Consumer & Visitor Insights For Neighborhoods (cbg_patterns)

1. Data Visualization and Data Summary

1.1 data summary

In [1]:
```
import pandas as pd  # data processing, csv file I/O
import numpy as np
import matplotlib.pyplot as plt

# load csv file
data = pd.read_csv('../cbg_patterns.csv')
```

Descriptors of the raw datase

In [2]:
```
data.info()
data_shape = data.shape
print(data_shape)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220735 entries, 0 to 220734
Data columns (total 13 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   census_block_group        220734 non-null  float64
 1   date_range_start          220735 non-null  int64  
 2   date_range_end            220735 non-null  int64  
 3   raw_visit_count           220629 non-null  float64
 4   raw_visitor_count         220629 non-null  float64
 5   visitor_home_cbgs         220735 non-null  object 
 6   visitor_work_cbgs         220735 non-null  object 
 7   distance_from_home        220518 non-null  float64
 8   related_same_day_brand    220735 non-null  object 
 9   related_same_month_brand  220735 non-null  object 
 10  top_brands                220735 non-null  object 
 11  popularity_by_hour        220735 non-null  object 
 12  popularity_by_day         220735 non-null  object 
dtypes: float64(4), int64(2), object(7)
memory usage: 21.9+ MB
(220735, 13)
```

Find indexes of nominal and numerical data

remove the redundent column of row number

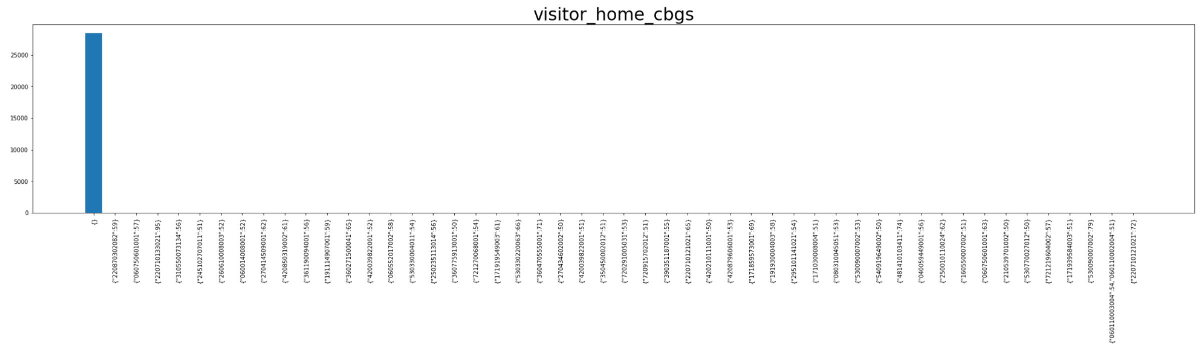
In [3]:
```
# Nominal index
nominal_index = ['visitor_home_cbgs', 'visitor_work_cbgs', 'related_same_day_brand', 'related_same_month_brand', 'top_brands', 'popularity_by_hour', 'popularity_by_day']
# Numerical index
numerical_index = ['census_block_group', 'date_range_start', 'date_range_end', 'raw_visit_count', 'raw_visitor_count', 'distance_from_home']

# Get frequency of each arrtibute
data_frequency = {key: data[key].value_counts() for key in data.columns}
```

Nominal data summary

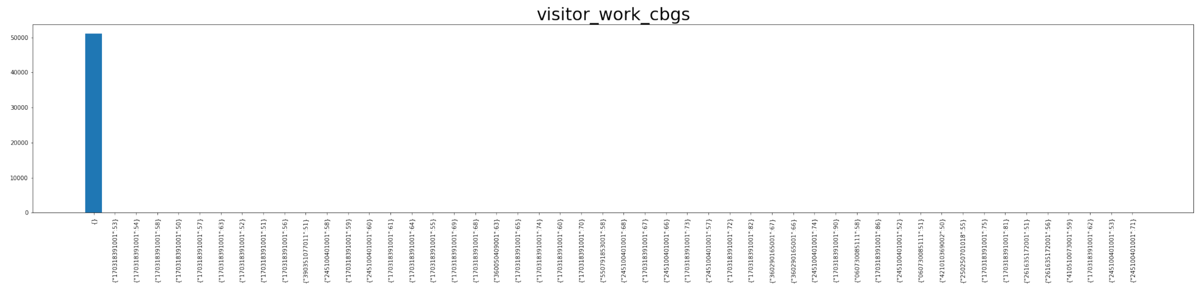
- Show top 5 frequency values
- Bar chart visualization of nominal data frequency (Top 50)

In [4]:
```
# bar chart for top 50 frequency of nominal_data
def bar_chart(nominal_data):
    x = nominal_data.index
    y = nominal_data.values

    print('Top 5 frequency of %s:'%nominal_data.name)
    top5_x = x[:5]
    top5_y = y[:5]
    top5_sum = np.sum(top5_y)
    s = ''
    for i in range(len(top5_x)):
        s += str(top5_x[i]) + ' ({:.2%})'.format(top5_y[i] / top5_sum) + ' | '
    print(s)

    plt.figure(figsize=(36,6))
    plt.title(nominal_data.name, fontsize=30)
    plt.bar(x[:50], y[:50])
    plt.xticks(rotation=90)
    plt.show()

# bar chart for each attribute
for i in nominal_index:
    bar_chart(data_frequency[i])
```

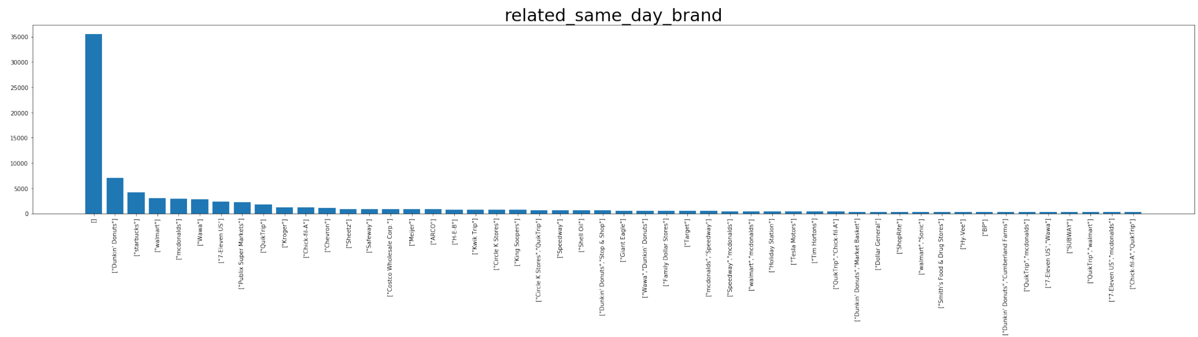
Top 5 frequency of visitor_home_cbgs:
{} (99.95%) | {"220870302082":59} (0.01%) | {"060750601001":57} (0.01%) | {"220710133021":95} (0.01%) | {"310550073134":56} (0.01%) |



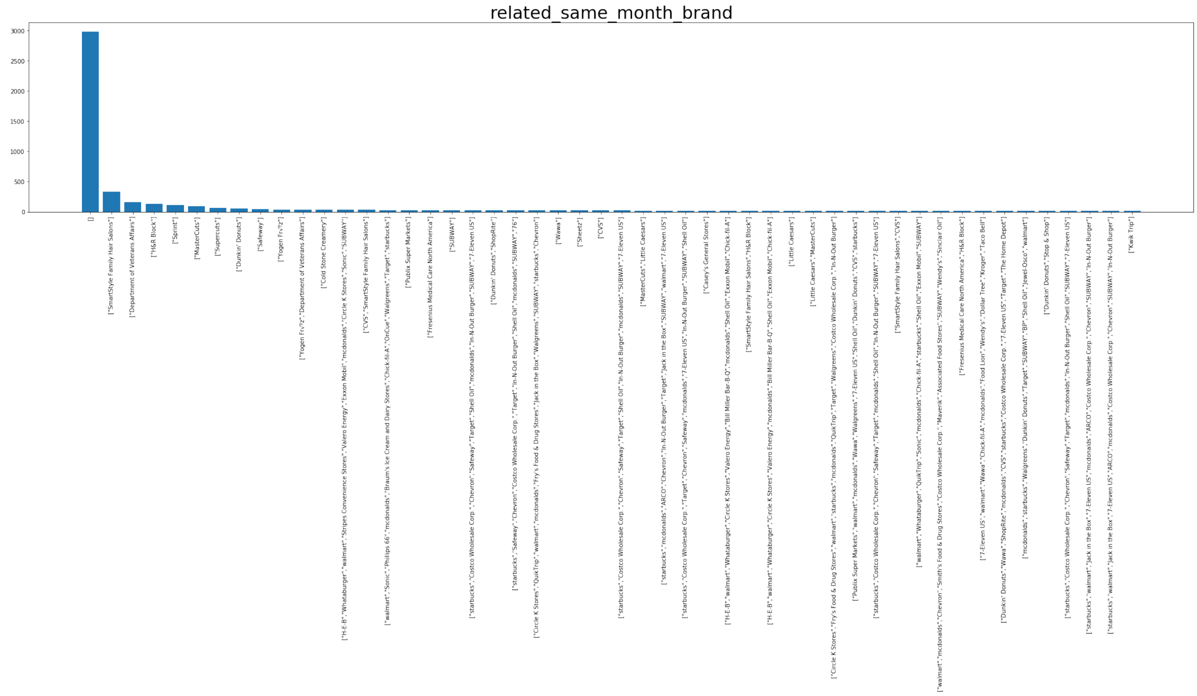
Top 5 frequency of visitor_work_cbgs:
{} (99.89%) | {"170318391001":53} (0.03%) | {"170318391001":54} (0.03%) | {"170318391001":58} (0.03%) | {"170318391001":50} (0.03%) |



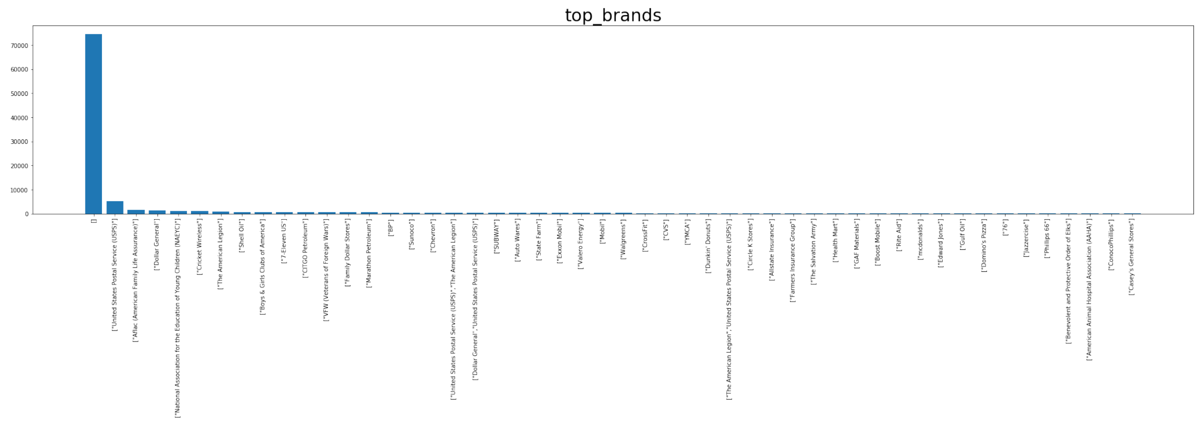
Top 5 frequency of related_same_day_brand:
[] (67.45%) | ["Dunkin' Donuts"] (13.30%) | ["starbucks"] (7.93%) | ["walmart"] (5.80%) | ["mcdonalds"] (5.52%) |



