENTRAL_01':'Center1',\
                                 'NPM_LTE_REGIONAL_CENTRAL_01':'Center1',\
                                'REGIONAL_CENTER02_102':'Center2',\
                                 'NPM_REGIONAL_CENTRAL_02':'Center2',\
                                'NPM_LTE_REGIONAL_CENTRAL_02':'Center2',\
                                 'REGIONAL_CENTER03_102':'Center3',
                               'NPM_REGIONAL_CENTRAL_03':'Center3',\
                                 'NPM_LTE_REGIONAL_CENTRAL_03':'Center3'})
```

15.1.8 Pivot_table(re-shape data set)

[9]:
```
df3=pd.DataFrame(df2.pivot_table(index=["Date",'Region'],\
            columns="Tech",\
            values=['CS Traffic','PS Traffic'],\
            margins=True,\
            aggfunc=sum)).\
            fillna(0).reset_index().\
            iloc[:-1, :]
```

15.1.9 Pre-processing on the header

[10]:
```
#header map and join
df3.columns = df3.columns.map('_'.join)
# replace and strip columns name
df3.columns = df3.columns.str.replace('_All', '(AllTechnologies)')
df3.columns = df3.columns.str.rstrip("_")
```

15.1.10 Export Final Data Set

[11]:
```
df3.to_csv('GUL_Daily.csv',index=False)
```

15.1.11 Output File Format

- Output File Format

16. Transpose All the Tabs in Excel Sheet

16.1 Transpose All the Tabs in Excel Sheet

16.1.1 Input File Format

- Reference Input Data Set

16.1.2 Import required Libraries

[1]:
```
import os
import pandas as pd
```

16.1.3 Set Working Path

[2]:
```
folder_path = 'D:/DataSets/RF Export/Transpose'
os.chdir(folder_path)
```

16.1.4 Import the Data Set

[3]:
```
# Import File
xls = pd.ExcelFile('transpose.xlsx')
# Get the Sheet Name
sheet_names = xls.sheet_names
# Import File with Sheet Name
sheets = pd.read_excel('transpose.xlsx', sheet_name=sheet_names)
```

16.1.5 Export Final Data Set

[4]:
```
# Export Output
writer = pd.ExcelWriter('transpose-output.xlsx')
for sheet in sheets.items():
    sheet_name = sheet[0]
    df = sheet[1]
    df=df.T
    df.to_excel(writer, sheet_name=sheet_name)
writer.save()
```

16.1.6 Output File Format

- Output File Format

17. LTE High Utilize Cells

17.1 LTE High Utilize Cells

17.1.1 Input File Format

- LTE High Utilize Cells

17.1.2 Import required Libraries

[1]:
```
import os
import pandas as pd
from glob import glob
```

17.1.3 Set Working Path

[2]:
```
folder_path = 'D:/DataSets/KPIs/4G_High_Utilize_Cells_DataSets'
os.chdir(folder_path)
```

17.1.4 Import & concat all csv Files

[3]:
```
all_files = glob('*.csv')
```

[4]:
```
df_from_each_file = (pd.read_csv(f,encoding='latin-1',header=3,\
            skipfooter=1,engine='python',\
            na_values=['NIL','/0'],parse_dates=['Date']).\
            assign(Region=os.path.basename(f).\
            split('.')[0]) for f in all_files)
```

[5]:
```
concatdf = pd.concat(df_from_each_file, ignore_index=True)
```

17.1.5 Count High Utilize Cells - w.r.t Date & Region

[6]:
```
hucellcount = concatdf[((concatdf['DL PRB Avg Utilization']>70) \
                        & (concatdf['DL User Thrp (Mbps)']<2))].\
                groupby('Date')['Region'].\
                value_counts().unstack().\
                add_suffix('_High_Utilize_Cells').\
                reset_index()
```

[7]:
```
hucellcount['Total_NW_High_Utilize_Cells']=hucellcount\
    ['Central_High_Utilize_Cells']+\
    hucellcount['North_High_Utilize_Cells']+\
    hucellcount['South_High_Utilize_Cells']
```

17.1.6 Daily Cell Count- w.r.t Date & Region

[8]:
```
dcellcount = concatdf.groupby('Date')['Region'].\
                value_counts().unstack().\
                add_suffix('_Total_Cells').\
                reset_index()
```

[9]:
```
dcellcount['Total_NW_Cells']= dcellcount['Central_Total_Cells']+\
                            dcellcount['North_Total_Cells']+\
                            dcellcount['South_Total_Cells']
```

17.1.7 Daily Regional DL Volume (GB)

[10]:
```
dlv = concatdf.groupby(['Date','Region'])['DL volume (GB)'].sum().\
            unstack().add_suffix('_DL_volume_(GB)').reset_index()
```

[11]:
```
dlv['Total_NW_DL_Volume_(GB)'] = dlv['Central_DL_volume_(GB)']+\
                                dlv['North_DL_volume_(GB)']+\
                                dlv['South_DL_volume_(GB)']
```

17.1.8 Final Data Set

[12]:
```
rfe=hucellcount.merge(dcellcount,on=['Date'],how="left")\
                .merge(dlv ,on=['Date'],how="left")
```

[13]:
```
# Calculate %age of High Utiliz Cells
rfe['%age_of_Central_High_Utilize_Cells']=\
(rfe['Central_High_Utilize_Cells']/rfe['Central_Total_Cells'])*100
```

[14]:
```
rfe['%age_of_North_High_Utilize_Cells']=\
(rfe['North_High_Utilize_Cells']/rfe['North_Total_Cells'])*100
```

[15]:
```
rfe['%age_of_South_High_Utilize_Cells']=\
(rfe['South_High_Utilize_Cells']/rfe['South_Total_Cells'])*100
```

[16]:
```
rfe['%age_of_NW_High_Utilize_Cells']=\
(rfe['Total_NW_High_Utilize_Cells']/rfe['Total_NW_Cells'])*100
```

17.1.9 Set Working Path

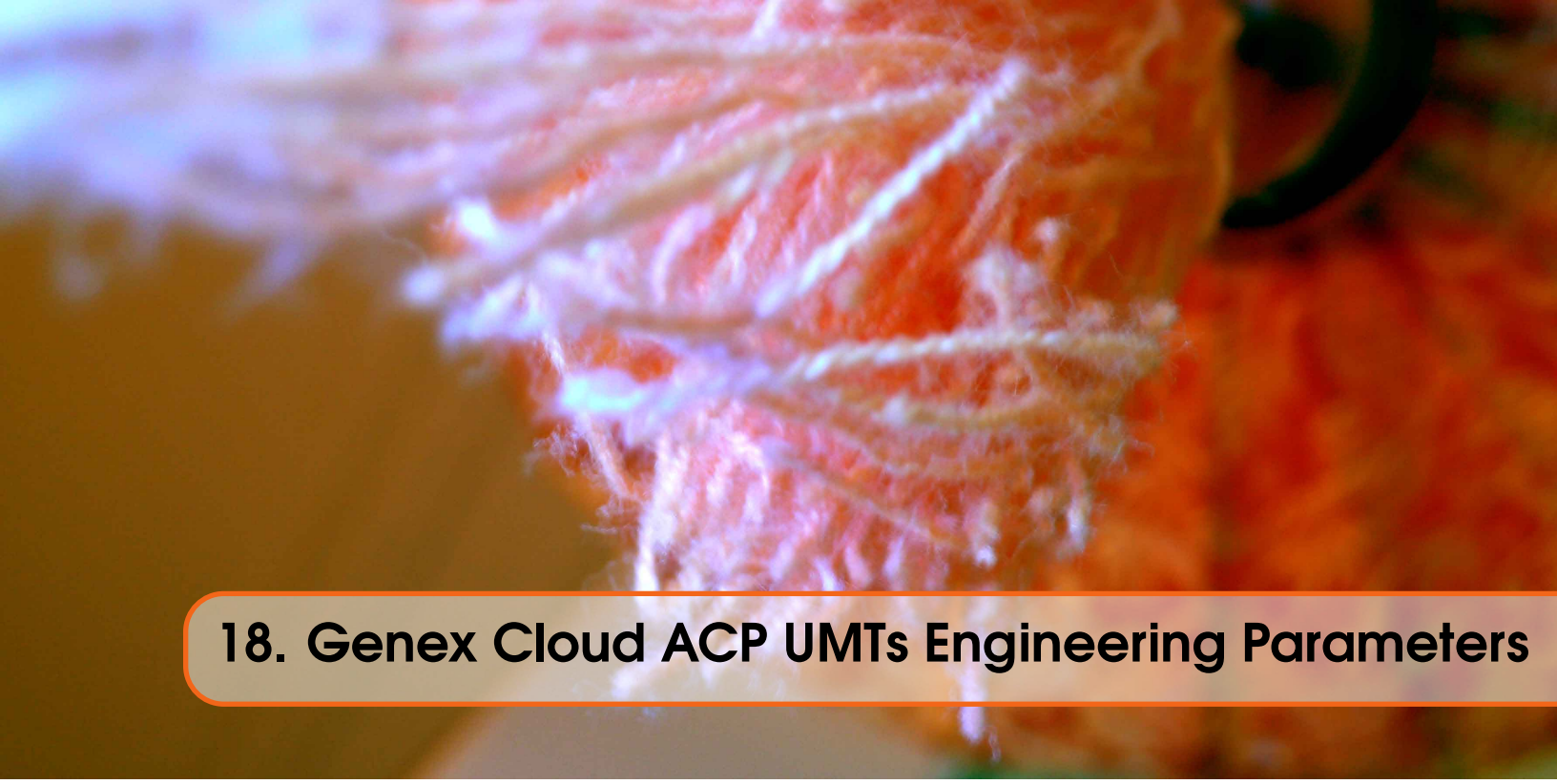
[17]:
```
folder_path = 'D:/DataSets/KPIs/4G_High_Utilize_Cells_DataSets/Output'
os.chdir(folder_path)
```

17.1.10 Export Final Data Set

[18]:
```
#import old data set
dfo = pd.read_csv('Output.csv',parse_dates=['Date'])
# concat with today's Output
dff=pd.concat([dfo,rfe],ignore_index=True)
dff.to_csv('Output.csv',index=False)
```

17.1.11 Output File Format

- Output File Format



18. Genex Cloud ACP UMTs Engineering Parameters

18.1 ACP UMTs Engineering Parameters

18.1.1 Input File Format

- Genex Cloud For ACP Input File

18.1.2 Import required Libraries

[1]:
```
import os
import numpy as np
import pandas as pd
from glob import glob
```

18.1.3 Set Working Path For Cell Files(RF Export)

[2]:
```
working_directory = 'D:/DataSets/KPIs/GenexCloud/3G_Cell_Export'
os.chdir(working_directory)
```

18.1.4 Import GCELL Files

[3]:
```
#check the file list in the directory
cell = sorted(glob('*.txt'))
#read all files, import only required columns
gell_export=pd.concat((pd.read_csv(file,header=1,\
        engine='python',usecols=['BSC Name','NodeB Name','Cell ID',\
        'Cell Name','Downlink UARFCN',\
        'DL Primary Scrambling Code',\
        'Max Transmit Power of Cell',\
        'PCPICHPOWER','Validation indication']) for file in cell)).\
        rename(columns={'BSC Name':'RNC'})
```

18.1.5 Calculate Max Transmite Power

[4]:
```
gell_export['Max Transmite Power'] = \
    10**(gell_export['Max Transmit Power of Cell']/100)/1000
```

18.1.6 Set Working Path For PRS Counter

[5]:
```
working_directory = 'D:/DataSets/KPIs/GenexCloud/PRS_Counter'
os.chdir(working_directory)
```

18.1.7 Import PRS Counter VS.MeanTCP(dBm)

- Resorce Busy Hour KPIs of last 7 days

[6]:
```
df = pd.read_csv('Counter.csv',header=3,\
                 skipfooter=1,engine='python',\
                 parse_dates=["Date"])
```

18.1.8 Average TCP Calculation

[7]:
```
df1=pd.DataFrame(df.groupby(['RNC','Cell ID'])\
                .apply(lambda x: np.average(x['VS.MeanTCP(dBm)'])).\
                           reset_index(name='AvgTCP'))
```

[8]:
```
# Calculate Average TCP
df1['AvgTCP Calculation']= 10**(df1['AvgTCP']/10)/1000
```

18.1.9 Merge GCELL and Counter Calculation

[9]:
```
df3=pd.merge(gell_export,df1,on=['RNC','Cell ID'],how="left")
```

[10]:
```
# Calculate TCP Utilization
df3['Actual Load Power DL']= \
```

