In [ ]:
```
import pandas as pd
import numpy as np
```

Exploratory Data Analysis.

This section aims to familiarise oneself with the data and explore initial insights that will inform further steps in the data analysis. The goal is to understand the dataset, identify the missing values & outliers if any using visual and quantitative methods to get a sense of the story it tells.

In [ ]:
```
import os
os.getcwd()
os.listdir('data')
```

Out[ ]:
```
['Field Descriptions.csv',
 'Field Descriptions.xlsx',
 'fields_description.csv',
 'Week1_challenge_data_source(CSV).csv',
 'Week1_challenge_data_source.xlsx']
```

In [ ]:
```
#import warnings
#warnings.filterwarnings('ignore')
#pd.set_option('max_column', None)
import pandas as pd
import os
#os."getcwd"()
db = pd.read_csv('data/Week1_challenge_data_source(CSV).csv')
db.head() #returns the first 5 rows.
#pd.set_option('display.float_format',) #review decimal places function
```

Out[ ]:

| | Bearer Id | Start | Start ms | End | End ms | Dur. (ms) | IMSI | MSISDN/Number | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1.311448e+19 | 4/4/2019 12:01 | 770.0 | 4/25/2019 14:35 | 662.0 | 1823652.0 | 2.082014e+14 | 3.366496e+10 | 3.5521 |
| **1** | 1.311448e+19 | 4/9/2019 13:04 | 235.0 | 4/25/2019 8:15 | 606.0 | 1365104.0 | 2.082019e+14 | 3.368185e+10 | 3.5794 |
| **2** | 1.311448e+19 | 4/9/2019 17:42 | 1.0 | 4/25/2019 11:58 | 652.0 | 1361762.0 | 2.082003e+14 | 3.376063e+10 | 3.5281 |
| **3** | 1.311448e+19 | 4/10/2019 0:31 | 486.0 | 4/25/2019 7:36 | 171.0 | 1321509.0 | 2.082014e+14 | 3.375034e+10 | 3.5356 |
| **4** | 1.311448e+19 | 4/12/2019 20:10 | 565.0 | 4/25/2019 10:40 | 954.0 | 1089009.0 | 2.082014e+14 | 3.369980e+10 | 3.5407 |

5 rows × 55 columns

Data exploration. Find out the content of the data. The table below shows the description of fields and columns in the data.

In [ ]:
```
#list column names
db.columns.tolist()
```

Out[ ]:
```
['Bearer Id',
```

```
'Start',
'Start ms',
'End',
'End ms',
'Dur. (ms)',
'IMSI',
'MSISDN/Number',
'IMEI',
'Last Location Name',
'Avg RTT DL (ms)',
'Avg RTT UL (ms)',
'Avg Bearer TP DL (kbps)',
'Avg Bearer TP UL (kbps)',
'TCP DL Retrans. Vol (Bytes)',
'TCP UL Retrans. Vol (Bytes)',
'DL TP < 50 Kbps (%)',
'50 Kbps < DL TP < 250 Kbps (%)',
'250 Kbps < DL TP < 1 Mbps (%)',
'DL TP > 1 Mbps (%)',
'UL TP < 10 Kbps (%)',
'10 Kbps < UL TP < 50 Kbps (%)',
'50 Kbps < UL TP < 300 Kbps (%)',
'UL TP > 300 Kbps (%)',
'HTTP DL (Bytes)',
'HTTP UL (Bytes)',
'Activity Duration DL (ms)',
'Activity Duration UL (ms)',
'Dur. (ms).1',
'Handset Manufacturer',
'Handset Type',
'Nb of sec with 125000B < Vol DL',
'Nb of sec with 1250B < Vol UL < 6250B',
'Nb of sec with 31250B < Vol DL < 125000B',
'Nb of sec with 37500B < Vol UL',
'Nb of sec with 6250B < Vol DL < 31250B',
'Nb of sec with 6250B < Vol UL < 37500B',
'Nb of sec with Vol DL < 6250B',
'Nb of sec with Vol UL < 1250B',
'Social Media DL (Bytes)',
'Social Media UL (Bytes)',
'Google DL (Bytes)',
'Google UL (Bytes)',
'Email DL (Bytes)',
'Email UL (Bytes)',
'Youtube DL (Bytes)',
'Youtube UL (Bytes)',
'Netflix DL (Bytes)',
'Netflix UL (Bytes)',
'Gaming DL (Bytes)',
'Gaming UL (Bytes)',
'Other DL (Bytes)',
'Other UL (Bytes)',
'Total UL (Bytes)',
'Total DL (Bytes)']
```