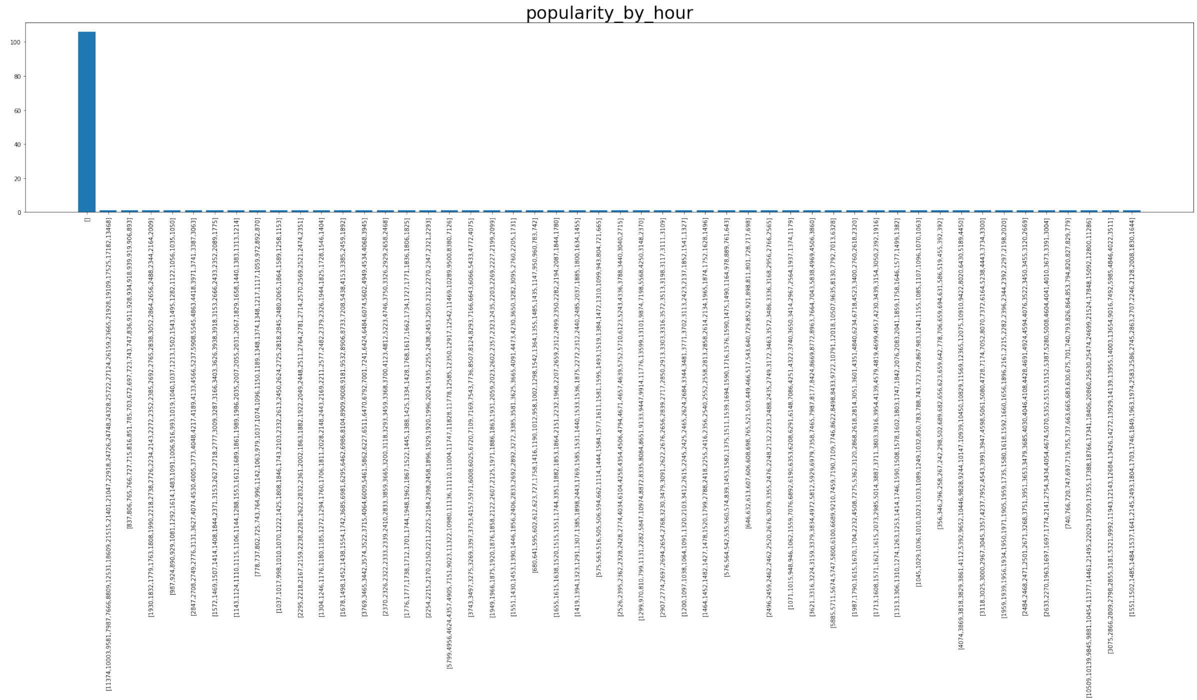
Top 5 frequency of related_same_month_brand:
[] (80.19%) | ["SmartStyle Family Hair Salons"] (9.01%) | ["Department of Veterans Affairs"] (4.19%) | ["H&R Block"] (3.60%) | ["Sprint"] (3.01%) |



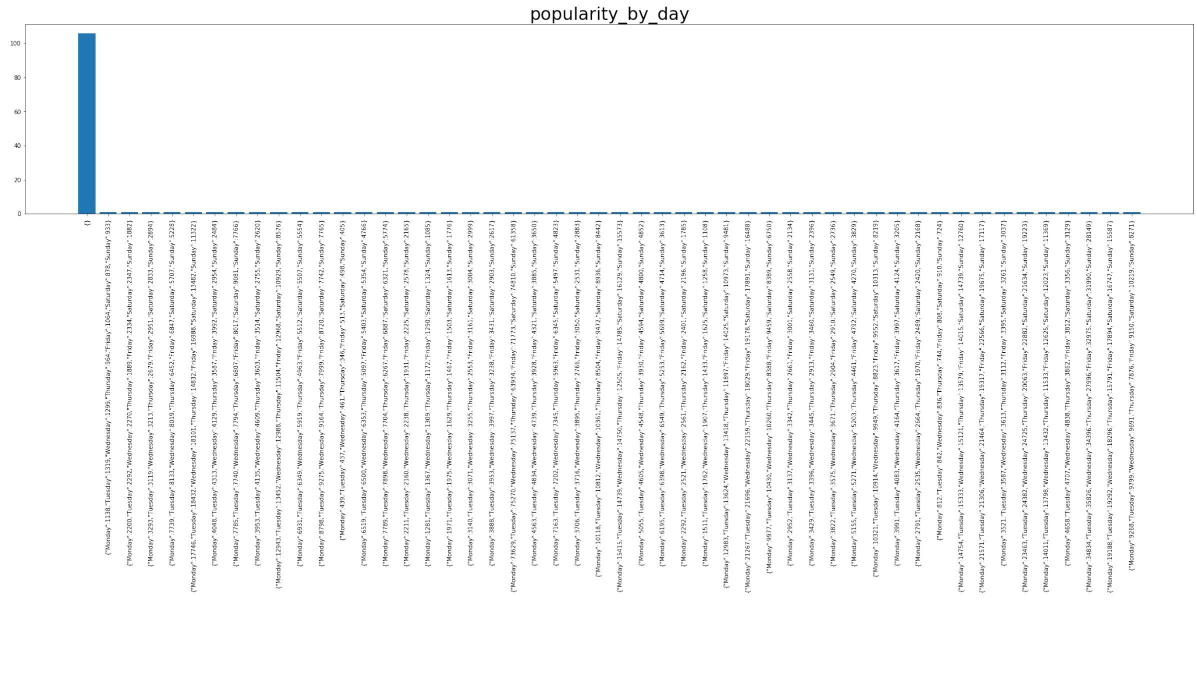
Top 5 frequency of top_brands:
[] (88.96%) | ["United States Postal Service (USPS)"] (6.39%) | ["Aflac (American Family Life Assurance)"] (1.81%) | ["Dollar General"] (1.56%) | ["National Association for the Education of Young Children (NAEYC)"] (1.28%) |



Top 5 frequency of popularity_by_hour:
[] (96.36%) | [11374,10003,9581,7987,7666,8809,12531,18609,21515,21401,21047,22918,24726,24748,24328,25722,27124,26159,23665,21928,19109,17525,17182,13468] (0.91%) | [837,806,765,766,727,715,816,851,785,703,672,697,723,743,747,836,911,928,934,918,939,919,906,893] (0.91%) | [1930,1832,1779,1763,1808,1990,2218,2738,2726,2234,2143,2272,2352,2385,2692,2765,2838,3052,2864,2656,2488,2344,2164,2009] (0.91%) | [987,924,890,929,1081,1292,1614,1483,1091,1006,916,993,1019,1040,1037,1213,1502,1543,1495,1282,1122,1056,1035,1050] (0.91%) |



Top 5 frequency of popularity_by_day:
{} (96.36%) | {"Monday":1138,"Tuesday":1319,"Wednesday":1299,"Thursday":964,"Friday":1064,"Saturday":878,"Sunday":933} (0.91%) | {"Monday":2200,"Tuesday":2292,"Wednesday":2270,"Thursday":1889,"Friday":2334,"Saturday":2347,"Sunday":1882} (0.91%) | {"Monday":3293,"Tuesday":3119,"Wednesday":3213,"Thursday":2679,"Friday":2951,"Saturday":2833,"Sunday":2894} (0.91%) | {"Monday":7739,"Tuesday":8133,"Wednesday":8019,"Thursday":6452,"Friday":6847,"Saturday":5707,"Sunday":5228} (0.91%) |



Numerical data summary

- Five-number summary:
    - Min
    - Q1(25%)
    - Q2(50%)
    - Q3(75%)
    - Max

In [5]:
```
def data_describe(data):
    """
    Generate descriptive statistics.

    Descriptive statistics include those that summarize the central
tendency, dispersion and shape of a dataset’s distribution, excludi
ng NaN values.
    """
    print('descriptive statistics (%s):' % data.name)
    info = data.describe()
    print('Min: ', info['min'], '\tQ1 (25%): ', info['25%'], '\tQ2
(50%): ', info['50%'], '\tQ3 (75%): ', info['75%'], '\tMax:', info[
'max'])
    print('Missing: %d'%(data_shape[0] - info['count']))
    print('-'*100)

# five-number summary for each attribute
for i in numerical_index:
    data_describe(data[i])
```

```
descriptive statistics (census_block_group):
Min:  10010201001.0     Q1 (25%):  131210116244.25      Q2 (50%):
290190012013.5  Q3 (75%):  420034897521.75      Max: 780309900000.
0
Missing: 1
-------------------------------------------------------------------
----------------------------------
descriptive statistics (date_range_start):
Min:  1538352000.0      Q1 (25%):  1538352000.0         Q2 (50%):
1538352000.0    Q3 (75%):  1538352000.0         Max: 1538352000.0
Missing: 0
-------------------------------------------------------------------
----------------------------------
descriptive statistics (date_range_end):
Min:  1541030400.0      Q1 (25%):  1541030400.0         Q2 (50%):
1541030400.0    Q3 (75%):  1541030400.0         Max: 1541030400.0
Missing: 0
-------------------------------------------------------------------
----------------------------------
descriptive statistics (raw_visit_count):
Min:  60.0      Q1 (25%):  17042.0      Q2 (50%):  30640.0      Q3
(75%):  56678.0         Max: 7179900.0
Missing: 106
-------------------------------------------------------------------
----------------------------------
descriptive statistics (raw_visitor_count):
Min:  50.0      Q1 (25%):  3430.0       Q2 (50%):  6541.0       Q3
(75%):  13099.0         Max: 6113949.0
Missing: 106
-------------------------------------------------------------------
----------------------------------
descriptive statistics (distance_from_home):
Min:  706.0     Q1 (25%):  8584.0       Q2 (50%):  14614.0      Q3
(75%):  31397.75        Max: 6297845.0
Missing: 217
-------------------------------------------------------------------
----------------------------------
```