```
    (df3['AvgTCP Calculation']/df3['Max Transmite Power'])
```

[11]:
```
df3['Pilot Power']= (df3['PCPICHPOWER'])/10
```

[12]:
```
df3['Max Power']= (df3['Max Transmit Power of Cell'])/10
```

18.1.10 Format Columns

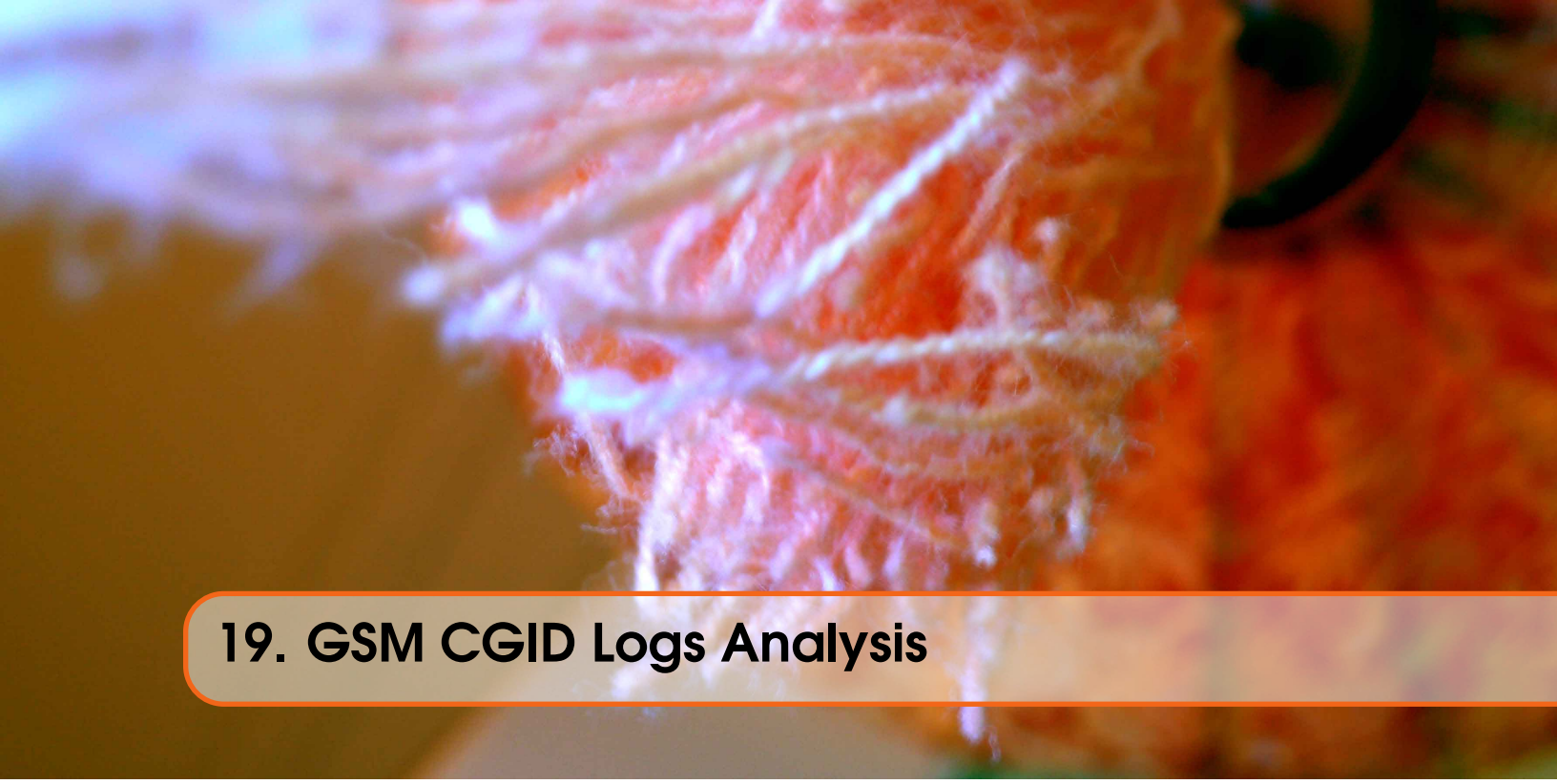
[13]:
```
df4= df3[['RNC','NodeB Name','Cell ID','Cell Name','Downlink UARFCN',
'DL Primary Scrambling Code','Max Transmit Power of Cell','PCPICHPOWER',
'Validation indication','Actual Load Power DL','Pilot Power','Max Power']]
```

18.1.11 Export Final Data Set

[14]:
```
df4.to_csv('GENEXCloud_Platform_ACP_UMTS_Engineer_Parameter.csv',index=False)
```

18.1.12 Output File Format

- Output File Format



19. GSM CGID Logs Analysis

19.1 CGID Log Analysis

19.1.1 Input File Format

- CGID Logs From NIC
- 2G Frequency Export

19.1.2 Import required Libraries

[1]:
```
import os
import glob
import zipfile
import pandas as pd
```

19.1.3 Import CGID Log Files

[2]:
```
#set the Path (Path must be same format)
path = 'D:/DataSets/RFExport/CGID'
os.chdir(path)
```

[3]:
```
# get the the file names in the dir, sub dir's
nicFiles = list()
for (path, dirnames, filenames) in os.walk(path):
    nicFiles += [os.path.join(path, file) for file in filenames]
#filter required Files
matchers = ['.log_']
matching = [s for s in nicFiles if any(xs in s for xs in matchers)]
```

[4]:
```
#concat the All the CGID log Files with specific columns
df_from_each_file = (pd.read_csv(f,header=None,usecols=[0,12],\
            names=['Time','Cell Index'])\
            .assign(File=f.split('.')[0]) for f in matching)
```

[5]:
```
concatdf = pd.concat(df_from_each_file, ignore_index=True)
```

[6]:
```
#Preprocessing to get the BSC NE
concatdf['NE FDN'] = concatdf['File'].str.split(' ').\
                    str[1].str.split('\\').str[0].\
                    str.split('_').str[1]
```

[7]:
```
# Data Pre-Processing
concatdf['Cell Index']=concatdf['Cell Index'].str.replace("CellId:","")
concatdf['Time']=concatdf['Time'].str.replace("Time:","")
concatdf['Time']=concatdf['Time'].astype('datetime64[ns]')
concatdf['Time']=concatdf['Time'].dt.date
concatdf['Cell Index']=concatdf['Cell Index'].astype(str)
```

19.1.4 Import BSC NE Info File

[8]:
```
bsc = pd.read_csv('TimeCostReport.csv')
```

19.1.5 Merge CGID Logs and BSC NE Files

[9]:
```
dff = pd.merge(concatdf,bsc[['NE FDN','NE Name']],on=['NE FDN'])
```

19.1.6 Import GCELL File

[10]:
```
path = 'D:/DataSets/RFExport/2GCellFreqExport'
os.chdir(path)
```

[11]:
```
gcell = pd.read_csv('gcell.txt',header=1,\
                usecols=['BSC Name','BTS Name','Cell Index',\
                         'Cell Name','Cell CI'],\
            dtype = {"Cell Index" : "str"}).\
            rename(columns={'BSC Name': 'NE Name'})
```

19.1.7 Merge CGID log with GCell File (From RF Export)

[12]:
```
dff1 = pd.merge(dff,gcell,on=['NE Name','Cell Index'])
```

19.1.8 Re-Shape CGID Log Analysis

[13]:
```
cgid_log_analysis=dff1.pivot_table\
        (index=['NE Name','Cell Index','BTS Name','Cell Name']\
        ,columns="Time",values='File',aggfunc='count').\
        fillna(0).reset_index()
```

19.1.9 Export CGID Log Summary

[14]:
```
cgid_log_analysis.to_csv('CGID_Analysis_File.csv',index=False)
```

19.1.10 Output File Format

- Output File Format

20. External Interference Tracker Parser

20.1 External Interference Tracker

20.1.1 Input File Format

- External Interference Trackers
- Input File must be Following Format; - input.xlsx

RACaseSerialNo	City	3G:U900	3G:U2100	4G
523/C	Lahore	27124; 27159	27124; 27159	27124; 27159

20.1.2 Import required Libraries

[1]:
```
import os
import numpy as np
import pandas as pd
```

20.1.3 Set Working Path

[2]:
```
folder_path = 'D:/DataSets/ExternalInterference_Tracker'
os.chdir(folder_path)
```

20.1.4 Import External Interference Tracker

[3]:
```
df = pd.read_excel('input.xlsx',sheet_name=0)
df.to_csv('External_Interference.txt',index=False)
df1 = pd.read_csv('External_Interference.txt')
```

20.1.5 Data Pre-Processing

[4]:
```
# Pre-Processing (2G Cells)
df1['2G']=  df1['2G'].str.replace(r'[^A-Za-z0-9,;]+', '')
df1['2G'] = df1['2G'].str.rstrip(';')
df1['2G'] = df1['2G'].str.replace(",",";")
df1['2G'] = df1['2G'].str.replace(";;","_")
df1['2G'] = df1['2G'].str.replace(";","_")