In [ ]:
```
# number of data points
print(f" There are {db.shape[0]} rows and {db.shape[1]} columns")
```

```
There are 150001 rows and 55 columns
```

Handling missing values.

In [ ]:
```
# how many missing values exist or better still what is the % of missing values in the
def percent_missing(df):
```

```
    # Calculate total number of cells in dataframe
    totalCells = np.product(df.shape)

    # Count number of missing values per column
    missingCount = df.isnull().sum()

    # Calculate total number of missing values
    totalMissing = missingCount.sum()

    # Calculate percentage of missing values
    print("TellCo's financial dataset contains", round(((totalMissing/totalCells) * 100

percent_missing(db)
```

```
TellCo's financial dataset contains 12.5 % missing values.
```

In [ ]:
```
# Now which column(s) has missing values
db.isna().sum()
#print (db)
```

Out[ ]:
```
Bearer Id                                  991
Start                                        1
Start ms                                     1
End                                          1
End ms                                       1
Dur. (ms)                                    1
IMSI                                       570
MSISDN/Number                             1066
IMEI                                       572
Last Location Name                        1153
Avg RTT DL (ms)                          27829
Avg RTT UL (ms)                          27812
Avg Bearer TP DL (kbps)                      1
Avg Bearer TP UL (kbps)                      1
TCP DL Retrans. Vol (Bytes)              88146
TCP UL Retrans. Vol (Bytes)              96649
DL TP < 50 Kbps (%)                        754
50 Kbps < DL TP < 250 Kbps (%)             754
250 Kbps < DL TP < 1 Mbps (%)              754
DL TP > 1 Mbps (%)                         754
UL TP < 10 Kbps (%)                        792
10 Kbps < UL TP < 50 Kbps (%)              792
50 Kbps < UL TP < 300 Kbps (%)             792
UL TP > 300 Kbps (%)                       792
HTTP DL (Bytes)                          81474
HTTP UL (Bytes)                          81810
Activity Duration DL (ms)                    1
Activity Duration UL (ms)                    1
Dur. (ms).1                                  1
Handset Manufacturer                       572
Handset Type                               572
Nb of sec with 125000B < Vol DL          97538
Nb of sec with 1250B < Vol UL < 6250B    92894
Nb of sec with 31250B < Vol DL < 125000B 93586
Nb of sec with 37500B < Vol UL          130254
Nb of sec with 6250B < Vol DL < 31250B   88317
Nb of sec with 6250B < Vol UL < 37500B  111843
Nb of sec with Vol DL < 6250B              755
Nb of sec with Vol UL < 1250B              793
Social Media DL (Bytes)                      0
Social Media UL (Bytes)                      0
Google DL (Bytes)                            0
Google UL (Bytes)                            0
Email DL (Bytes)                             0
```

```
Email UL (Bytes)                          0
Youtube DL (Bytes)                        0
Youtube UL (Bytes)                        0
Netflix DL (Bytes)                        0
Netflix UL (Bytes)                        0
Gaming DL (Bytes)                         0
Gaming UL (Bytes)                         0
Other DL (Bytes)                          0
Other UL (Bytes)                          0
Total UL (Bytes)                          1
Total DL (Bytes)                          1
dtype: int64
```

The data above shows the columns and the respective number of missing values. Since the dataset has 150001 rows, we can drop the rows with the missing values, and still remain with a substantial data for our analysis.