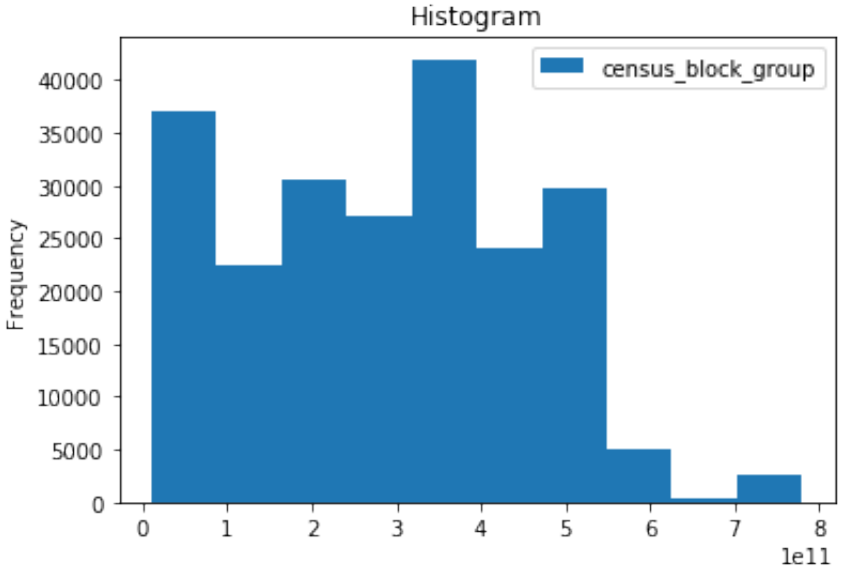
1.2 Data Visualization

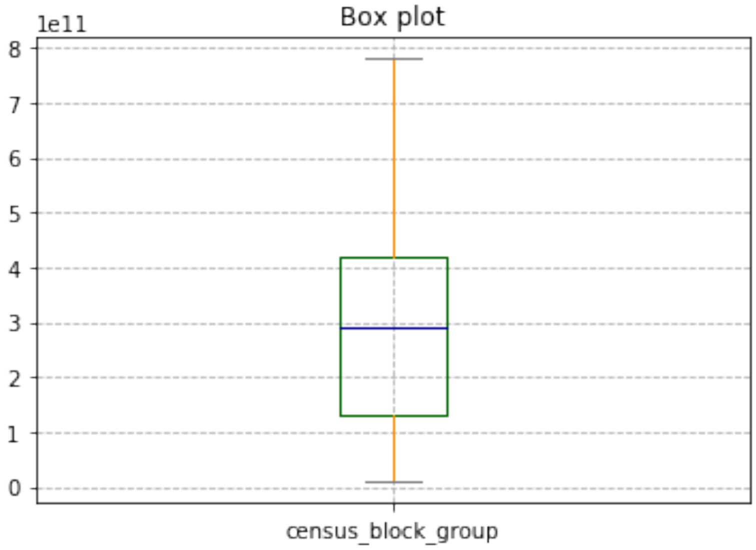
Numerical data visualization

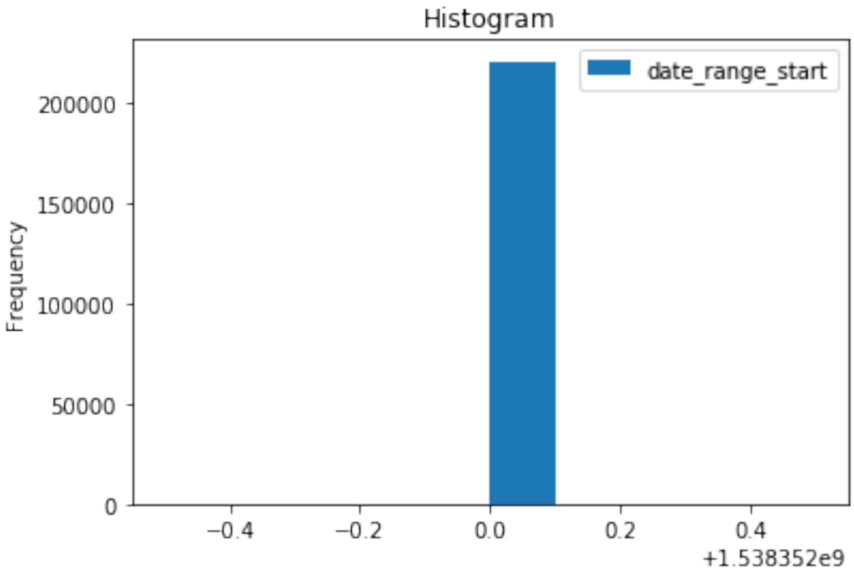
- histogram
- box plot
- scatter matrix

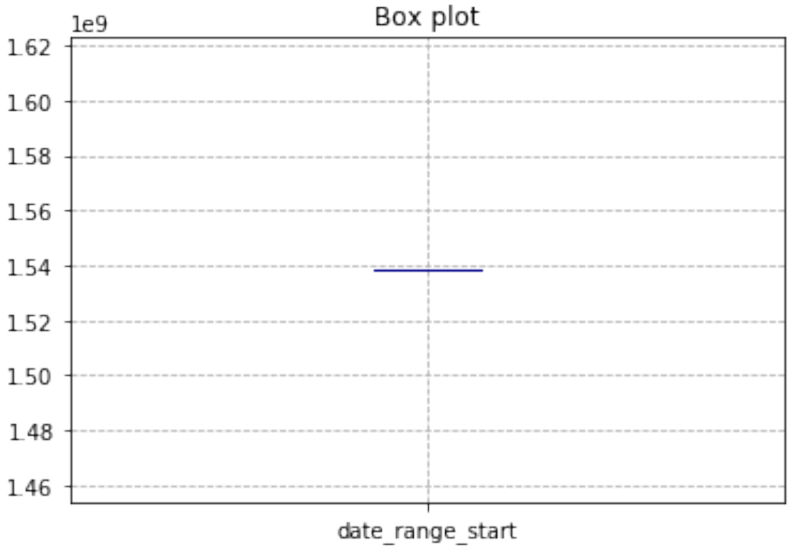
In [6]:
```
# histogram
def histogram(data):
    df = pd.DataFrame(data)
```

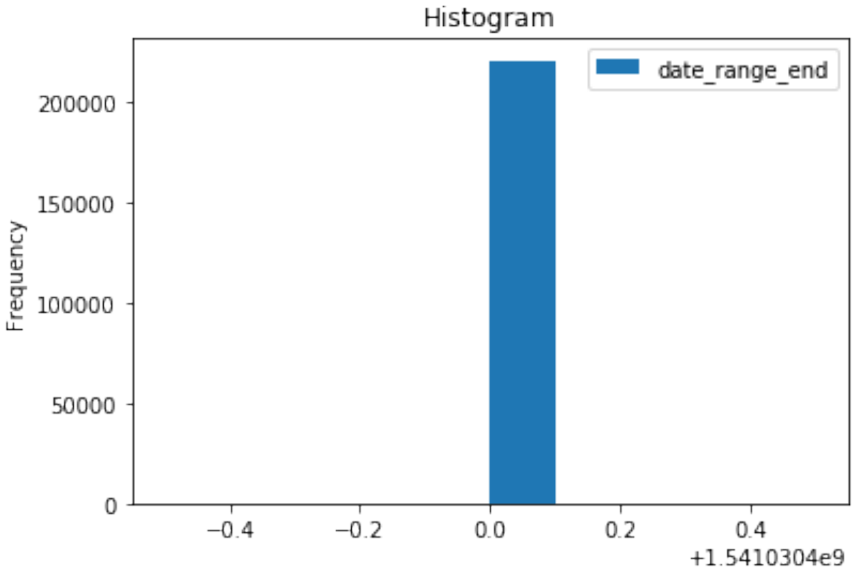
```
    df.plot.hist()
    plt.title('Histogram')
    plt.show()

# box plot
def box_plot(data):
    # boxes 箱线
    # whiskers 分为数于error bar横线之间的竖线的颜色
    # medians 中位线的颜色
    # caps error bar 横线的颜色
    color = dict(boxes  = 'DarkGreen',whiskers = 'DarkOrange',media
ns = 'DarkBlue',caps ='Gray')
    data.plot.box(grid=True, color=color)   # color 样式填充
    plt.grid(linestyle='--')
    plt.title('Box plot')
    plt.show()

def box_plot1(data):
    df = pd.DataFrame(data)
    df.boxplot(sym = 'o',    #异常点形状
               vert = True,          # 是否垂直
               whis=1.5,             # IQR
               patch_artist = True,  # 上下四分位框是否填充
               meanline = False,showmeans = True,  # 是否有均值线
及其形状
               showbox = True,   # 是否显示箱线
               showfliers = True,  #是否显示异常值
               notch = False,    # 中间箱体是否缺口
               return_type='dict')  # 返回类型为字典
    plt.grid(linestyle='--')
    plt.title('Box plot')
    plt.show()

# scatter plot
def scatter_plot(data1, index1, index2):
    data.plot.scatter(x=index1,
               y=index2,
               c='DarkBlue')
    plt.title('Scatter plot')
    plt.show()

for i in numerical_index:
    histogram(data[i])
    box_plot(data[i])

# scatter matrix
# The diagonal of the matrix is the KDE(Kernel Density Estimation)
of ['raw_visit_count', 'raw_visitor_count', 'distance_from_home'] f
eature
pd.plotting.scatter_matrix(data[['raw_visit_count', 'raw_visitor_co
unt', 'distance_from_home']], marker='o', figsize=(20, 20), diagona
l='kde')
plt.show()
```

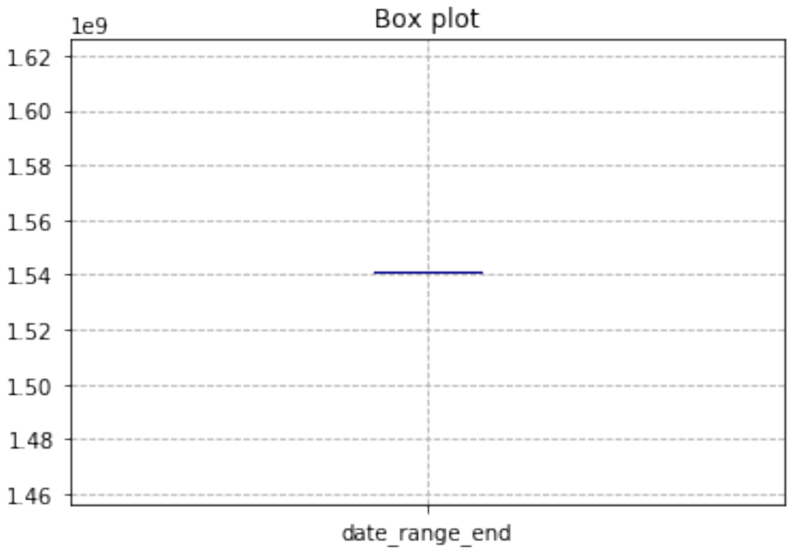

Histogram
census_block_group
Frequency
40000
35000
30000
25000
20000
15000
10000
5000
0
0 1 2 3 4 5 6 7 8
1e11