# Pre-Processing (3G-U900 Cells)
df1['3G:U900'] = df1['3G:U900'].str.replace(" ","")
df1['3G:U900']=  df1['3G:U900'].str.replace(r'[^A-Za-z0-9,;]+', '')
df1['3G:U900'] = df1['3G:U900'].str.rstrip(';')
df1['3G:U900'] = df1['3G:U900'].str.rstrip(',')
df1['3G:U900'] = df1['3G:U900'].str.replace(",",";")
df1['3G:U900'] = df1['3G:U900'].str.replace(";;","_")
df1['3G:U900'] = df1['3G:U900'].str.replace(";","_")
df1['3G:U900'] = df1['3G:U900'].str.replace("l","_")
# Pre-Processing (3G-U2100 Cells)
df1['3G:U2100'] = df1['3G:U2100'].str.replace(" ",";")
df1['3G:U2100'] = df1['3G:U2100'].str.replace(r'[^A-Za-z0-9,;]+', '')
df1['3G:U2100'] = df1['3G:U2100'].str.rstrip(';')
df1['3G:U2100'] = df1['3G:U2100'].str.rstrip(',')
df1['3G:U2100'] = df1['3G:U2100'].str.replace(",",";")
df1['3G:U2100'] = df1['3G:U2100'].str.replace(";;","_")
df1['3G:U2100'] = df1['3G:U2100'].str.replace(";","_")
df1['3G:U2100'] = df1['3G:U2100'].str.replace("l","_")
# Pre-Processing (LTE Cells)
df1['4G'] = df1['4G'].str.replace(" ",";")
df1['4G'] = df1['4G'].str.replace(r'[^A-Za-z0-9,;]+', '')
df1['4G'] = df1['4G'].str.rstrip(';')
df1['4G'] = df1['4G'].str.rstrip(',')
df1['4G'] = df1['4G'].str.replace(",",";")
df1['4G'] = df1['4G'].str.replace(";;","_")
df1['4G'] = df1['4G'].str.replace(";","_")
df1['4G'] = df1['4G'].str.replace("l","_")
```

20.1.6 Drop un-Required Rows for each technology

[5]:
```
#drop na rows for each technology
df2=df1.dropna(subset = ['2G'],axis=0).copy()
df3=df1.dropna(subset = ['3G:U900'],axis=0).copy()
df4=df1.dropna(subset = ['3G:U2100'],axis=0).copy()
df5=df1.dropna(subset = ['4G'],axis=0).copy()
```

20.1.7 apply list on cell list

[6]:
```
# cells convert to the list
df2.loc[:, '2G']       = df2['2G'].str.split('_').apply(list)
df3.loc[:, '3G:U900']  = df3['3G:U900'].str.split('_').apply(list)
df4.loc[:, '3G:U2100'] = df4['3G:U2100'].str.split('_').apply(list)
df5.loc[:, '4G']       = df5['4G'].str.split('_').apply(list)
```

20.1.8 Re-shape & Filter Required Columns

[7]:
```
df21=df2.explode('2G').reset_index(level=-1, drop=True)
df21 = df21[['RACaseSerialNo','City','2G']]
df31=df3.explode('3G:U900').reset_index(level=-1, drop=True)
df31 = df31[['RACaseSerialNo','City','3G:U900']]
df41=df4.explode('3G:U2100').reset_index(level=-1, drop=True)
df41 = df41[['RACaseSerialNo','City','3G:U2100']]
df51=df5.explode('4G').reset_index(level=-1, drop=True)
df51 = df51[['RACaseSerialNo','City','4G']]
```

20.1.9 Export Final Data Set

[8]:
```
with pd.ExcelWriter('CenterRegion_IOI_Cell_List.xlsx') as writer:
    df21.to_excel(writer,sheet_name="2G_Cell_List",\
                  engine='openpyxl',na_rep='N/A',\
                  index=False,float_format='%.2f')
    df31.to_excel(writer,sheet_name="3G_U900_Cell_List",\
                  engine='openpyxl',na_rep='N/A',\
                  index=False,float_format='%.2f')
    df41.to_excel(writer,sheet_name="3G_U2100_Cell_List",\
                  engine='openpyxl',na_rep='N/A',\
                  index=False,float_format='%.2f')
    df51.to_excel(writer,sheet_name="4G_Cell_List",\
                  engine='openpyxl',na_rep='N/A',\
                  index=False,float_format='%.2f')
```

20.1.10 Output File Format

- Output File Format

21. External Interference Tracker Final Output

21.1 Final Output For External Interference

21.1.1 Input File Format

- IOI Cell List

21.1.2 Import required Libraries

[1]:
```
# import requied library
import pandas as pd
```

21.1.3 Import Data Set

[2]:
```
# import required data set
df = pd.read_csv('IOI_Final_Output_Format.csv',\
                 dtype = {"CI" : "object"})
```

21.1.4 groupby join in a row

[3]:
```
#convert the multi rows to single group row
df1=pd.DataFrame(df.groupby(['RACaseSerialNo'])['CI'].\
                 apply(';'.join)).reset_index()
```

21.1.5 Count of cells in a list

[4]:
```
# get the count of cell for each RCA Cells count
df1['Count']=df1['CI'].str.split(';').\
                apply(set).str.len()
```

21.1.6 Export Data Set

[5]:
```
# export the output
df1.to_csv('ioi_cell_2G.csv',index=False)
```

21.2 Compare External Interference Trackers

21.2.1 Input File

- Compare External Interference Trackers

21.2.2 Import required Libraries

[1]:
```
# import requied library
import pandas as pd
import numpy as np
```

21.2.3 Import Data Set

[2]:
```
# import required data set
df = pd.read_csv('Compare.csv')
```

21.2.4 Convert the column in a set

[3]:
```
# data pre-processing and convert the required columns to set
df.loc[:,'May_CI']  = df['May_CI'].\
                    replace(';','_').\
                    replace(np.nan,'').\
                    str.split('_').\
                    apply(set)

df.loc[:,'June_CI'] = df['June_CI'].\
                    replace(';','_').\
                    replace(np.nan,'').\
                    str.split('_').\
                    apply(set)
```

21.2.5 Data Pre Processing

[4]:
```
# find the set difference
df['Cell_exc_in_June'] = df.apply(lambda x: x['May_CI'].
 ↪difference(x['June_CI']), axis=1)
df['Cell_add_in_June'] = df.apply(lambda x: x['June_CI'].
 ↪difference(x['May_CI']), axis=1)
```

21.2.6 empty set

[5]:
```
# replace the empty set to No Change
df.loc[:] = df[:].replace(set(),'NoChange')
```

[6]:
```
# conver the set to normal string format
df.loc[:]= df[:].applymap(lambda x: ",".join(x) if isinstance(x, set) else x)
```

[7]:
```
# fill the missing values
df.loc[:,'Cell_exc_in_June'] = df['Cell_exc_in_June'].replace('','NewCase')
df.loc[:,'Cell_add_in_June'] = df['Cell_add_in_June'].replace('','CaseClose')
```

21.2.7 Export Final Data Set

[8]:
```
#export
df1.to_csv('Compare_results.csv',index=False)
```

21.2.8 Output File Format

- Output File Format

22. UMTs Timers Merge w.r.t Column

22.1 UMTs Timers concat using index columns

22.1.1 Input File Format

- 3G Timers

22.1.2 Import required Libraries

[6]:
```
import os
import pandas as pd
```

22.1.3 Set Working Path

[7]:
```
folder_path = 'D:/DataSets/RF Export/3GTimers'
os.chdir(folder_path)
```

22.1.4 Import all the csv File as a list

[8]:
```
dfs = [pd.read_csv(f, index_col=['NeName'])
       for f in os.listdir(os.getcwd()) if f.endswith('csv')]
```

22.1.5 Merge the Files w.r.t rows

[9]:
```
finaldf = pd.concat(dfs, axis=1, join='outer').reset_index()
```

22.1.6 Remove duplicated columns

[10]:
```
finaldf = finaldf.loc[:,~finaldf.columns.duplicated()]
```

22.1.7 Export Final Data Set

[11]:
```
finaldf.to_csv('3G_Timers_Ouput.csv',index=False)
```

22.1.8 Output File Format

- Output File Format

23. GSM DSP Formatting

23.1 GSM DSP fixed-width formatted lines

23.1.1 Input File Format

- GSM DSP File
- 2G Frequency Export

23.1.2 Import required Libraries

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
from glob import glob
```

23.1.3 Set Working Path

[2]:
```
working_directory = 'D:/DataSets/DSP/2G_DSP'
os.chdir(working_directory)
```

23.1.4 Import 2G DSP File

[3]:
```
#import 2G DSP File
df=pd.read_fwf('2G_DSP.txt',\
    colspecs = [(0,19),(0,19),(19,34),(34,300)],\
    names=['BSCName','Cn Operator Index', \
           'Operator Name', \
           'License Identifier_License Item_Allocated_Usage'],\
               comment='+++')
```

23.1.5 Data Pre-Processing

[4]:
```
# get the BSC Name for each row
df.BSCName = df.BSCName.where(df.BSCName.str.contains('BSC')).ffill()
```

[5]:
```
#drop NaN values
df = df.dropna(subset=['License Identifier_License Item_Allocated_Usage'])
```

[6]:
```
# Condition Filter
df = df[df['Cn Operator Index'] == '0']
```

[7]:
```
#strip
df['License Identifier_License Item_Allocated_Usage']= \
    df['License Identifier_License Item_Allocated_Usage']\
                .str.strip().str.replace('\s\s+', ';')
```

[8]:
```
#split
df[['License Identifier', 'License Item', 'Allocated','Usage']] = \
    df['License Identifier_License Item_Allocated_Usage']\
                            .str.split(';', expand=True)
```

[9]:
```
# drop unreqired column
df=df.drop(['License Identifier_License Item_Allocated_Usage'],axis=1)\
                                                    .reset_index(drop=True)
```

23.1.6 TRX Count BSC Level

[10]:
```
# import gtrx
gtrx=pd.read_csv('GTRX.txt',header=1)
# Identify the TRX Band
gtrx['Band'] = np.where(
            ((gtrx['Frequency']>=25) & (gtrx['Frequency']<=62)),
            'GSM',
            np.where(
                    (gtrx['Frequency']>=556) & (gtrx['Frequency']<=599),
                    'DCS',
                     'Other Band'))
```

[11]:
```
# cross tab
trxcount=pd.crosstab(gtrx['BSC Name'],gtrx['Band']).reset_index().fillna(0)
```

[12]:
```
# total TRXs per BSC
trxcount['TotalTRXCount']=trxcount['GSM']+trxcount['DCS']
```

[13]:
```
#rename coluns
trxcount=trxcount.rename(columns={'BSC Name':'BSCName'})
```