In [ ]:
```
#drop rows with entirely missing values.
df= db
df.dropna()
```

Out[ ]:

| | Bearer Id | Start | Start ms | End | End ms | Dur. (ms) | IMSI | MSISDN/Number | |
|---|---|---|---|---|---|---|---|---|---|
| **11** | 1.311448e+19 | 4/15/2019 11:33 | 626.0 | 4/25/2019 18:44 | 542.0 | 889834.0 | 2.082019e+14 | 3.366447e+10 | 8.( |
| **20** | 1.304243e+19 | 4/17/2019 0:35 | 363.0 | 4/25/2019 4:52 | 691.0 | 706649.0 | 2.082010e+14 | 3.365875e+10 | 8.( |
| **68** | 1.304243e+19 | 4/21/2019 10:09 | 1.0 | 4/25/2019 1:39 | 272.0 | 314961.0 | 2.082014e+14 | 3.366274e+10 | 3. |
| **78** | 1.304243e+19 | 4/21/2019 19:35 | 344.0 | 4/25/2019 2:59 | 259.0 | 285833.0 | 2.082018e+14 | 3.366917e+10 | 3. |
| **84** | 1.304243e+19 | 4/22/2019 1:17 | 301.0 | 4/25/2019 1:17 | 938.0 | 259201.0 | 2.082014e+14 | 3.366555e+10 | 8.( |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **149935** | 1.304243e+19 | 4/29/2019 7:28 | 312.0 | 4/30/2019 7:28 | 218.0 | 86399.0 | 2.082014e+14 | 3.366396e+10 | 8.( |
| **149951** | 1.304243e+19 | 4/29/2019 7:28 | 728.0 | 4/30/2019 7:28 | 643.0 | 86399.0 | 2.082014e+14 | 3.366513e+10 | 8.( |
| **149968** | 1.304243e+19 | 4/29/2019 7:28 | 131.0 | 4/30/2019 7:08 | 257.0 | 85215.0 | 2.082014e+14 | 3.365069e+10 | 3. |
| **149973** | 7.277826e+18 | 4/29/2019 7:28 | 548.0 | 4/30/2019 7:28 | 451.0 | 86399.0 | 2.082010e+14 | 3.366856e+10 | 8.( |
| **149990** | 1.304243e+19 | 4/29/2019 7:28 | 438.0 | 4/30/2019 6:46 | 83.0 | 83844.0 | 2.082014e+14 | 3.376127e+10 | 3. |

17558 rows × 55 columns

In [ ]:
```
df_clean = db.drop(['Nb of sec with 6250B < Vol UL < 37500B', 'Nb of sec with 37500B <
df_clean.shape
```

Out[ ]:
```
(150001, 50)
```

In [ ]:
```
# fill missing with ffill method for columns (diag_1, diag_2, diag_3)

def fix_missing_ffill(df, col):
    df[col] = df[col].fillna(method='ffill')
    return df[col]


def fix_missing_bfill(df, col):
    df[col] = df[col].fillna(method='bfill')
    return df[col]

df_clean['HTTP DL (Bytes)'] = fix_missing_ffill(df_clean, 'HTTP DL (Bytes)')
df_clean['HTTP UL (Bytes)'] = fix_missing_ffill(df_clean, 'HTTP UL (Bytes)')
df_clean['Avg RTT DL (ms)'] = fix_missing_ffill(df_clean, 'Avg RTT DL (ms)')
df_clean['Avg RTT DL (ms)'] = fix_missing_ffill(df_clean, 'Avg RTT DL (ms)')
df_clean['TCP DL Retrans. Vol (Bytes)'] = fix_missing_ffill(df_clean, 'TCP DL Retrans.
df_clean['TCP UL Retrans. Vol (Bytes)'] = fix_missing_ffill(df_clean, 'TCP UL Retrans.
df_clean['Handset Type'] = df_clean['Handset Type'].fillna(df_clean['Handset Type'].mod

percent_missing(df_clean)
```

```
TellCo's financial dataset contains 1.72 % missing values.
```