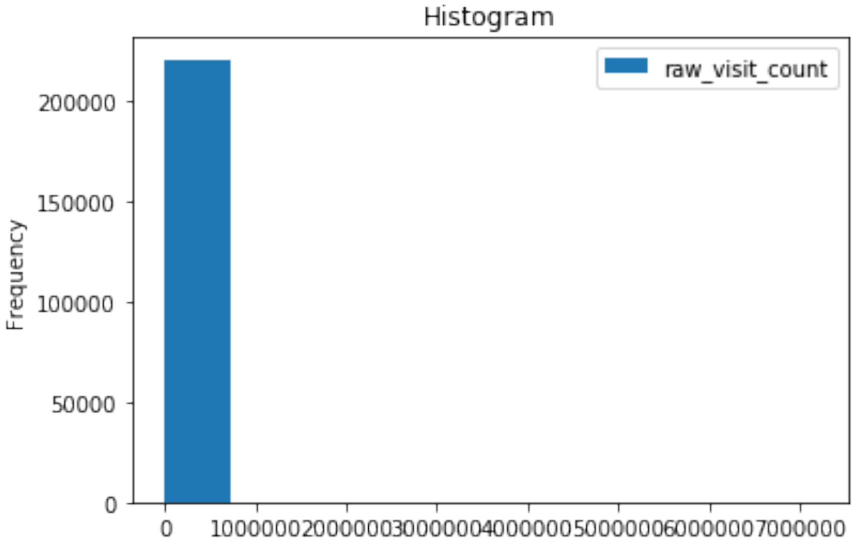

Box plot
1e11
8
7
6
5
4
3
2
1
0
census_block_group

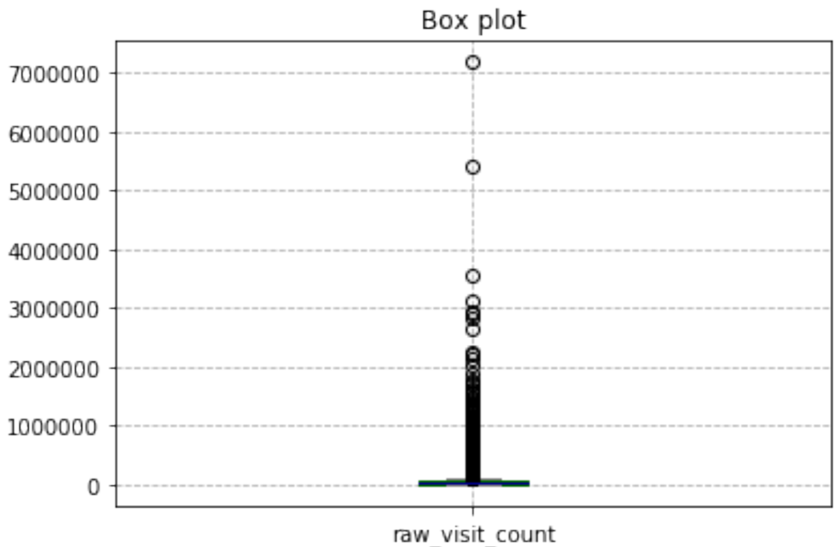

Histogram
date_range_start
Frequency
200000
150000
100000
50000
0
−0.4 −0.2 0.0 0.2 0.4
+1.538352e9

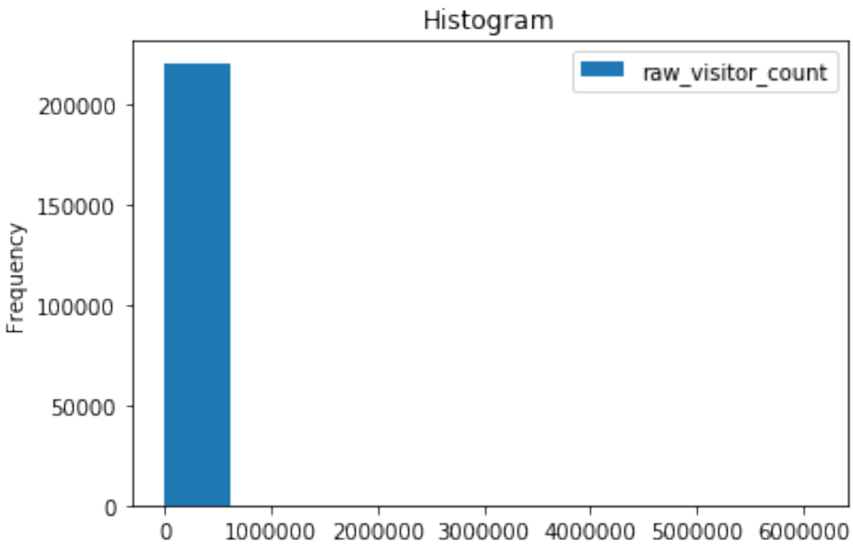

Box plot
1e9
1.62
1.60
1.58
1.56
1.54
1.52
1.50
1.48
1.46
date_range_start

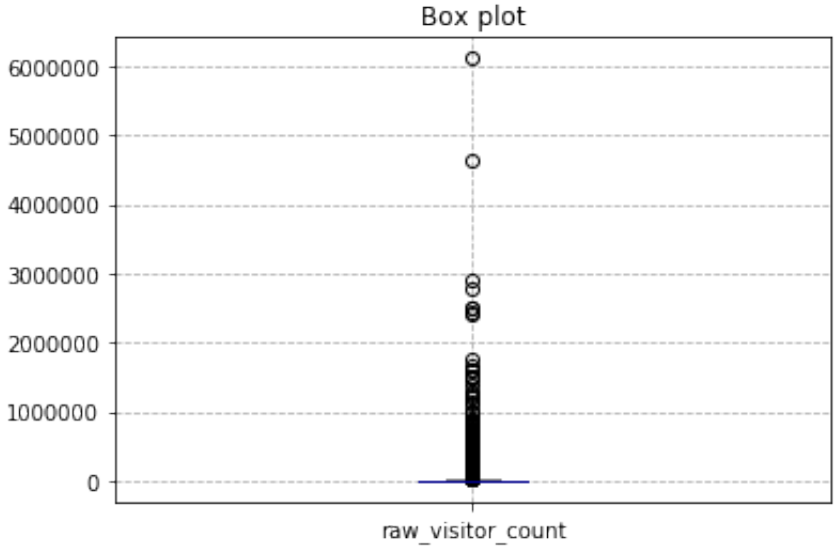

Histogram
date_range_end
Frequency
200000
150000
100000
50000
0
−0.4
−0.2
0.0
0.2
0.4
+1.5410304e9

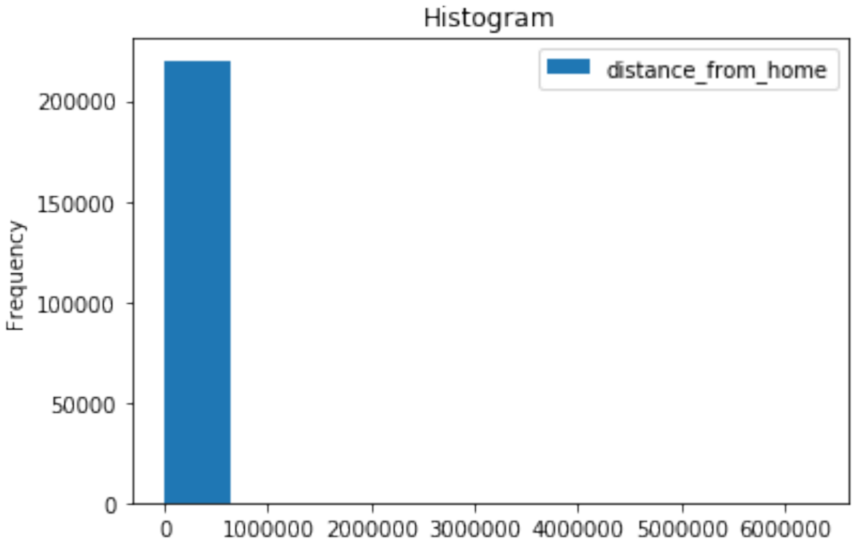

Box plot
1e9
1.62
1.60
1.58
1.56
1.54
1.52
1.50
1.48
1.46
date_range_end

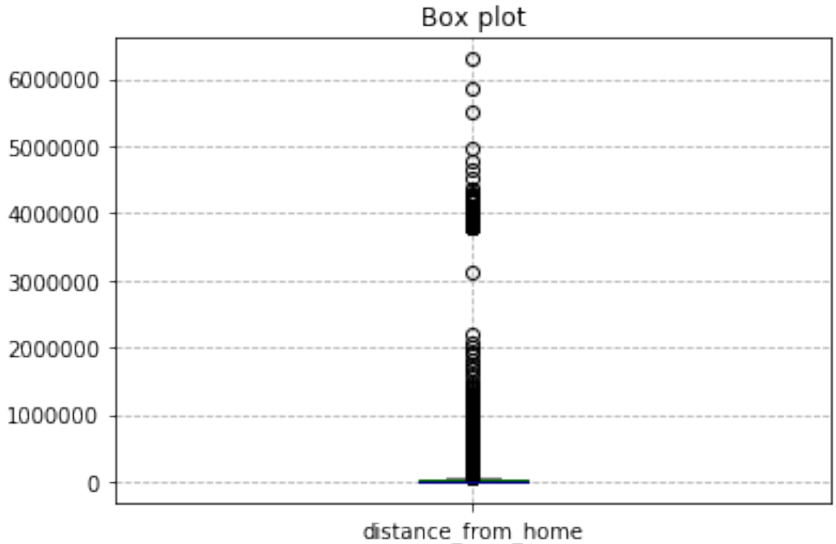

Histogram
raw_visit_count
Frequency
0
50000
100000
150000
200000
1000000
2000000
3000000
4000000
5000000
6000000
7000000

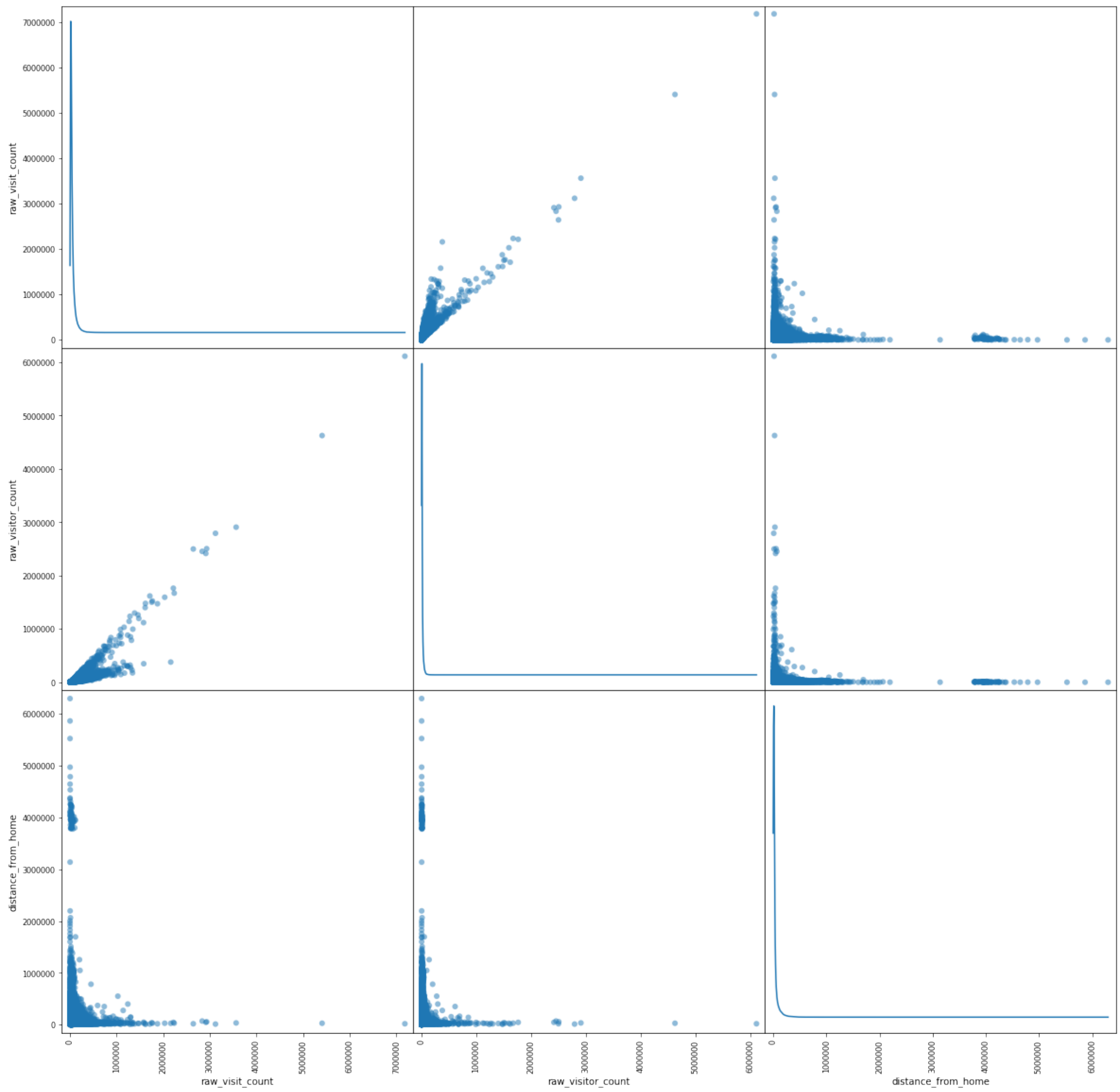

Box plot
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
raw_visit_count


Histogram
raw_visitor_count
Frequency
0
50000
100000
150000
200000
1000000
2000000
3000000
4000000
5000000
6000000