23.1.7 Merge DSP File and TRX Count

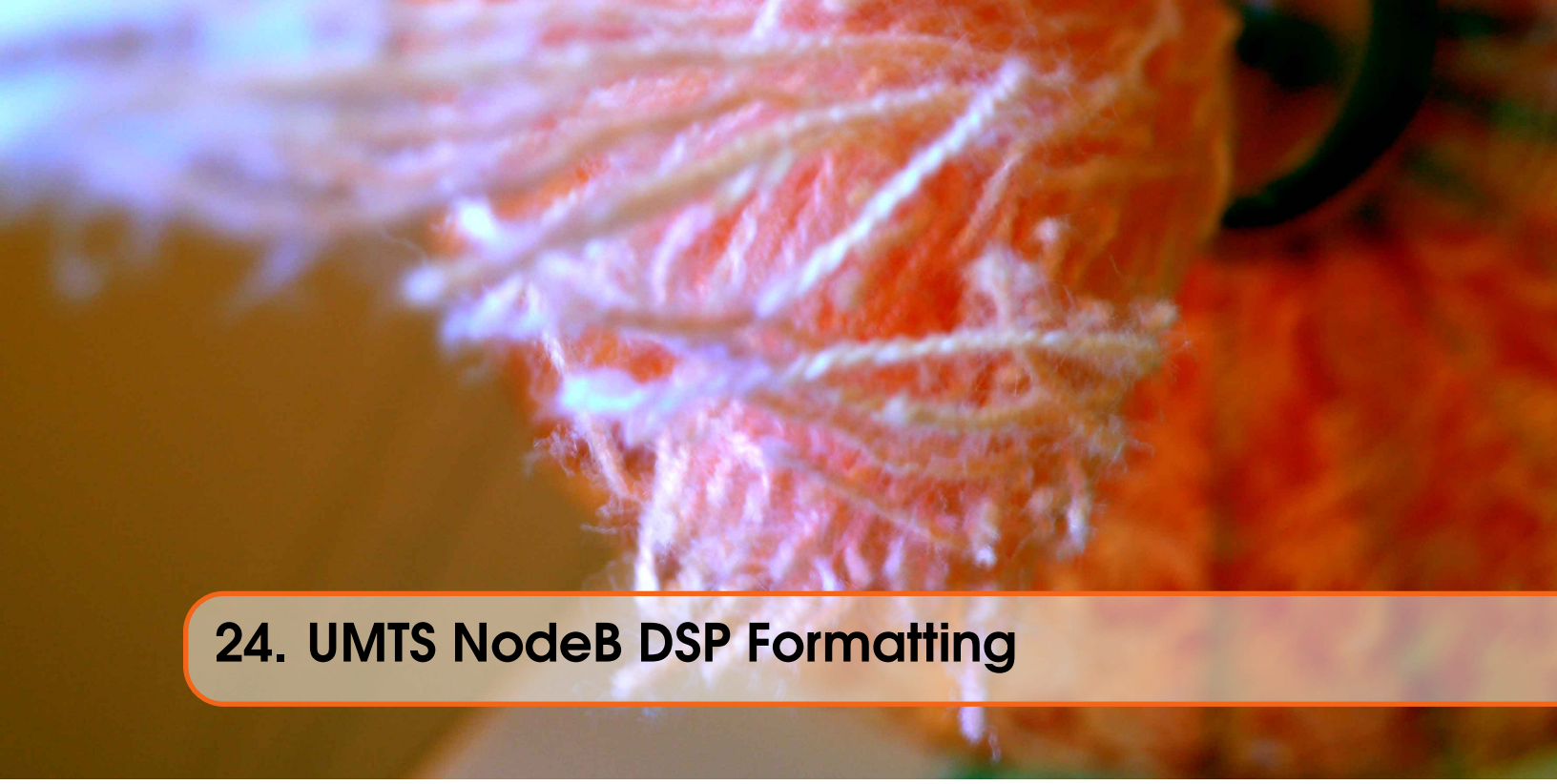
[14]:
```
# merge data frame
df0=pd.merge(df,trxcount,on=['BSCName'])
```

23.1.8 Export Final Data Set

[15]:
```
# export
df0.to_csv('DSP_2G_output.csv',index=False)
```

23.1.9 Output File Format

- Output File Format



24. UMTS NodeB DSP Formatting

24.1 UMTS DSP NodeB File

24.1.1 Input File Format

- DSP File NodeB Level

24.1.2 Input File Format

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
from glob import glob
```

24.1.3 Set Working Path

[2]:
```
working_directory = 'D:/DataSets/DSP/3G_DSP_NodeB'
os.chdir(working_directory)
```

24.1.4 Import 3G NodeB Level DSP File

[3]:
```
df=pd.read_fwf('DSP_NodeB.txt',\
               colspecs = [(0,500),(0,15),(16,30),(31,50),(51,220)],\
               names=['NodeName',\
                'Operator Index',\
                'Operator Name',\
                'License Identifier',\
                'License Item_Allocated_Expiration Date'],\
               comment='+++')
```

24.1.5 Data Pre-Processing

[4]:
```
# get Node Name
df.NodeName = df.NodeName.where(df.NodeName.str.startswith('3G-')).ffill()
```

[5]:
```
#Filter only Requied Data
df = df[df['Operator Index'] == '65535']
df = df[df['License Identifier'] != 'Unlimited frequency']
```

[6]:
```
#remove multiple spaces (with single steps)
df['License Item_Allocated_Expiration Date']= \
    df['License Item_Allocated_Expiration Date'].\
        str.strip().str.replace('\s\s+', '_')
```

[7]:
```
# split column
df[['License Item', 'Allocated', 'Expiration Date']] = \
        df['License Item_Allocated_Expiration Date']\
        .str.split('_', expand=True)
```

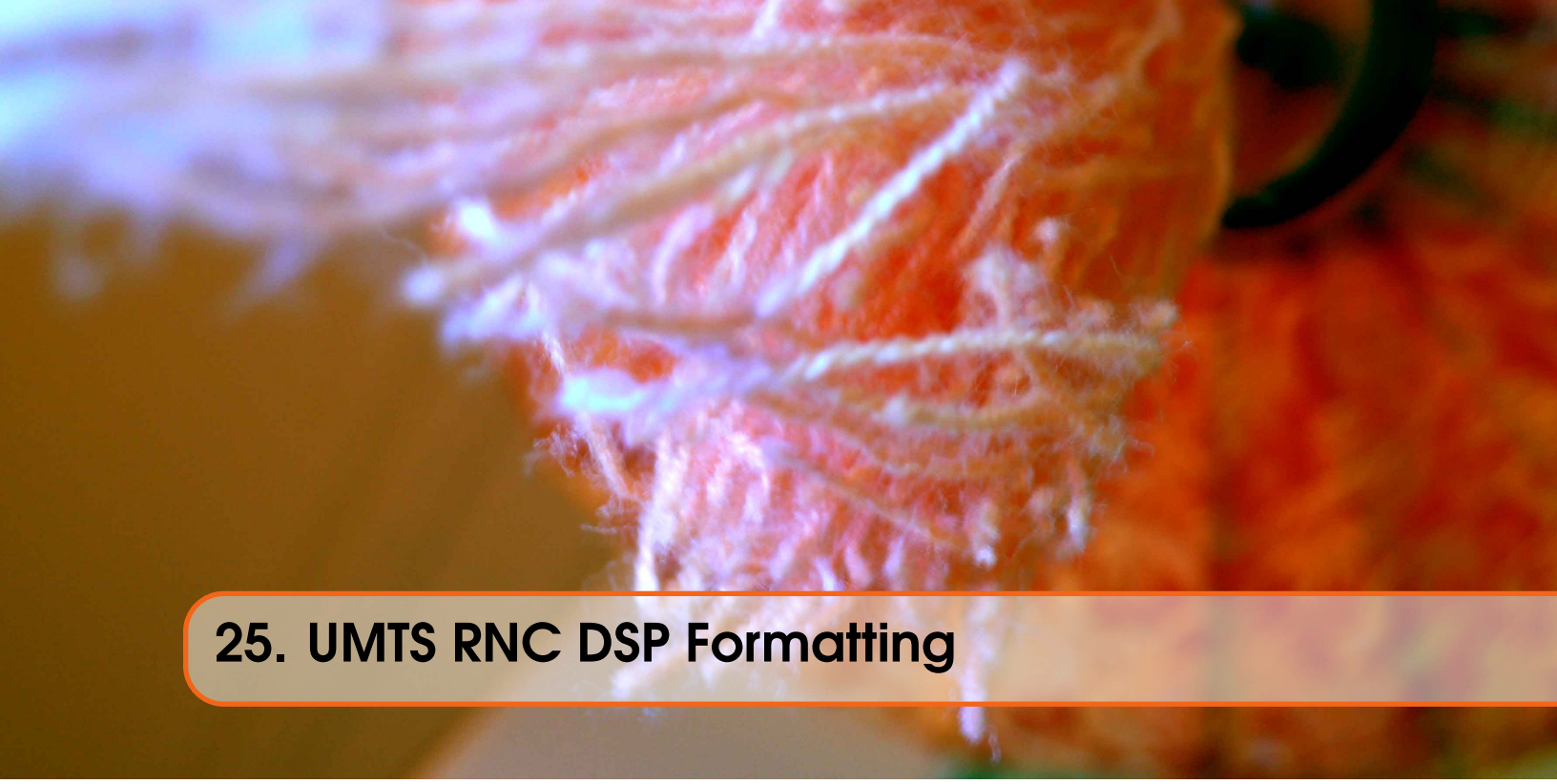
[8]:
```
# drop unreqired column
df=df.drop(['License Item_Allocated_Expiration Date'],axis=1)
```

24.1.6 Export Final Data Set

[9]:
```
df.to_csv('DSP_3GNodeB_output.csv.csv',index=False)
```

24.1.7 Output File Format

- Output File Format



25. UMTS RNC DSP Formatting

25.1 UMTS DSP RNC File

25.1.1 Input File Format

- DSP File RNC Level

25.1.2 Input File Format

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
from glob import glob
```

25.1.3 Set Working Path

[2]:
```
working_directory = 'D:/DataSets/DSP/3G_DSP_RNC'
os.chdir(working_directory)
```

25.1.4 Import 3G RNC Level DSP File

[3]:
```
df=pd.read_fwf('DSP_RNC_3G.txt',\
               colspecs = [(0,20),(0,20),(20,35),(35,55),(55,300)],\
               names=['RNCName','Cn Operator Index', \
                      'Operator Name',\
                      'License Identifier',\
                      'License Item_Allocated_Usage'],\
               comment='+++')
```

25.1.5 Data Pre-Processing

[4]:
```
# get the BSC Name for each row
df.RNCName = df.RNCName.where(df.RNCName.str.contains('RNC')).ffill()
```

[5]:
```
# Condition Filter
df = df[df['Cn Operator Index'] == '0']
```

[6]:
```
# strip
df['License Item_Allocated_Usage']= df['License Item_Allocated_Usage']\
                                .str.strip().str.replace('\s\s+', ';')
```

[7]:
```
# split column
df[['License Item','Allocated','Usage']] = \
        df['License Item_Allocated_Usage'].str.split(';', expand=True)
```