TellCo's financial dataset has reduced from 12% to 1.72%

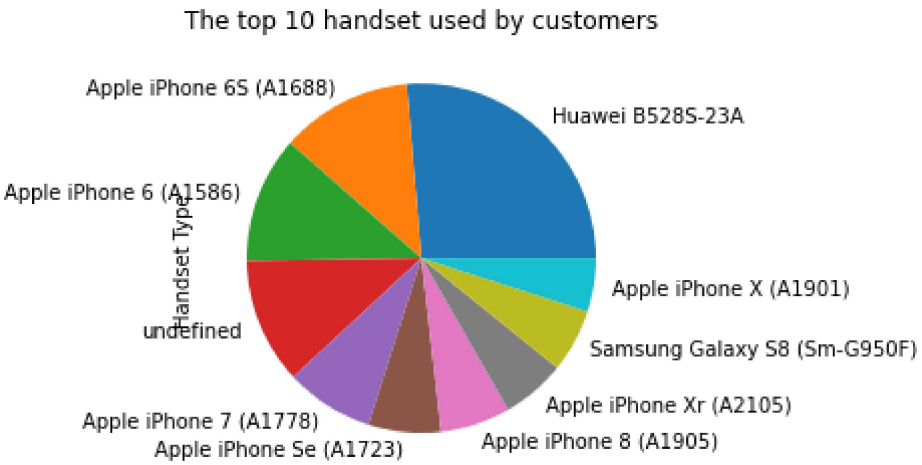
User Overview Analysis

This part aims to identify the top handsets used by the customers, handset manufacturers, and the top handsets per handset manufacturers. Make a short interpretation and recommendation to marketing teams

In [ ]:
```
all = df_clean['Handset Type'].value_counts().head(10)
print(all)
all.plot(kind="pie", title="The top 10 handset used by customers")
```

```
Huawei B528S-23A                20324
Apple iPhone 6S (A1688)          9419
Apple iPhone 6 (A1586)           9023
undefined                        8987
Apple iPhone 7 (A1778)           6326
Apple iPhone Se (A1723)          5187
Apple iPhone 8 (A1905)           4993
Apple iPhone Xr (A2105)          4568
Samsung Galaxy S8 (Sm-G950F)     4520
Apple iPhone X (A1901)           3813
Name: Handset Type, dtype: int64
```

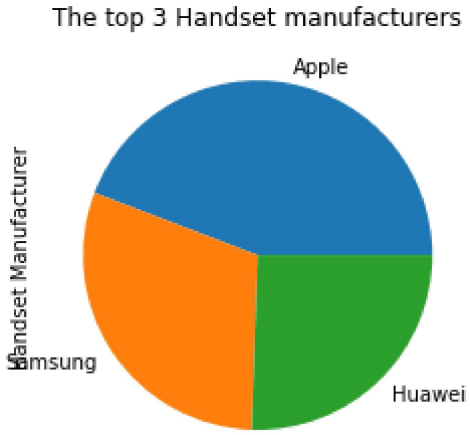
Out[ ]:
```
<AxesSubplot:title={'center':'The top 10 handset used by customers'}, ylabel='Handset Ty
pe'>
```




The top three handset manufacturers

In [ ]:
```
mode= df_clean['Handset Manufacturer'].mode()
df_clean['Handset Manufacturer'].fillna(mode,inplace=True)
x = df_clean['Handset Manufacturer'].value_counts().head(3)
print(x)
x.plot(kind="pie", title="The top 3 Handset manufacturers");
```

```
Apple      59565
Samsung    40839
Huawei     34423
Name: Handset Manufacturer, dtype: int64
```