Box plot
6000000
5000000
4000000
3000000
2000000
1000000
0
raw_visitor_count


Histogram
distance_from_home
Frequency
200000
150000
100000
50000
0
0
1000000
2000000
3000000
4000000
5000000
6000000


Box plot
6000000
5000000
4000000
3000000
2000000
1000000
0
distance_from_home



2. Processing of missing data

2.1 Remove missing values

In [7]:
```
# Remove missing values
data_remove = data.dropna()
data_remove.info()

# Get frequency of each arrtibute (after remove missing values)
data_remove_frequency = {key: data_remove[key].value_counts() for k
ey in data_remove.columns}
```

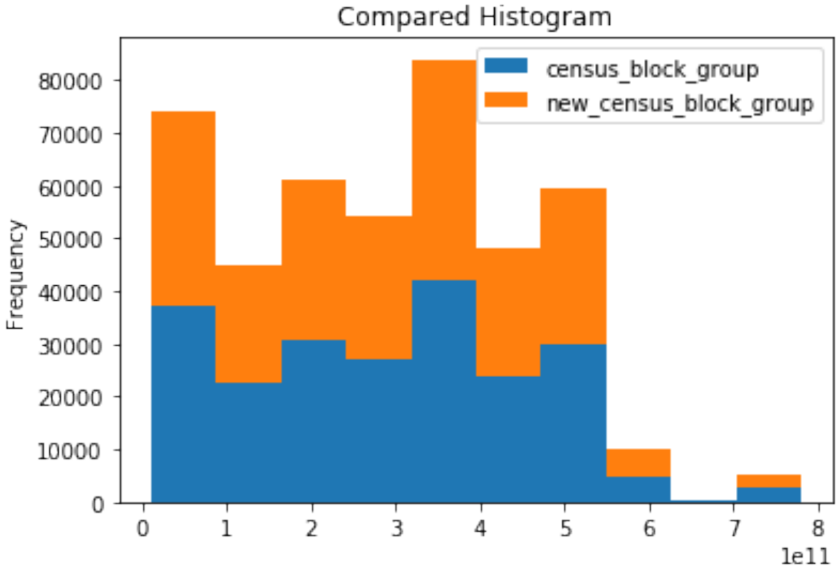
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 220518 entries, 0 to 220628
Data columns (total 13 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   census_block_group        220518 non-null  float64
 1   date_range_start          220518 non-null  int64  
 2   date_range_end            220518 non-null  int64  
 3   raw_visit_count           220518 non-null  float64
 4   raw_visitor_count         220518 non-null  float64
 5   visitor_home_cbgs         220518 non-null  object 
 6   visitor_work_cbgs         220518 non-null  object 
 7   distance_from_home        220518 non-null  float64
 8   related_same_day_brand    220518 non-null  object 
 9   related_same_month_brand  220518 non-null  object 
 10  top_brands                220518 non-null  object 
 11  popularity_by_hour        220518 non-null  object 
 12  popularity_by_day         220518 non-null  object 
dtypes: float64(4), int64(2), object(7)
memory usage: 23.6+ MB
```

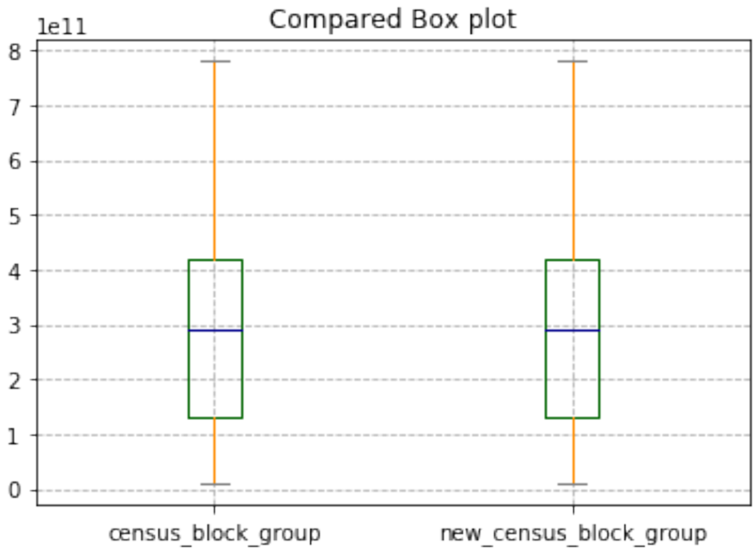
compare with raw data

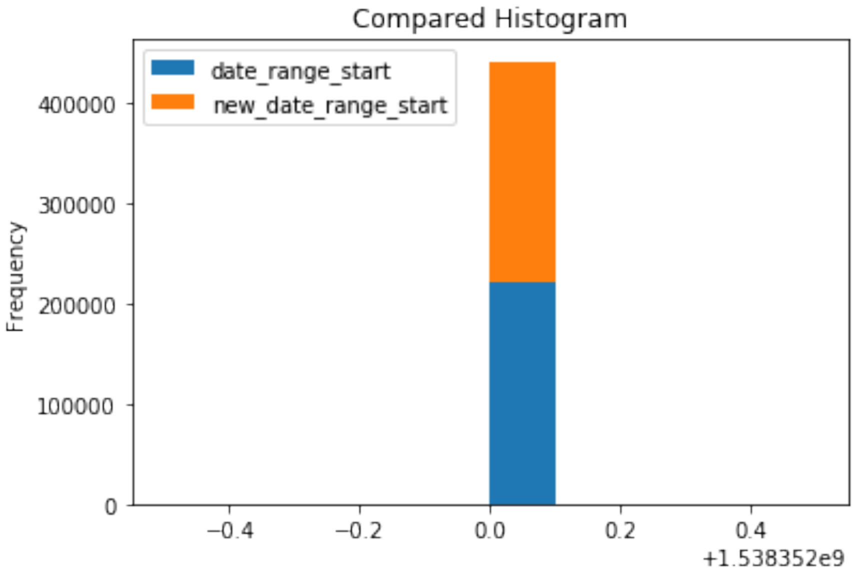
There is no missing data of nominal data, so we need no visualization numerical data compared visualization

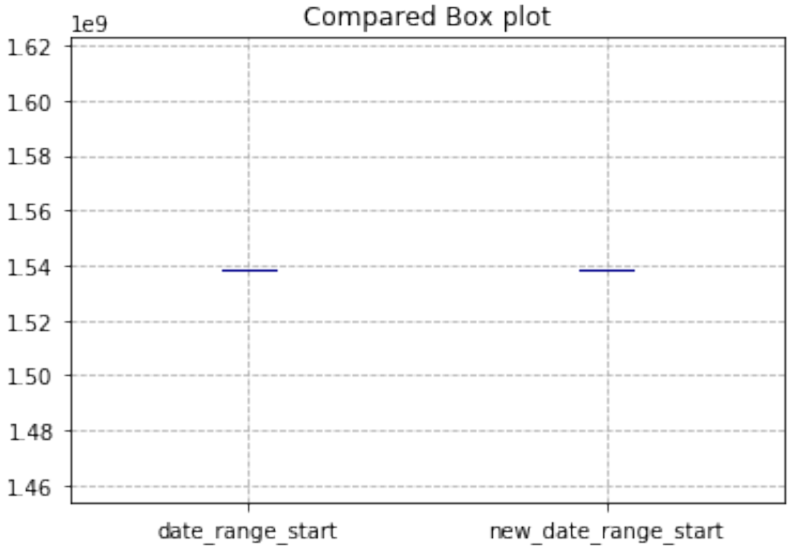
- Compared histogram
- Compared box plot
- Compared scatter plot

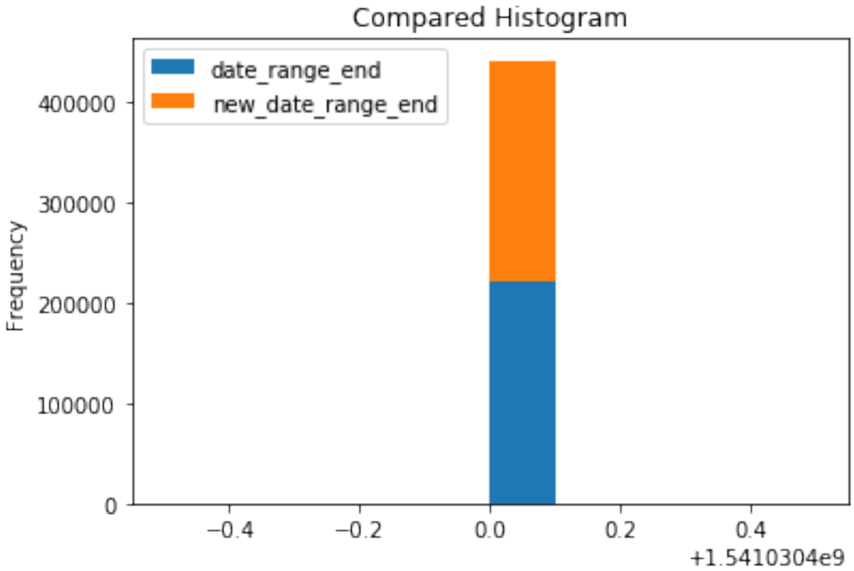
In [9]:
```
# histogram
def histogram_compare(raw_data, new_data):
    new_name = ['new_' + new_data.name]
    raw_data = pd.DataFrame(raw_data)
    new_data = pd.DataFrame(new_data)
    new_data.columns=new_name
    df = raw_data.append(new_data)
    df.plot.hist(stacked=True)
    plt.title('Compared Histogram')
    plt.show()

# box plot
def box_plot_compare(raw_data, new_data):
    # boxes 箱线
    # whiskers 分为数于error bar横线之间的竖线的颜色
    # medians 中位线的颜色
    # caps error bar 横线的颜色
    new_name = ['new_' + new_data.name]
    raw_data = pd.DataFrame(raw_data)
    new_data = pd.DataFrame(new_data)
    new_data.columns=new_name
    df = raw_data.append(new_data)
    color = dict(boxes  = 'DarkGreen',whiskers = 'DarkOrange',media
ns = 'DarkBlue',caps ='Gray')
    df.plot.box(grid=True, color=color)   # color 样式填充
    plt.grid(linestyle='--')
    plt.title('Compared Box plot')
    plt.show()