[8]:
```
# drop unreqired column
df=df.drop(['License Item_Allocated_Usage'],axis=1)
# df['RNCName'] = df['RNCName'].str.replace("NE : ","")
```

25.1.6 Export Final Data Set

[9]:
```
df.to_csv('DSP_3GRNC_output.csv',index=False)
```

25.1.7 Output File Format

- Output File Format

26. Frequency Export

26.1 Frequency Export For IFOS

26.1.1 Input File Format

- 2G RF Export Frequency Tab

26.1.2 Import required Libraries

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
from glob import glob
```

26.1.3 working path

[2]:
```
folder_path = 'D:/DataSets/RFExport/2GCellFreqExport'
os.chdir(folder_path)
```

26.1.4 GCELL RF Export(Frequency)

[3]:
```
df0 = pd.read_csv('GCELL.txt',header=1,\
    usecols=['BSC Name', 'BTS Name','Cell Name','MCC',
    'MNC', 'Cell LAC', 'Cell CI', 'NCC', 'BCC',\
             'active status'])
```

[4]:
```
# Site Name in the Required Format
df0['Site Name(*)'] = np.where(
            (df0['BTS Name'].str.startswith ('CI-')),
            df0['BTS Name'].str[0:7],
            np.where(
                    (df0['BTS Name'].str.startswith ('CII-')),
                    df0['BTS Name'].str[0:8],
                    np.where(
                            (df0['BTS Name'].str.startswith ('S-')),
                             df0['BTS Name'].str[0:6],
                              np.where(
                                      (df0['BTS Name'].str.startswith ('N-')),
                                        df0['BTS Name'].str[0:6],
                                        df0['BTS Name'].str[0:4])))) 
```

[5]:
```
# Site ID in numeric format
df0['Site ID(*)']=df0['Site Name(*)'].str.extract('(\d+)', expand=False)
```

[6]:
```
# Cell Name in the requied format
df0['Cell Name(*)'] = np.where(
            (df0['Cell Name'].str.startswith ('CI-')),
            df0['Cell Name'].str[0:9],
            np.where(
                    (df0['Cell Name'].str.startswith ('CII-')),
                    df0['Cell Name'].str[0:10],
                    np.where(
                            (df0['Cell Name'].str.startswith ('S-')),
                             df0['Cell Name'].str[0:8],
                              np.where(
                                      (df0['Cell Name'].str.startswith ('N-')),
                                        df0['Cell Name'].str[0:8],
                                       df0['Cell Name'].str[0:5])))) 
```

26.1.5 GTRX RF Export(Frequency)

[7]:
```
df1 = pd.read_csv('GTRX.txt',header=1,\
                  usecols=['BSC Name', 'Cell Name','Frequency', \
                           'Is Main BCCH TRX','Active Status'])
```

[8]:
```
# Band Identification
df1['Band'] = np.where(
            ((df1['Frequency']>=25) & (df1['Frequency']<=62)),
            'PTML-GSM-TRX',
            np.where(
                    (df1['Frequency']>=556) & (df1['Frequency']<=585),
                    'PTML-DCS-TRX',
                     'Other-Band-TRX'))
```

[9]:
```
# convert Frequency data Type
df1= df1.astype({"Frequency": str})
```

[10]:
```
# Band wise TRX Count (Per Cell)
df3=pd.crosstab([df1["BSC Name"],df1["Cell Name"]],\
                     df1['Band']).reset_index().fillna(0)
```

[11]:
```
#900 MAIO Calculation
MAIO_900M=[]
for index, row in df3.iterrows():
    MAIO_900M.append(np.arange(0, row['PTML-GSM-TRX']-1, 1).tolist())
df3['MAIO_900M'] =  MAIO_900M
df3['MAIO_900M'] =  pd.DataFrame([str(line).\
                    strip('[').strip(']') \
                    for line in df3['MAIO_900M']])
```

[12]:
```
#1800 MAIO Calculation
MAIO_1800M=[]
for index, row in df3.iterrows():
    if row['PTML-GSM-TRX']==0:
        MAIO_1800M.append(np.arange(0, row['PTML-DCS-TRX']-1, 1).tolist())
    else:
        MAIO_1800M.append(np.arange(0, row['PTML-DCS-TRX'], 1).tolist())
df3['MAIO_1800M'] = MAIO_1800M
df3['MAIO_1800M'] = pd.DataFrame([str(line).\
                        strip('[').strip(']') \
                        for line in df3['MAIO_1800M']])
```

[13]:
```
# Replace , with ; in MAIOs
df3['MAIO_900M'] = df3['MAIO_900M'].str.replace(",",";")
df3['MAIO_1800M'] = df3['MAIO_1800M'].str.replace(",",";")
```

26.1.6 Import GCELLMAGRP

[14]:
```
df2 = pd.read_csv('GCELLMAGRP.txt',header=1)
```

[15]:
```
df2=df2[list(df2.columns[0:2])+list(df2.columns[7:])]
```

[16]:
```
#re-shape data set
df4=pd.melt(df2, \
            id_vars=['BSC Name', 'Cell Name'],\
            value_name='MAL').dropna()\
            .astype({"MAL": int})\
            .astype({"MAL": str})
```

26.1.7 Merge Data Frame

[17]:
```
df5 = df0.merge(df3,on=['BSC Name', 'Cell Name'],how="left").\
      merge(df1.loc[df1['Is Main BCCH TRX']=='YES']\
            [['BSC Name','Cell Name','Frequency']]\
            ,on=['BSC Name','Cell Name'],how="left").\
      merge(df1.loc[df1['Is Main BCCH TRX']=='NO'].\
            groupby(['BSC Name','Cell Name'])['Frequency'].\
            apply(';'.join),on=['BSC Name','Cell Name'],how="left").\
      merge(df4.groupby(['BSC Name','Cell Name'])['MAL'].\
            apply(';'.join),on=['BSC Name','Cell Name'],how="left").\
        rename(columns={"Frequency_x":"BCCH","Frequency_y":"TCH"})
```

26.1.8 Union of the sets

[18]:
```
df5.loc[:,'TCH']  = df5.loc[:,'TCH'].\
                replace(';','_').replace(np.nan,'-1').\
                    str.split(';').\
                    apply(set).\
                apply(lambda x: {int(i) for i in x})

df5.loc[:,'MAL'] =  df5.loc[:,'MAL'].\
                    replace(';','_').replace(np.nan,'-1').\
                    str.split(';').apply(set).\
                    apply(lambda x: {int(i) for i in x})