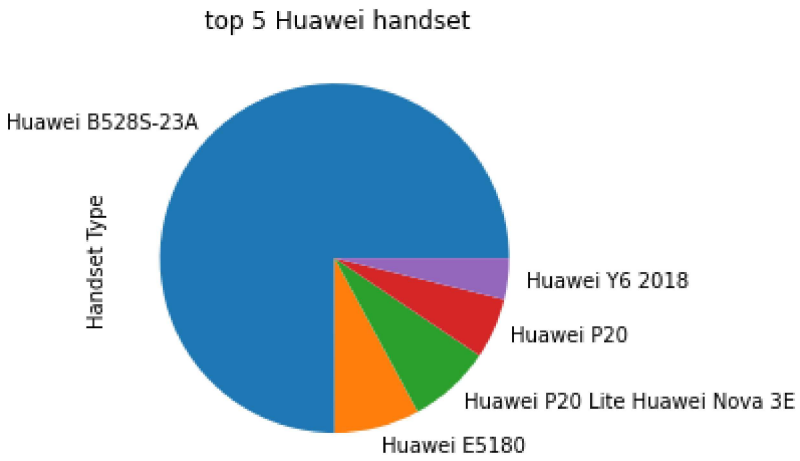



The top 3 handset manufacturers are; Apple, Samsung and Huawei. We are now going to list all the top handset types in the three manufacturers.

In [ ]:
```
#top 5 Huawei handset types
top3 = df_clean.groupby('Handset Manufacturer')['Handset Type'].value_counts()['Huawei'
print(top3)
top3.plot(kind="pie" , title=" top 5 Huawei handset")
```

```
Handset Type
Huawei B528S-23A                19752
Huawei E5180                     2079
Huawei P20 Lite Huawei Nova 3E   2021
Huawei P20                       1480
Huawei Y6 2018                    997
Name: Handset Type, dtype: int64
```

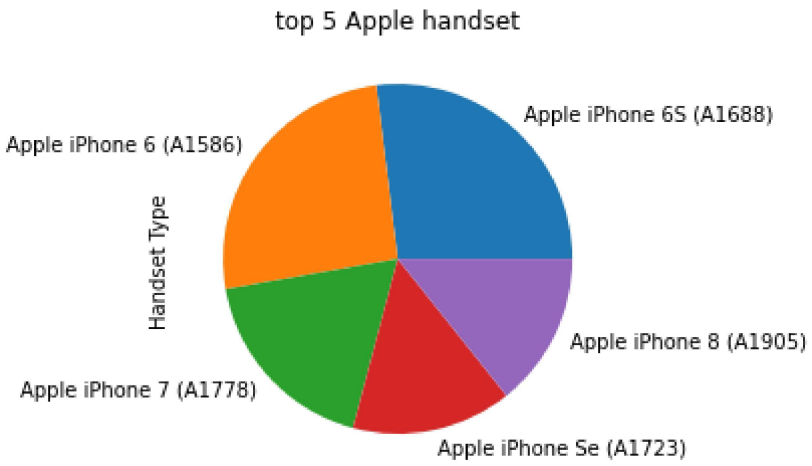
Out[ ]:
```
<AxesSubplot:title={'center':' top 5 Huawei handset'}, ylabel='Handset Type'>
```




In [ ]:
```
#top 5 Apple handset types.
apple = df_clean.groupby('Handset Manufacturer')['Handset Type'].value_counts()['Apple'
print(apple)
apple.plot(kind="pie",  title = 'top 5 Apple handset')
```

```
Handset Type
Apple iPhone 6S (A1688)    9419
Apple iPhone 6 (A1586)     9023
Apple iPhone 7 (A1778)     6326
Apple iPhone Se (A1723)    5187
Apple iPhone 8 (A1905)     4993
Name: Handset Type, dtype: int64
```