# scatter plot
def scatter_plot_compare(data1, data2, index1, index2):
    ax = data.plot.scatter(x=index1, y=index2, c='DarkBlue', label=
'Raw data')
    data_remove.plot.scatter(x=index1, y=index2, c='r', label='New
data', ax=ax)
    plt.title('Compared Scatter plot')
    plt.show()

for i in numerical_index:
    histogram_compare(data[i], data_remove[i])
    box_plot_compare(data[i], data_remove[i])

scatter_plot_compare(data, data_remove, 'raw_visit_count', 'raw_vis
itor_count')
scatter_plot_compare(data, data_remove, 'raw_visit_count', 'distanc
e_from_home')
```

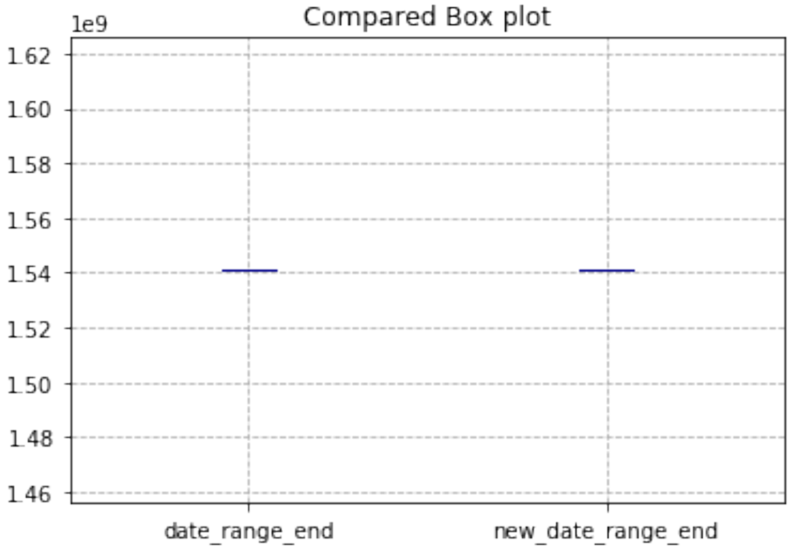

Compared Histogram
census_block_group
new_census_block_group
80000
70000
60000
50000
40000
30000
20000
10000
0
Frequency
0 1 2 3 4 5 6 7 8
1e11

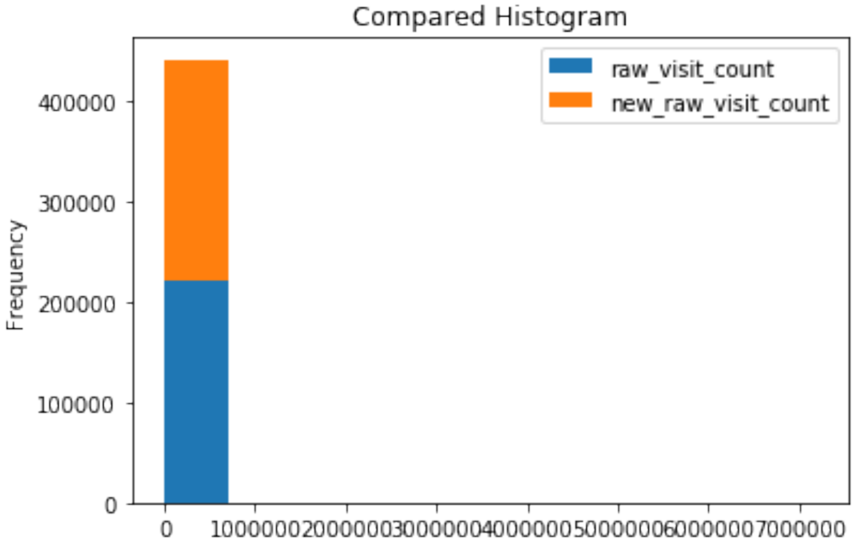

Compared Box plot
1e11
8
7
6
5
4
3
2
1
0
census_block_group
new_census_block_group

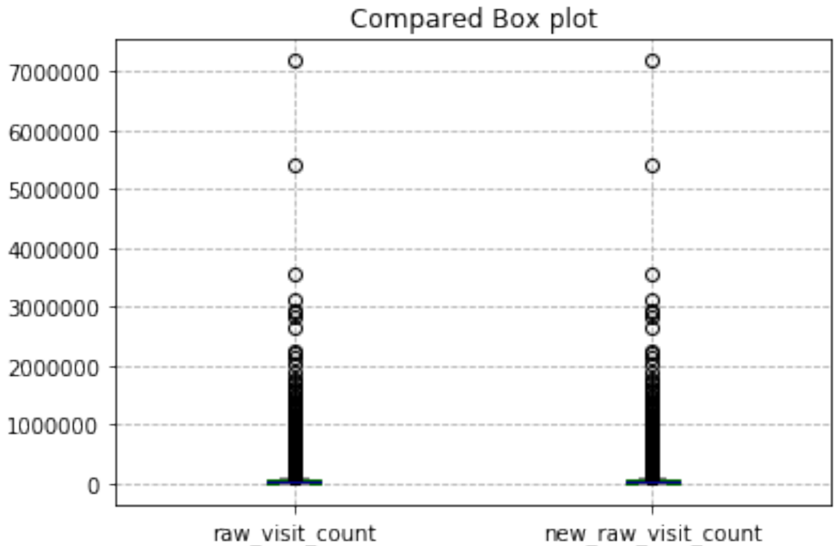

Compared Histogram
date_range_start
new_date_range_start
400000
300000
200000
100000
0
Frequency
−0.4 −0.2 0.0 0.2 0.4
+1.538352e9

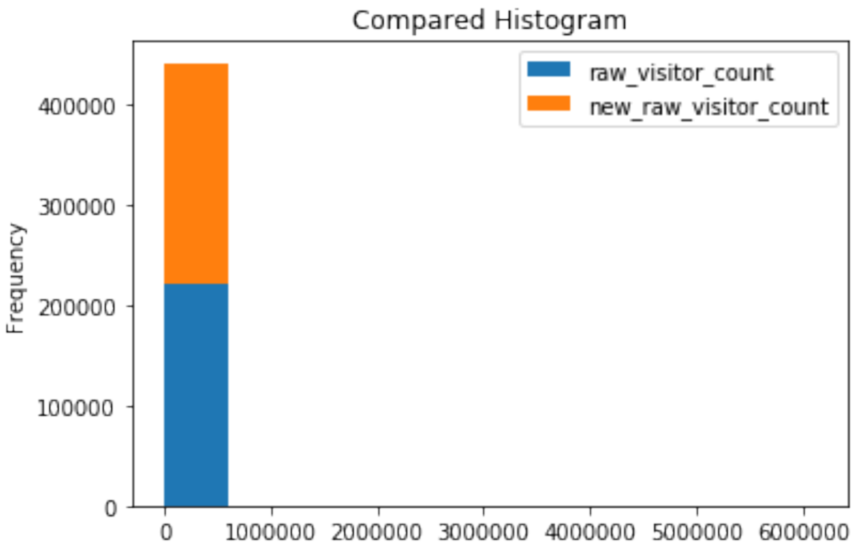

Compared Box plot
1e9
1.62
1.60
1.58
1.56
1.54
1.52
1.50
1.48
1.46
date_range_start
new_date_range_start

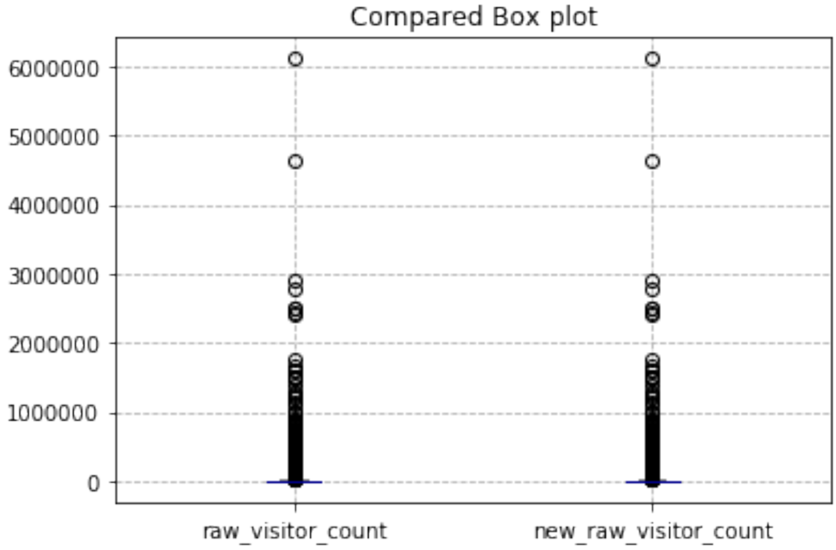

Compared Histogram
date_range_end
new_date_range_end
Frequency
400000
300000
200000
100000
0
−0.4
−0.2
0.0
0.2
0.4
+1.5410304e9

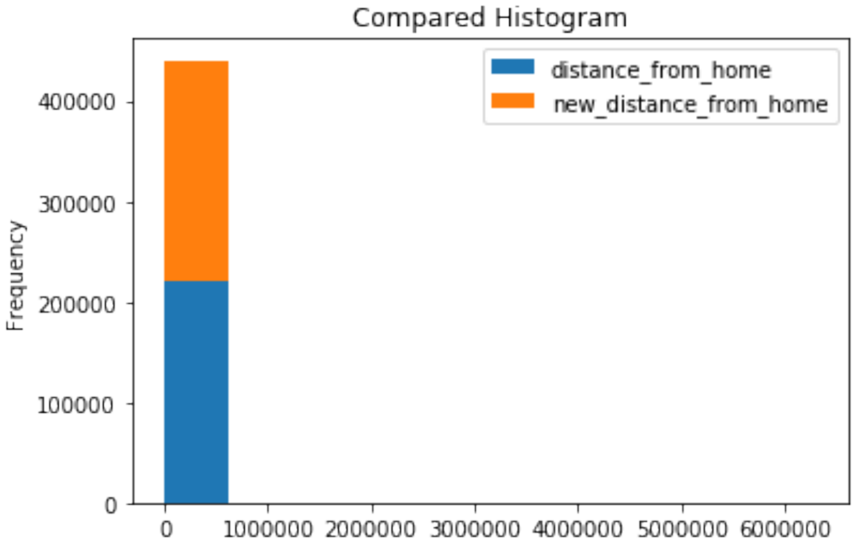

Compared Box plot
1e9
1.62
1.60
1.58
1.56
1.54
1.52
1.50
1.48
1.46
date_range_end
new_date_range_end

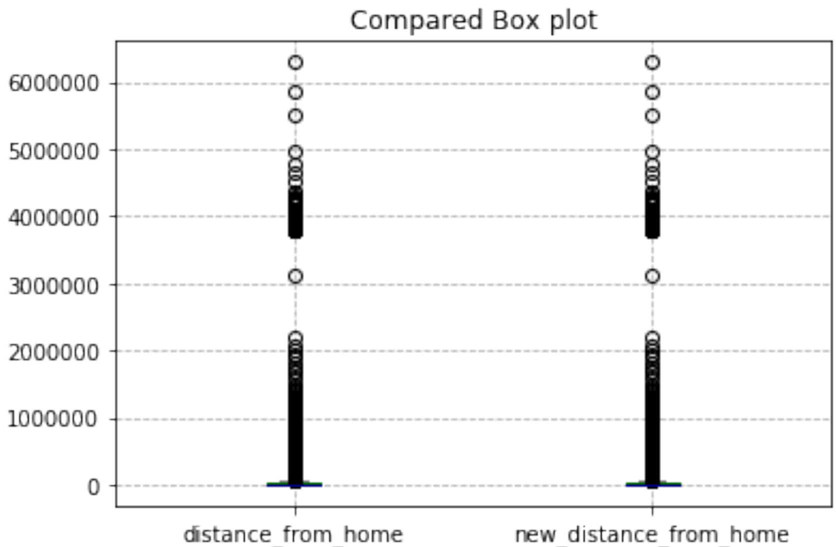

Compared Histogram
raw_visit_count
new_raw_visit_count
Frequency
0
100000
200000
300000
400000
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000

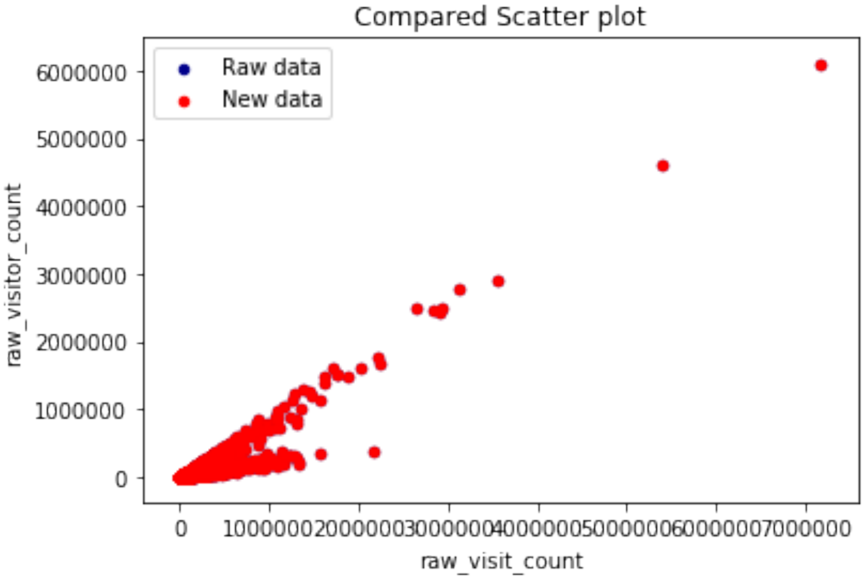

Compared Box plot
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
raw_visit_count
new_raw_visit_count

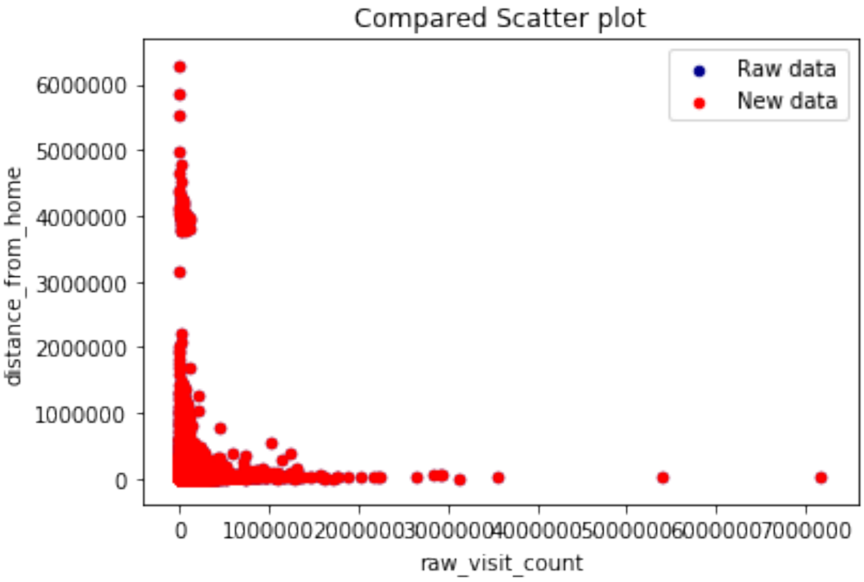

Compared Histogram
raw_visitor_count
new_raw_visitor_count
Frequency
0
100000
200000
300000
400000
0
1000000
2000000
3000000
4000000
5000000
6000000


Compared Box plot
6000000
5000000
4000000
3000000
2000000
1000000
0
raw_visitor_count
new_raw_visitor_count


Compared Histogram
distance_from_home
new_distance_from_home
Frequency
400000
300000
200000
100000
0
0
1000000
2000000
3000000
4000000
5000000
6000000


Compared Box plot
6000000
5000000
4000000
3000000
2000000
1000000
0
distance_from_home
new_distance_from_home





2.2 Fill NA/NaN values using the highest frequency value

In [10]:
```
# Fill NA/NaN values using the highest frequency value.
highest_frequency_values = {key: data[key].value_counts().index[0]
for key in data.columns}
data_highest = data.fillna(value=highest_frequency_values)
data_highest.info()