df5['TCH_MAL'] = df5.apply(lambda x: x['TCH'].union(x['MAL']), axis=1).\
                           apply(lambda x: {int(i) for i in x})
```

26.1.9 Sort the values and get the results in list

[19]:
```
df5.loc[:,'TCH_MAL']  = df5.loc[:,'TCH_MAL'] .apply(lambda x: sorted(x))
df5.loc[:,'MAL']  = df5.loc[:,'MAL'] .apply(lambda x: sorted(x))
df5.loc[:,'TCH']  =df5.loc[:,'TCH'].apply(lambda x: sorted(x))
```

26.1.10 Remove Dummay Value from the list

[20]:
```
df5['TCH_MAL'] = df5['TCH_MAL'].apply(lambda x: [i for i in x if i != -1])
df5['TCH'] = df5['TCH'].apply(lambda x: [i for i in x if i != -1])
df5['MAL'] = df5['MAL'].apply(lambda x: [i for i in x if i != -1])
```

26.1.11 Remove Breakets from the list

[21]:
```
df5['TCH_MAL'] = pd.DataFrame([str(line).strip('[').strip(']') \
                               for line in df5['TCH_MAL']])
df5['TCH']  = pd.DataFrame([str(line).strip('[').strip(']') \
                            for line in df5['TCH']])
df5['MAL'] = pd.DataFrame([str(line).strip('[').strip(']') \
                           for line in df5['MAL']])
```

26.1.12 Replace values in the Frequencies Columns

[22]:
```
df5['TCH_MAL'] = df5['TCH_MAL'].str.replace(",",";").str.replace(" ","")
df5['TCH'] = df5['TCH'].str.replace(",",";").str.replace(" ","")
df5['MAL'] = df5['MAL'].str.replace(",",";").str.replace(" ","")
```

26.1.13 Export Final Data Set

[23]:
```
df5.to_csv('Frequency_Export2.csv',index=False)
```

26.1.14 Output File Format

- Output File Format

27. GSM RF Export Parameter Utilization

27.1 RF Export Values Utilization

27.1.1 Input File Format

- 2G RF Export Cell Level Parameters

27.1.2 Import required Libraries

[1]:
```
import os
import pandas as pd
```

27.1.3 Import RF Exort (All Files Except PTPBVC)

[2]:
```
# working path
folder_path = 'D:/DataSets/RFExport/2GCellParamExport/Post'
os.chdir(folder_path)
```

[3]:
```
# Import All files in different Variables
filelist = [file for file in os.listdir(folder_path)]
database = {}
for file in filelist:
    database[file] = pd.read_csv(file,\
                     low_memory=False,header=0,\
                     skiprows=[1]).fillna('-')
```

[4]:
```
# re-shapre (melt) and concat all the data frames
dframe = []
for key in database:
    df=pd.melt(database[key],\
    id_vars=['BSCName', 'CELLNAME', 'CELLID'],\
    var_name="Parameter", value_name='Parameter-Value')
    df['FileName']= key[:-4]
    dframe.append(df)
result= pd.concat(dframe)
```

[5]:
```
#ignore few Parameters
aa= result[~result['Parameter'].\
           isin(['BTSID', 'BTSNAME','CI','GLOCELLID','REMARK',\
            'ADMSTAT','OPNAME','BROADCASTCONTENT'])]
```

27.1.4 Value counts For each Parameter Values

[6]:
```
df_value_counts= pd.DataFrame(aa.groupby(['Parameter','FileName'])\
                    ['Parameter-Value'].\
                    value_counts().\
                    reset_index(name='count'))
```

27.1.5 Export Data Set (Count)

[7]:
```
df_value_counts.to_csv('2G_RF_Export_GCELL_Audit.csv',index=False)
```

27.1.6 Adjact the value of count as per requirement

[8]:
```
df_value_counts1 = (df_value_counts[df_value_counts['count'] == 1]).\
                    reset_index(drop=True)
```

27.1.7 Discrepancy cell list

[9]:
```
fds = pd.merge(df_value_counts1,\
               aa,\
    on=['Parameter','Parameter-Value','FileName'],how='left')
```

27.1.8 Required sequence

[10]:
```
fds = fds[['BSCName','CELLNAME',\
           'CELLID','FileName',\
           'Parameter','Parameter-Value']]
```

27.1.9 Export Data Set (discrepancy cell list)

[11]:
```
fds.to_csv('Discrepancy_Cell_List.csv',index=False)
```

27.1.10 Output File Format

- Output File Format

28. Pre Post GSM RF Export Audit

28.1 Compare Pre and Post GSM RF Export

28.1.1 Input File Format

- 2G RF Export Cell Level Parameters Pre and Post

28.1.2 Import required Libraries

[1]:
```
import os
import pandas as pd
```

28.1.3 Import Post RF Exort (All Files Except PTPBVC)

[2]:
```
# working path
folder_path = 'D:/DataSets/RFExport/2GCellParamExport/Post'
os.chdir(folder_path)
```

[3]:
```
# Import All files in different Variables
filelist = [file for file in os.listdir(folder_path)]
database = {}
for file in filelist:
    database[file] = pd.read_csv(file,low_memory=False,\
                     header=0,skiprows=[1]).fillna('-')
```

[4]:
```
# re-shapre (melt) and concat all the data frames
dframe = []
for key in database:
    df=pd.melt(database[key],\
    id_vars=['BSCName', 'CELLNAME', 'CELLID'],\
    var_name="Parameter", value_name='Parameter-Value-Post')
    df['FileName']= key[:-4]
    dframe.append(df)
result= pd.concat(dframe)
```

[5]:
```
#ignore few Parameters
aa= result[~result['Parameter'].\
        isin(['BTSID', 'BTSNAME','CI','GLOCELLID','REMARK',\
        'ADMSTAT','OPNAME','BROADCASTCONTENT'])]
```

28.1.4 Import Pre RF Exort (All Files Except PTPBVC)

[6]:
```
# working path
folder_path = 'D:/DataSets/RFExport/2GCellParamExport/Pre'
os.chdir(folder_path)
```

[7]:
```
# Import All files in different Variables
filelist1 = [file for file in os.listdir(folder_path)]
database1 = {}
for file in filelist1:
    database1[file] = pd.read_csv(file,low_memory=False,\
                    header=0,skiprows=[1]).fillna('-')
```

[8]:
```
# re-shapre (melt) and concat all the data frames
dframe1 = []
for key in database1:
    df1=pd.melt(database1[key],\
    id_vars=['BSCName', 'CELLNAME', 'CELLID'],\
    var_name="Parameter", value_name='Parameter-Value-Pre')
    df1['FileName']= key[:-4]
    dframe1.append(df1)
result1= pd.concat(dframe1)
```

[9]:
```
#ignore few Parameters
bb= result1[~result1['Parameter'].\
        isin(['BTSID', 'BTSNAME','CI','GLOCELLID','REMARK',\
                'ADMSTAT','OPNAME','BROADCASTCONTENT'])]
```

28.1.5 Compare Pre and Post RF Export

[10]:
```
# merge pre and post RF Export
df_tett=pd.merge(aa,\
                 bb,\
        on=['BSCName', 'CELLID', 'Parameter','FileName'],how="left")
```

[11]:
```
# compare pre and post values
df_tett['comp']\
    =df_tett['Parameter-Value-Post'].astype(str) \
    == df_tett['Parameter-Value-Pre'].astype(str)
```

[12]:
```
# Filter only False Values
audit = df_tett[df_tett.comp==False]
```

28.1.6 Export Final Data Set

[13]:
```
# set working path
folder_path = 'D:/DataSets/RFExport'
os.chdir(folder_path)
# export Audit
audit.to_csv('2G_Export_Audit.csv',index=False)
```

28.1.7 Output File Format

- Output File Format

29. UMTS RF Export Audit

29.1 3G RF Export Audit

29.1.1 Input File Format

- 3G RF Export Cell Level Parameters
- 3G RF Export must be in .txt Format.
- Following files are exclude during the parameter Audit.
  - CELLRLPWR.txt,
  - FACH.txt,
  - FACHDYNTFS.txt,
  - PRACHTFC.txt,
  - RACHDYNTFS.txt,
  - SCCPCHBASIC.txt,
  - SCCPCHTFC.txt

29.1.2 Import required Libraries

[1]:
```
import os
import pandas as pd
```

29.1.3 Set Working Path

[2]:
```
# working path
folder_path = 'D:/DataSets/RFExport/3GRFExport/3G Export/3GCellParamExport'
os.chdir(folder_path)
```

29.1.4 Import RF Exort (GCELL)

[3]:
```
# Import All files in different Variables
filelist = [file for file in os.listdir(folder_path)]
database = {}
for file in filelist:
    database[file] = pd.read_csv(file,low_memory=False,\
                     header=0,skiprows=[1]).fillna('-')
```

[4]:
```
# re-shapre (melt) and concat all the data frames
dframe = []
for key in database:
    df=pd.melt(database[key],\
    id_vars=['BSCName', 'CELLNAME', 'CELLID'],\
    var_name="Parameter", value_name='Parameter-Value')
    df['FileName']= key[:-4]
    dframe.append(df)
result= pd.concat(dframe)
```

[5]:
```
#ignore few Files
aa= result[~result['FileName'].\
    isin(['CELLRLPWR','FACH','FACHDYNTFS','PRACHTFC',\
          'RACHDYNTFS','SCCPCHBASIC','SCCPCHTFC'])]
```

[6]:
```
#ignore few Parameters
bb= aa[~aa['Parameter'].isin(['NODEBNAME'])]
```

29.1.5 Band Identification

[7]:
```
cc = pd.merge(bb,\
              database['CELL.txt'][['BSCName','CELLID','BANDIND']]\
              ,on=['BSCName','CELLID'],how='left')
```

29.1.6 Value counts For each Parameter Values

[8]:
```
df_value_counts= pd.DataFrame(cc.groupby(['Parameter','FileName','BANDIND'])\
                              ['Parameter-Value'].value_counts().
 ↪reset_index(name='count'))
```

29.1.7 Export Final Data Set

[9]:
```
df_value_counts.to_csv('3G_RF_Export_GCELL_Audit.csv',index=False)
```

29.1.8 Output File Format

- Output File Format

30. Mege ZTe UMTS RF Exports

30.1 ZTe RF Export

30.1.1 Input File Format

- 3G RF Export

30.1.2 Import required Libraries

[1]:
```
import os
import glob
import pandas as pd
```

30.1.3 Set Working Path

[2]:
```
path = 'D:/DataSets/RFExport/Zexport'
all_csv = glob.glob(path+"/*/*.xlsm",recursive=True)
```

30.1.4 Concat All the Files

[3]:
```
%%%time
sheets = pd.ExcelFile(all_csv[0]).sheet_names
dfs = {s: pd.concat(pd.read_excel(f, sheet_name=s,header=0,skiprows=[1,2,3,4]) \
                                    for f in all_csv) for s in sheets}
```

```
Wall time: 11min 24s
```

30.1.5 Export Final Data Set

[4]:
```
working_directory = 'D:/DataSets/RFExport/Zexport'
os.chdir(working_directory)
```

[6]:
```
writer = pd.ExcelWriter('Zexport_data.xlsx')
for key in dfs:
        dfs[key].to_excel(writer, key,index=False)
writer.save()
```

30.1.6 Output File Format

- Output File Format

31. Miscellaneous Operations

31.1 Percentage Number Handling in Pandas

31.1.1 Import required Libraries

[1]:
```
import os
import pandas as pd
```

31.1.2 Set Working Path

[2]:
```
working_directory = 'D:/DataSets/RFExport/MAMO'
os.chdir(working_directory)
```

31.1.3 Input File Format

- MIMO Data Set

31.1.4 Import Data Set

[3]:
```
df = pd.read_csv('MIMO.csv',header=5)
```

31.1.5 Check the Data Tpyes of each column

[4]:
```
df.dtypes
```

[4]:
```
Index                                    int64
Start Time                              object
End Time                                object
Query Granularity                       object
Subnetwork                               int64
Subnetwork Name                         object
ManagedElement                           int64
ManagedElement Name                     object
Cell                                     int64
Cell Name                               object
eNodeB                                   int64
eNodeB Name                             object
Product                                 object
Online Number of RRC Connection User     int64
UL Volume (GB)                         float64
DL Volume (GB)                         float64
DL User Throughput (Mbps)              float64
UL User Throughput (Mbps)              float64
RANK=2 Ratio                            object
Number of RRC Establishment Attempt      int64
RRC Successful Establishment Number      int64
dtype: object
```

31.1.6 'RANK=2 Ratio' variable data type is object, we have to convert into float

[5]:
```
df['RANK=2 Ratio'] = df['RANK=2 Ratio'].\
                     str.rstrip('%').\
                     str.replace(',','').\
                     astype('float')
```

31.1.7 Conditional Filtering

[6]:
```
df0 = df[df['RANK=2 Ratio']<5]
```

31.1.8 Export Final Data Set

[7]:
```
df0.to_csv('MIMO_Output.csv',index=False)
```

31.1.9 Output File Format

- Output File Format

31.2 Conditional Filtering in Python list using regex

31.2.1 Import Required Libraries

[8]:
```
import os
import re
import pandas as pd
```

31.2.2 Set Working Path

[9]:
```
working_directory = 'D:/DataSets/RFExport/MAMO'
os.chdir(working_directory)
```

31.2.3 Input File Format

- Degraded KPIs File

31.2.4 Import Data Set

[10]:
```
deg= pd.read_excel('Degraded_KPI_Extract.xlsx')
```

31.2.5 Data Pre-Processing

[11]:
```
# Data Pre-Processing replace ,, values with .
deg['Notes']=deg['Notes'].str.replace(',,','.')
```

31.2.6 Convert the Notes variable to list

[12]:
```
deg.loc[:,'Notes']  = deg.loc[:,'Notes'].str.split('.').apply(list)
```

31.2.7 Conditional Filtering in Python list using regex

[13]:
```
deg['KPI_List'] = deg['Notes'].\
    apply(lambda x: [x for x in x if re.search('Degraded KPI', x)])
```

31.2.8 Formatting For Output

[14]:
```
deg['KPI_List'] =  pd.DataFrame([str(line).strip('[').strip(']')\
                    for line in deg['KPI_List']])
deg['KPI_List'] = deg['KPI_List'].str.replace("' Degraded KPI =",'')
deg['KPI_List'] = deg['KPI_List'].str.replace("' Degraded KPI=",'')
deg['KPI_List'] = deg['KPI_List'].str.replace("'",'')
deg['KPI_List'] = deg['KPI_List'].str.strip()
```

31.2.9 Export Final Data Set

[15]:
```
deg.to_csv('degraded_kpis_output.csv',index=False)
```

31.2.10 Output File Format

- Output File Format

32. BH KPIs Month Level

32.1 Month/Week Level BH KPIs Calculation

32.1.1 Input File Format

Following *PRS Report* use to prepare the Cell On Cluster Busy Hour data;

- Cluster BH Report
- Input File must be *.zip* and *.csv* Format, *Date and Time must be in different columns*

32.1.2 Import Libraries

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
from glob import glob
from collections import ChainMap
```

32.1.3 Set Working Path

[2]:
```
folder_path = 'D:/DataSets/Conformance/Quaterly Conformance Working'
os.chdir(folder_path)
```