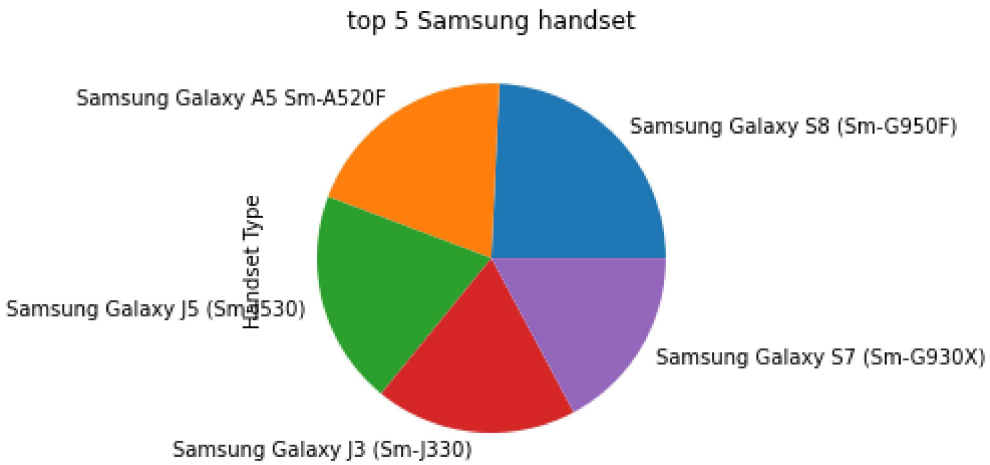
Out[ ]:
```
<AxesSubplot:title={'center':'top 5 Apple handset'}, ylabel='Handset Type'>
```




In [ ]:
```
#top 5 Samsung handset types.
samsung = df_clean.groupby('Handset Manufacturer')['Handset Type'].value_counts()['Sams
print(samsung)
samsung.plot(kind="pie", title = 'top 5 Samsung handset')
```

```
Handset Type
Samsung Galaxy S8 (Sm-G950F)    4520
Samsung Galaxy A5 Sm-A520F      3724
Samsung Galaxy J5 (Sm-J530)     3696
Samsung Galaxy J3 (Sm-J330)     3484
Samsung Galaxy S7 (Sm-G930X)    3199
Name: Handset Type, dtype: int64
```

Out[ ]:
```
<AxesSubplot:title={'center':'top 5 Samsung handset'}, ylabel='Handset Type'>
```




From the output above, we can see the statistics top 10 handsets used by customers. Just from their names, you can tell the type of operating systems used and this can be very helpful to the marketing teams since they can stay connected to consumers through the relevant marketing channels and help develop strategy to engage consumers. An example is through search engine optimization on an easily navigable website.

Another insight we get from the types of top handsets and handset types,is to be able to get a asense of what handsets are suitable for their users, so they can partner with the handset manufacturing company, taking advantage of a phone uptake in the market to push their network as the preferred provider for use with the handset. A real life example is; Customers who buy the Techno Camon 15 automatically get connected with a free SIM card along with 3GB of data for three months from MTN. The Infinix-Airtel partnership is valid on a range of five kinds of phones that include NOTE 7, HOT9 Play, HOT9, S5 Pro and NOTE 7 Lite packaged with a complimentary of 5GB of data for a month. With this partnership, they can also advise the top handset types and whicj one to increase production, and which handsets are fading. Win-win for both.