# Get frequency of each arrtibute (after remove missing values)
data_highest_frequency = {key: data_highest[key].value_counts() for
key in data_highest.columns}
```

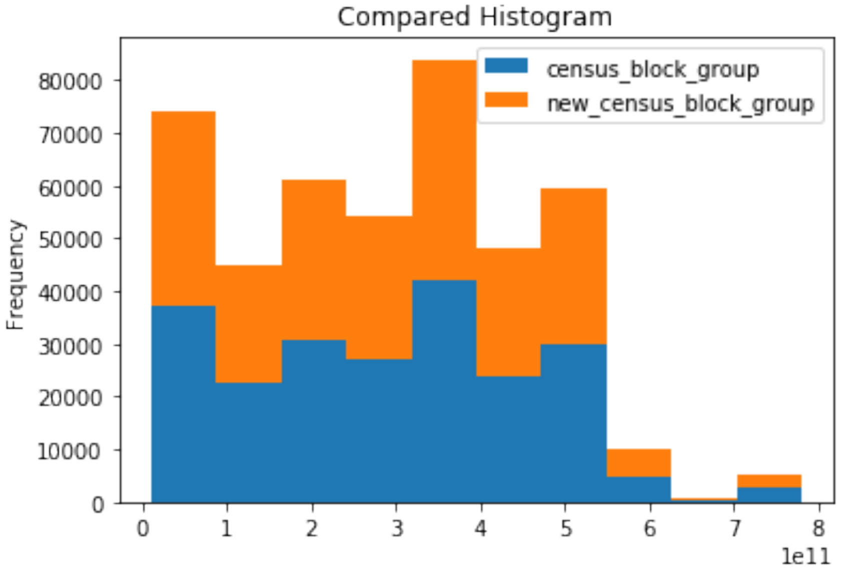
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220735 entries, 0 to 220734
Data columns (total 13 columns):
 #   Column                     Non-Null Count   Dtype  
---  ------                     --------------   -----  
 0   census_block_group         220735 non-null  float64
 1   date_range_start           220735 non-null  int64  
 2   date_range_end             220735 non-null  int64  
 3   raw_visit_count            220735 non-null  float64
 4   raw_visitor_count          220735 non-null  float64
 5   visitor_home_cbgs          220735 non-null  object 
 6   visitor_work_cbgs          220735 non-null  object 
 7   distance_from_home         220735 non-null  float64
 8   related_same_day_brand     220735 non-null  object 
 9   related_same_month_brand   220735 non-null  object 
 10  top_brands                 220735 non-null  object 
 11  popularity_by_hour         220735 non-null  object 
 12  popularity_by_day          220735 non-null  object 
dtypes: float64(4), int64(2), object(7)
memory usage: 21.9+ MB
```

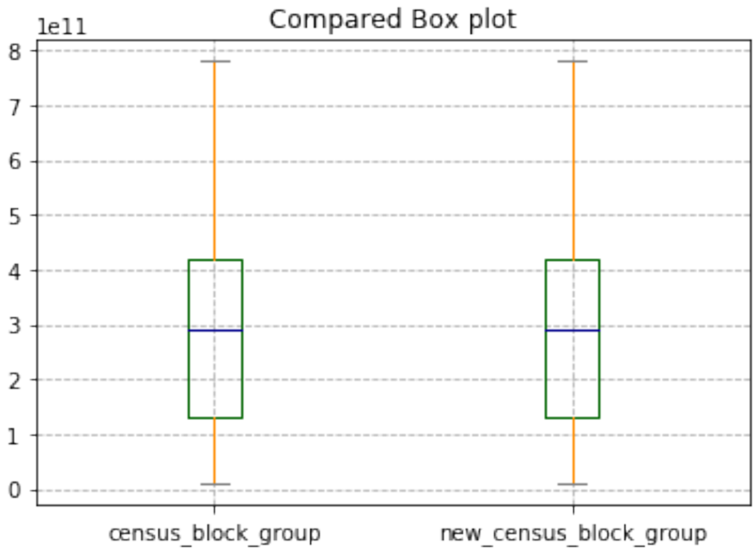
compare with raw data

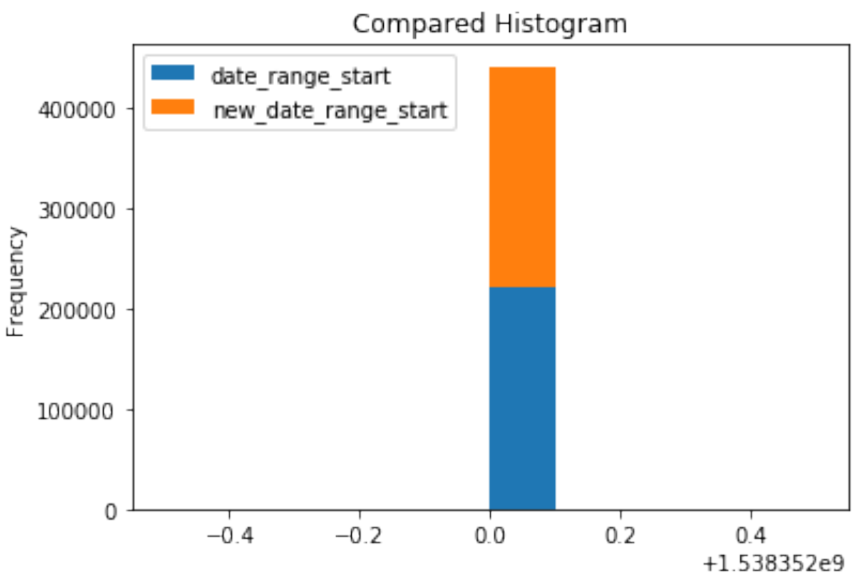
There is no missing data of nominal data, so we need no visualization numerical data compared visualization