32.1.4 Unzip Files

[3]:
```
for file in os.listdir(folder_path):   # get the list of files
    if zipfile.is_zipfile(file): # if it is a zipfile, extract it
        with zipfile.ZipFile(file) as item: # treat the file as a zip
           item.extractall()  # extract it in the working directory
```

32.1.5 List the Files in the Path

[4]:
```
busy_hour_files = sorted(glob('*.csv'))
busy_hour_files
```

[4]:
```
['01_Umer Saeed_Cluster_BH_28042021.csv',
 '02_Umer Saeed_Cluster_BH_19052021.csv',
 '03_Umer Saeed_Cluster_BH_06062021.csv',
 '04_Umer Saeed_Cluster_BH_26062021.csv']
```

32.1.6 Concat All the csv Files

[5]:
```
# concat all the Cluster BH Files
cluster_bh=pd.concat((pd.read_csv(file,skiprows=[0,1,2,3,4],\
            skipfooter=1,engine='python',\
            usecols=['Date','Time','GCell Group',\
            '_CallSetup TCH GOS(%)_D','_CallSetup TCH GOS(%)_N',\
            '_GOS-SDCCH(%)_D','_GOS-SDCCH(%)_N',\
            '_Mobility TCH GOS(%)_D','_Mobility TCH GOS(%)_N','GlobelTraffic'],\
            parse_dates=["Date"]) for file in busy_hour_files)).
 ↪sort_values('Date')
```

32.1.7 Delete csv File from the Path

[6]:
```
for filename in os.listdir(folder_path):
    if filename.endswith('.csv'):
        os.unlink(os.path.join(folder_path, filename))
```

32.1.8 Find Month and Year From Date

[7]:
```
# Get Month number
cluster_bh['Month']=cluster_bh['Date'].dt.month.astype(str) +"_"+␣
 ↪cluster_bh['Date'].dt.year.astype(str)
# Get Week number
#cluster_bh['Month']=cluster_bh['Date'].dt.week.astype(str) +"_"+␣
 ↪cluster_bh['Date'].dt.year.astype(str)
```

32.1.9 Sum of Counters on Month Level

[8]:
```
df_sum = cluster_bh.groupby(['Month','GCell Group'])\
        [['_CallSetup TCH GOS(%)_D', '_CallSetup TCH GOS(%)_N', \
        '_GOS-SDCCH(%)_D','_GOS-SDCCH(%)_N',\
        '_Mobility TCH GOS(%)_D', '_Mobility TCH GOS(%)_N']]\
        .sum().reset_index()
```

32.1.10 Calculate Busy Hour KPIs

[9]:
```
df_sum['CS_TCH_GOS'] = (df_sum['_CallSetup TCH GOS(%)_N']/df_sum['_CallSetup TCH␣
↪GOS(%)_D'])*100
df_sum['MoB_TCH_GOS'] = (df_sum['_Mobility TCH GOS(%)_N']/df_sum['_Mobility TCH␣
↪GOS(%)_D'])*100
df_sum['SDCCH_GOS'] = (df_sum['_GOS-SDCCH(%)_N']/df_sum['_GOS-SDCCH(%)_D'])*100
```

32.1.11 Calculate Average Busy Hour Traffic

[10]:
```
df_tavg =cluster_bh.groupby(['Month','GCell Group']).\
            apply(lambda x: np.average(x['GlobelTraffic']))\
            .reset_index(name='GlobelTraffic_Avg')
```

32.1.12 Final Data Set

[11]:
```
dff = pd.merge(df_sum,df_tavg,on=['Month','GCell Group'])
```

32.1.13 Export Data Set

[12]:
```
dff.to_csv('BH_Monthly_Level_KPIs.csv',index=False)
```

32.1.14 Output File Format

- Output File Format

33. DA KPIs Month Level

33.1 Month/Week Level DA KPIs Calculation

33.1.1 Input File Format

Following *PRS Report* use to prepare the Cell On Cluster Busy Hour data;

- Cluster DA Report
- Input File must be *.zip* and *.csv* Format, *Date and Time must be in different columns*

33.1.2 Import Libraries

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
from glob import glob
from collections import ChainMap
```

33.1.3 Set Working Path

[2]:
```
folder_path = 'D:/DataSets/KPIs/Cluster_DA'
os.chdir(folder_path)
```

33.1.4 Unzip Files

[3]:
```
for file in os.listdir(folder_path):   # get the list of files
    if zipfile.is_zipfile(file): # if it is a zipfile, extract it
        with zipfile.ZipFile(file) as item: # treat the file as a zip
           item.extractall()  # extract it in the working directory
```

33.1.5 List the Files in the Path

[4]:
```
da_files = sorted(glob('*.csv'))
da_files
```

[4]:
```
['Umer Saeed_Cluster_DA_01062021-26062021(Subreport 1).csv',
 'Umer Saeed_Cluster_DA_Query_01072020-31052021(Subreport 1).csv']
```

33.1.6 Concat All the csv Files

[5]:
```
# concat all the Cluster DA Files
cluster_da=pd.concat((pd.read_csv(file,skiprows=[0,1,2,3,4],\
        skipfooter=1,engine='python',\
        usecols=['Date','GCell Group',\
        '_DCR_D','_DCR_N',\
        '_RxQual Index DL_1','_RxQual Index DL_2',\
        '_RxQual Index UL_1','_RxQual Index UL_2',\
        '_HSR%_D','_HSR%_N',\
        'CSSR_Non Blocking_1_N','CSSR_Non Blocking_1_D',\
        'CSSR_Non Blocking_2_N','CSSR_Non Blocking_2_D',
        'TCH Availability Rate(%)','GlobelTraffic'],\
        parse_dates=["Date"]) for file in da_files),ignore_index=False).
 ↪sort_values('Date')
```

33.1.7 Delete csv File from the Path

[6]:
```
for filename in os.listdir(folder_path):
    if filename.endswith('.csv'):
        os.unlink(os.path.join(folder_path, filename))
```

33.1.8 Find Month and Year From Date

[7]:
```
# Get Month number
cluster_da['Month']=cluster_da['Date'].dt.month.astype(str) +"_"+␣
 ↪cluster_da['Date'].dt.year.astype(str)
# Get Week number
#cluster_da['Month']=cluster_da['Date'].dt.week.astype(str) +"_"+␣
 ↪cluster_da['Date'].dt.year.astype(str)
```

33.1.9 Sum of Counters on Month Level

[8]:
```
df_sum = cluster_da.groupby(['Month','GCell Group'])\
        [['_DCR_D', '_DCR_N',
        '_RxQual Index DL_1', '_RxQual Index DL_2',
        '_RxQual Index UL_1', '_RxQual Index UL_2',
        '_HSR%_D', '_HSR%_N',
        'CSSR_Non Blocking_1_N', 'CSSR_Non Blocking_2_N',
        'CSSR_Non Blocking_1_D', 'CSSR_Non Blocking_2_D']].sum().reset_index()
```

33.1.10 Calculate Day Average KPIs

[9]:
```
df_sum['DCR'] = (df_sum['_DCR_N']/df_sum['_DCR_D'])*100
df_sum['HSR'] = (df_sum['_HSR%_N']/df_sum['_HSR%_D'])*100
df_sum['RxQual Index DL'] = (df_sum['_RxQual Index DL_1']/df_sum['_RxQual Index␣
↪DL_2'])*100
df_sum['RxQual Index UL'] = (df_sum['_RxQual Index UL_1']/df_sum['_RxQual Index␣
↪UL_2'])*100
df_sum['Call Setup Success Rate'] =  (1-(df_sum['CSSR_Non Blocking_1_N']/
↪df_sum['CSSR_Non Blocking_1_D']))*\
            (1-(df_sum['CSSR_Non Blocking_2_N']/df_sum['CSSR_Non␣
↪Blocking_2_D']))*100
```

33.1.11 Calculate Aveage DA Traffic ant TCH Availability

[10]:
```
df_tavg = cluster_da.groupby(['Month','GCell Group']).\
            apply(lambda x: pd.Series([np.average(x['GlobelTraffic']),
            np.average(x['TCH Availability Rate(%)'])],
            index=['Avg_GTC','Avg_TCH_Ava_Rate'])).\
            reset_index()
```

33.1.12 Final Data Set

[11]:
```
dff = pd.merge(df_sum,df_tavg,on=['Month','GCell Group'])
```

33.1.13 Export Data Set

[12]:
```
dff.to_csv('DA_Monthly_Level_KPIs.csv',index=False)
```

33.1.14 Output File Format

- Output File Format

34. 2G RF Utilization

34.1 Calculation For 2G RF Utilization Cell and Network Level

34.1.1 Input File Format

Following *PRS Report* use to prepare the Cell On Cluster Busy Hour data;

- Cell Hourly KPIs
- Input File must be *.zip* and *.csv* Format, *Date and Time must be in different columns*

34.1.2 Import Libraries

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
from glob import glob
```

34.1.3 Set Working Path

[2]:
```
folder_path = 'D:/DataSets/KPIs/2G_RF_Utilization'
os.chdir(folder_path)
```

34.1.4 Import Erlang B Table

[3]:
```
df = pd.read_html('https://github.com/Umersaeed81/
 ↪KPI_Data_Set_For_Python_Scripts/blob/main/erlang_b.csv')
df=df[0]
df=df.iloc[:,1:].\
    astype({'No of Trunks (N)':'str'}).\
    rename(columns={"with 2% Blocking":"Offer Traffic"})
```

34.1.5 Unzip Files

[4]:
```
for file in os.listdir(folder_path):   # get the list of files
    if zipfile.is_zipfile(file): # if it is a zipfile, extract it
        with zipfile.ZipFile(file) as item: # treat the file as a zip
           item.extractall()  # extract it in the working directory
```

34.1.6 List the Files in the Path

[5]:
```
da_files = sorted(glob('*.csv'))
da_files
```

[5]:
```
['GoS_Cell_Hourly_27062021(Cell Hourly).csv',
 'GoS_Cell_Hourly_28062021(Cell Hourly).csv',
 'GoS_Cell_Hourly_29062021(Cell Hourly).csv']
```

34.1.7 Concat All the csv Files

[6]:
```
# concat all the Cluster DA Files
cluster_da=pd.concat((pd.read_csv(file,skiprows=[0,1,2,3,4],\
        skipfooter=1,engine='python',\
        parse_dates=["Date"],\
        na_values=['NIL','/0']) for file in da_files),\
        ignore_index=False).sort_values('Date').fillna(0)
```

34.1.8 Delete csv File from the Path

[7]:
```
for filename in os.listdir(folder_path):
    if filename.endswith('.csv'):
        os.unlink(os.path.join(folder_path, filename))
```

34.1.9 Calculate FR and HR Traffic Share

[8]:
```
cluster_da['FR_Share(%)'] = (cluster_da['U_EFR TRAFFIC']/
 ↪cluster_da['GlobelTraffic'])*100
cluster_da['HR_Share(%)'] = (cluster_da['U_AMR HR TRAFFIC']/
 ↪cluster_da['GlobelTraffic'])*100
```