# Number of xDR sessions

xDR_sessions =

In [ ]:
```
#Check the frequency of user sessions. The results should show a total duration for eac
sessionsCountData=df_clean['MSISDN/Number'].value_counts().head()

sessionsCount=sessionsCountData.values.tolist()

msisdn=sessionsCountData.index.values

sessionPerUserDictionary = dict(zip(msisdn, sessionsCount))

print(sessionPerUserDictionary)
df_clean.groupby('MSISDN/Number')['Dur. (ms)'].sum()
```

```
{33626320676.0: 18, 33625779332.0: 17, 33614892860.0: 17, 33659725664.0: 16, 3367587720
2.0: 15}
```

Out[ ]:
```
MSISDN/Number
3.360100e+10    116720.0
3.360100e+10    181230.0
```

```
3.360100e+10    134969.0
3.360101e+10     49878.0
3.360101e+10     37104.0
                   ...
3.379000e+10      8810.0
3.379000e+10    140988.0
3.197021e+12    877385.0
3.370000e+14    253030.0
8.823971e+14    869844.0
Name: Dur. (ms), Length: 106856, dtype: float64
```

In [ ]:
```
#Calculationg total DL and UL per user, filtered to the first 20.
df_clean = df_clean.rename(columns = {'Total DL (Bytes)' : 'totalDL','Total UL (Bytes)'
sum_column = df_clean["totalUL"] + df_clean["totalDL"]
df_clean["totalData"] = sum_column
totalDataDF=df_clean.groupby('MSISDN/Number')['totalData'].sum()
totalDataValues = totalDataDF.values
msisdn=totalDataDF.index.values

dataPerUser = dict(zip(msisdn, totalDataValues))

dict_items = dataPerUser.items()
first_twenty = list(dict_items)[:20]
first_twenty
```

Out[ ]:
```
[(33601001722.0, 878690574.0),
 (33601001754.0, 156859643.0),
 (33601002511.0, 595966483.0),
 (33601007832.0, 422320698.0),
 (33601008617.0, 1457410944.0),
 (33601010682.0, 615217221.0),
 (33601011634.0, 654723066.0),
 (33601011959.0, 332660357.0),
 (33601014694.0, 990132189.0),
 (33601020306.0, 732463761.0),
 (33601021045.0, 114976036.0),
 (33601021217.0, 630092434.0),
 (33601022743.0, 482419082.0),
 (33601024291.0, 358911900.0),
 (33601025738.0, 230324396.0),
 (33601026147.0, 541779966.0),
 (33601027208.0, 697559667.0),
 (33601031129.0, 637053075.0),
 (33601032846.0, 718452416.0),
 (33601032987.0, 647203715.0)]
```

In [ ]:
```
#The above results show the aggregate data used by each customer on many different sess
df_clean=df_clean.rename(columns = {'Total DL (Bytes)' : 'totalDL','Total UL (Bytes)' :

sum_column = df_clean["totalUL"] + df_clean["totalDL"]


google = df_clean['Google DL (Bytes)']+ df_clean['Google UL (Bytes)']
email = df_clean['Email DL (Bytes)']+ df_clean['Email UL (Bytes)']
gaming = df_clean['Gaming DL (Bytes)']+ df_clean['Gaming UL (Bytes)']
youtube = df_clean['Youtube DL (Bytes)']+ df_clean['Youtube UL (Bytes)']
netflix = df_clean['Netflix DL (Bytes)']+ df_clean['Netflix UL (Bytes)']
social = df_clean['Social Media DL (Bytes)']+ df_clean['Social Media UL (Bytes)']

df_clean['google']=google
df_clean['email']=email
df_clean['gaming']=gaming
```

```
df_clean['youtube']=youtube
df_clean['netflix']=netflix
df_clean['social']=social

relevant_data=df_clean[['msisdn', 'google','email','gaming','youtube','netflix','social
relevant_data["totalData"] = sum_column
relevant_data.groupby('msisdn')['totalData'].sum()
msisdn
```

```
<ipython-input-34-3664385c4bfd>:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  relevant_data["totalData"] = sum_column
```

Out[ ]:
```
array([3.36010017e+10, 3.36010018e+10, 3.36010025e+10, ...,
       3.19702088e+12, 3.37000037e+14, 8.82397108e+14])
```