34.1.10 Convert K3015 Counter from float to integer

[9]:
```
cluster_da['No of Trunks (N)'] = cluster_da['K3015:Available TCHs']\
                                 .apply(np.floor)\
                                 .astype(int)\
                                 .astype(str)
```

34.1.11 Calculate Offer Traffic Per Cell/Hour

[10]:
```
df0 = pd.merge(cluster_da,df,on=['No of Trunks (N)'])
```

34.1.12 Calculate 2G RF Utilization (Cell Hourly)

[11]:
```
df0['2G RF Utilization'] = df0['GlobelTraffic'].\
                           div(df0['Offer Traffic'])*100
```

34.1.13 Calculate 2G RF Utilization (Cell Busy Hour)

[12]:
```
df_cell_bh=df0.loc[df0.groupby(['Date','GBSC','Cell CI'])\
                   ['GlobelTraffic'].idxmax()]
```

34.1.14 Sum Network Level Traffic and Offer Traffic

[13]:
```
df1 = df0.groupby(['Date','Time'])\
        [['GlobelTraffic','Offer Traffic']].\
        sum().reset_index()
```

34.1.15 Calculation 2G RF Utilization(Network Level Hourly)

[14]:
```
df1['2G RF Utilization'] = df1['GlobelTraffic'].\
                           div(df1['Offer Traffic'])*100
```

34.1.16 Calculation 2G RF Utilization(Network Level Busy Hour)

[15]:
```
df_n_bh=df1.loc[df1.groupby(['Date'])\
        ['GlobelTraffic'].idxmax()]
```

34.1.17 Export Final Data Set

[16]:
```
df0.to_csv('2G_Cell_Hourly_RF_Utilization.csv',index=False)

df_cell_bh.to_csv('2G_Cell_Busy_Hourly_RF_Utilization.csv',index=False)
df1.to_csv('2G_Network_Hourly_RF_Utilization.csv',index=False)
df_n_bh.to_csv('2G_Network_BH_RF_Utilization.csv',index=False)
```

34.1.18 SLA Target Values

[17]:
```
df_sla=pd.DataFrame({
    'KPI':['GOS-SDCCH(%)','CallSetup TCH GOS(%)','Mobility TCH GOS(%)'],
                'Target Value':[0.1,2,2]})
```

34.1.19 Re-shape Cell Busy Hour Data

[18]:
```
# select required columns
df_cell_bhmr = df_cell_bh[['Date','GBSC','Cell CI',\
                'GOS-SDCCH(%)','CallSetup TCH GOS(%)',\
                'Mobility TCH GOS(%)']]
# re-shpare
df_cell_bhm=pd.melt(df_cell_bhmr,\
    id_vars=['Date', 'GBSC', 'Cell CI'],\
    var_name="KPI", value_name='KPI-Value')

# merge cell busy data and re-shape
df_cell_summary = pd.merge(df_cell_bhm,df_sla,on=['KPI'])
```

34.1.20 Compare KPIs with Target Values

[19]:
```
df_cell_summary['comp']=df_cell_summary['KPI-Value'] >= df_cell_summary['Target␣
 ↪Value']
```

34.1.21 Conditional Pivot table

[20]:
```
kp3=df_cell_summary.loc[df_cell_summary['comp']==True]\
        .pivot_table(index=['Date'],\
        columns='KPI',values=['comp'],\
        aggfunc='count')\
        .fillna(0)\
        .apply(np.int64)\
        .reset_index()
```

34.1.22 Export Summary

[21]:
```
kp3.to_excel('Summary.xlsx')
```

34.1.23 Output File Format

- Output File Format

35. Unzip gz Files in All Sub-directories

35.1 Unzip gz Files in All Sub-directories

35.1.1 Input File Format

- Input Folder, Sub-Folder and Files

35.1.2 Import Required Libraries

[1]:
```
import os
import gzip
import shutil
```

35.1.3 Set Working Path

[2]:
```
path = 'D:/DataSets/KPIs/GZ_Files'
os.chdir(path)
```

35.1.4 Get the List of All the Sub-directories

[3]:
```
gz_files = [os.path.join(root, name)
            for root, dirs, files in os.walk(path)
            for name in files
            if name.endswith((".gz"))]
gz_files
```

[3]:
```
['D:/DataSets/KPIs/GZ_Files\\OB_DO_N_4321_ExpoRMU_D\\BAKDATA20210714162432\\GNBC
FG.XML.gz',
 'D:/DataSets/KPIs/GZ_Files\\OB_DO_N_4322_ExpoRMU_A\\BAKDATA20210714162432\\GNBC
FG.XML.gz',
 'D:/DataSets/KPIs/GZ_Files\\OB_DO_N_4335_ExpoVillageResi3\\BAKDATA2021071416243
3\\GNBCFG.XML.gz',
 'D:/DataSets/KPIs/GZ_Files\\OC_DS_L_443020_RMU1AlWaslPlaza\\BAKDATA202107141624
34\\CFGDATA.XML.gz',
 'D:/DataSets/KPIs/GZ_Files\\OC_DS_L_443021_RMU2AlWaslplaza\\BAKDATA202107141624
34\\CFGDATA.XML.gz']
```

35.1.5 Unzip gz Files in All Sub-directories

[4]:
```
file_type = ".gz"
for fname in gz_files:
    if fname.endswith(file_type):
        with gzip.open(fname,'rb') as f_in:
            with open(fname+'.log','wb') as f_out:
                shutil.copyfileobj(f_in,f_out)
```

35.1.6 Output File Format

- Output Folder, Sub-Folder and Files

36. UMTS High Utilized Cells

36.1 3G High Utilize Cells

36.1.1 Input File Format

- Input Files

36.1.2 Import Required Libraries

[1]:
```
import os
import zipfile
import numpy as np
import pandas as pd
```

36.1.3 working path

[2]:
```
folder_path = 'D:/DataSets/KPIs/3G_Utilization'
os.chdir(folder_path)
```

36.1.4 Import All the Excel Sheets and Tabs

[3]:
```
#all_files = glob('*.xlsx')
# All the Excel Files Nmae (in a directory) in a list
filelist = [file for file in os.listdir(folder_path)]
filelist
```

[3]:
```
['Ufone Regional KPIs_3G_1.xlsx', 'Ufone Regional KPIs_3G_2.xlsx']
```

[4]:
```
# get the sheet name in file 0 index
sheets = pd.ExcelFile(filelist[0]).sheet_names
sheets
```

[4]:
```
['South', 'North', 'Central']
```

[5]:
```
# concat All Files in a dict (tab name become a key), also Assign Sheet name
database = {}
for file in filelist:
    database[file] = pd.concat([pd.read_excel(file,header=0, sheet_name=s,\
                            na_values=['NIL','/0'],dtype = {"Cell ID" : "str"},\
                            converters={'Integrity': lambda value: '{:,.0f}%'.
 ↪format(value * 100)})\
                            .assign(Region=s) for s in sheets])
```

[6]:
```
# concat
df_list = [v for k,v in database.items()]
df = pd.concat(df_list ,axis=0)
```

36.1.5 Calculate4 UL and DL Traffic Volumne

[7]:
```
df['DL Traffic Volume'] = (df['HSDPA RLC Traffic Volume (MB)']+df['R99 PS DL␣
 ↪Traffic Volume (MB)'])/1024
df['UL Traffic Volume'] = (df['HSUPA RLC Traffic Volume (MB)']+df['R99 PS UL␣
 ↪Traffic Volume (MB)'])/1024
```

36.1.6 Cell Count

[8]:
```
dfCon_count = df.groupby(['RNC','Cell ID']).\
                agg({'Cell Name': 'value_counts'}).\
                rename(columns={'Cell Name': 'counts'}).\
                reset_index()
```

36.1.7 Calculate Average Value of each KPI

[9]:
```
df_tavg = df.groupby(['RNC','Cell ID']).\
            apply(lambda x: pd.Series([np.average(x['AMR CS Traffic (Erl)']),
            np.average(x['HSDPA RLC Traffic Volume (MB)']),
            np.average(x['HSUPA RLC Traffic Volume (MB)']),
            np.average(x['R99 PS DL Traffic Volume (MB)']),
            np.average(x['R99 PS UL Traffic Volume (MB)']),
            np.average(x['HSDPA User Throughput(kbit/s)']),
            np.average(x['HSUPA User Throughput(kbit/s)']),
            np.average(x['VS.MeanRTWP(dBm)']),
            np.average(x['Number of HSDPA Users (Mean)']),
            np.average(x['Number of HSDPA Users (Max)']),
```

```
            np.average(x['Number of HSUPA Users (Mean)']),
            np.average(x['Number of HSUPA Users (Max)']),
            np.average(x['Total TCP Utilization Rate (Cell-)(%)']),
            np.average(x['HSDPA Throughput (kbps)']),
            np.average(x['HSUPA Throughput (kbps)']),
            np.average(x['DL Traffic Volume']),
            np.average(x['UL Traffic Volume'])],
                index=['AMR_CS_Traffic_Avg','HSDPA_RLC_Traffic_Volume_Avg',
                'HSUPA_RLC_Traffic_Volume_Avg','R99_PS_DL_Traffic_Volume_Avg',
                'R99_PS_UL_Traffic_Volume_Avg','HSDPA_User_Throughput_Avg',
                'HSUPA_User_Throughput_Avg',
                'VS.MeanRTWP(dBm)_Avg','Number_of_HSDPA_Users_(Mean)_Avg',
               ␣
↪'Number_of_HSDPA_Users_(Max)_Avg','Number_of_HSUPA_Users_(Mean)_Avg',
               ␣
↪'Number_of_HSUPA_Users_(Max)_Avg','Total_TCP_Utilization_Rate_(Cell-)_Avg',
                'HSDPA_Throughput_Avg','HSUPA_Throughput_Avg',
                'DL_Traffic_Volume_Avg','UL_Traffic_Volume_Avg'])).\
                reset_index()
```

36.1.8 Final Data Set

[10]:
```
df_fds =df_tavg.merge(dfCon_count,on=['RNC','Cell ID'],how='left').\
        merge(df[['RNC','Cell ID','Region']],on=['RNC','Cell ID'],how='left')
df_fds = df_fds.loc[df_fds.duplicated(),:]
```

36.1.9 Conditional Filtering

[11]:
```
df_fds_c = df_fds[df_fds['counts'].eq(2)]
```

36.1.10 Export Data Set

[12]:
```
df_fds_c.to_csv('Final_Output.csv',index=False)
```

36.1.11 Reference Code (For Testing Only)

[13]:
```
#workbook = pd.ExcelFile('Ufone Regional KPIs_3G_1.xlsx')
#sheets = workbook.sheet_names
#df = pd.concat([pd.read_excel(workbook, sheet_name=s)
#                .assign(sheet_name=s) for s in sheets])
#df.to_csv('test.csv',index=False)
```

36.1.12 Output File Format

- Output File Format