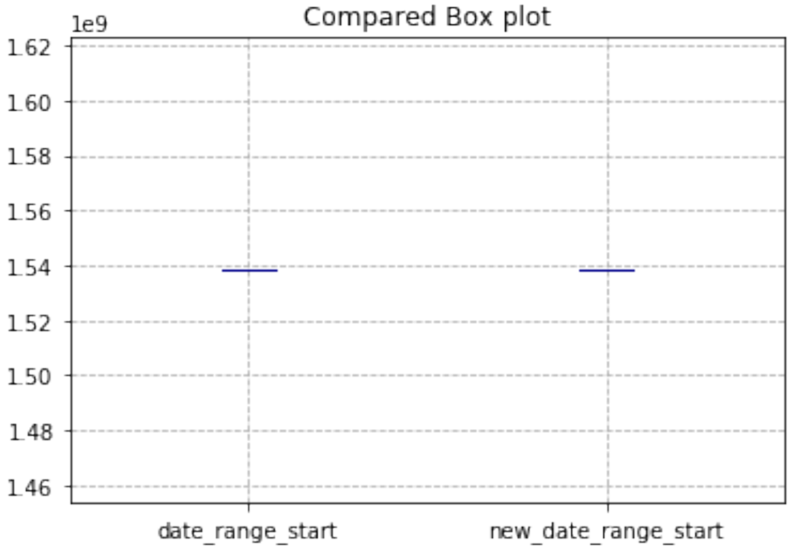
- Compared histogram
- Compared box plot
- Compared scatter plot

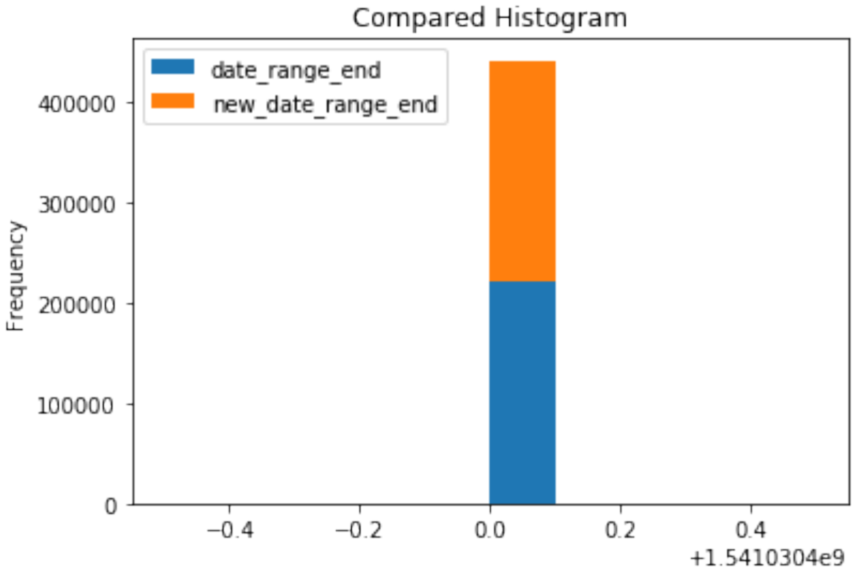
In [12]:
```
for i in numerical_index:
    histogram_compare(data[i], data_highest[i])
    box_plot_compare(data[i], data_highest[i])

scatter_plot_compare(data, data_highest, 'raw_visit_count', 'raw_vi
sitor_count')
scatter_plot_compare(data, data_remove, 'raw_visit_count', 'distanc
e_from_home')
```

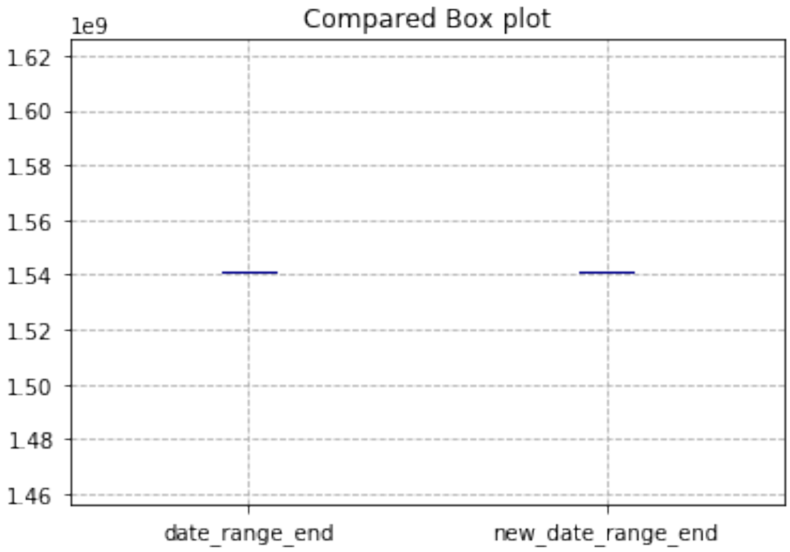

Compared Histogram
census_block_group
new_census_block_group
Frequency
80000
70000
60000
50000
40000
30000
20000
10000
0
0
1
2
3
4
5
6
7
8
1e11

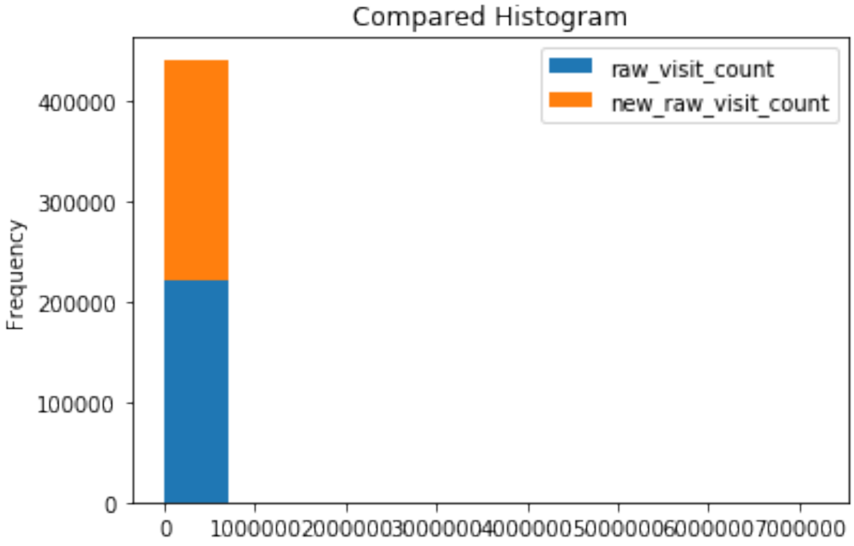

Compared Box plot
1e11
8
7
6
5
4
3
2
1
0
census_block_group
new_census_block_group

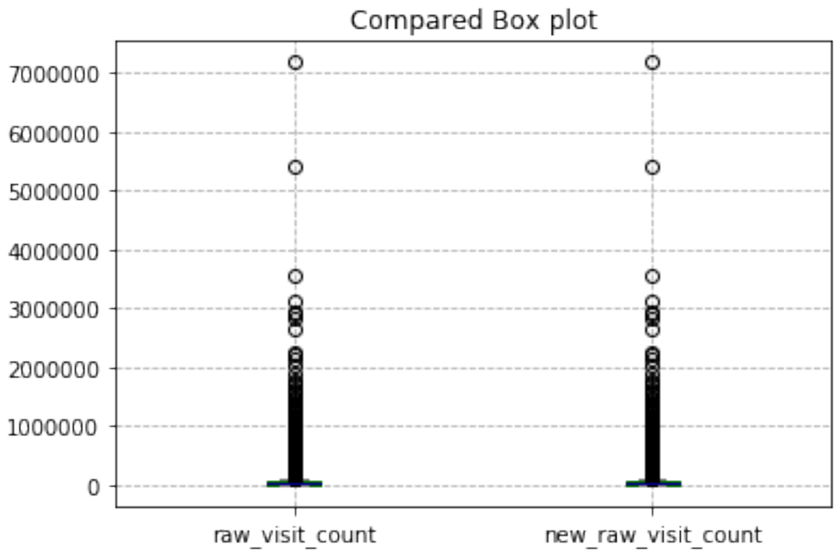

Compared Histogram
date_range_start
new_date_range_start
Frequency
400000
300000
200000
100000
0
−0.4
−0.2
0.0
0.2
0.4
+1.538352e9

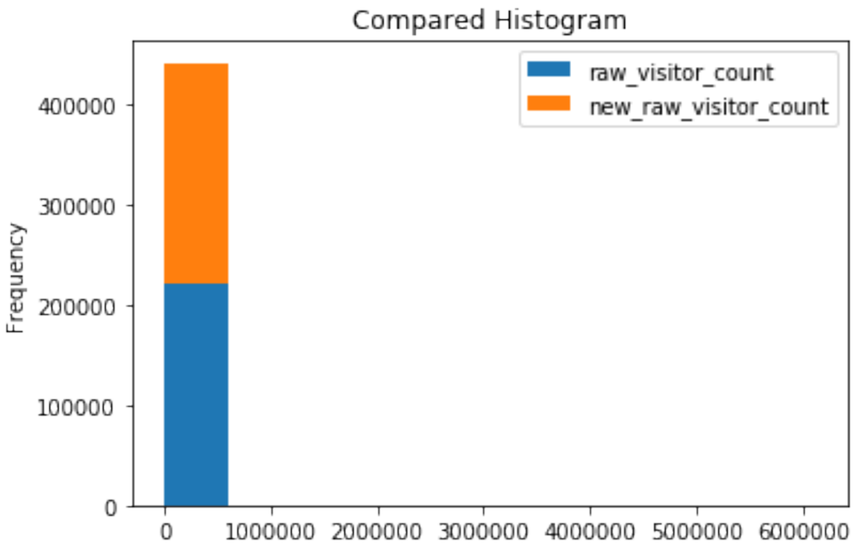

Compared Box plot
1e9
1.62
1.60
1.58
1.56
1.54
1.52
1.50
1.48
1.46
date_range_start
new_date_range_start

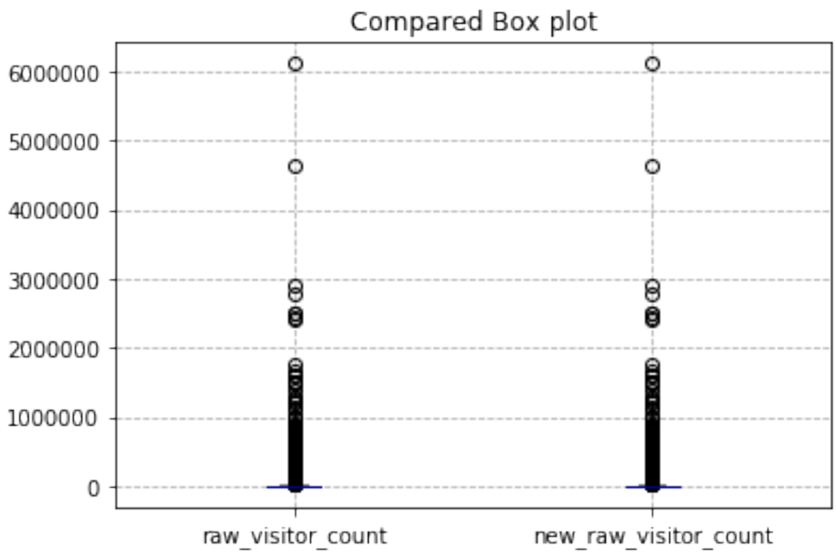

Compared Histogram
date_range_end
new_date_range_end
Frequency
400000
300000
200000
100000
0
−0.4
−0.2
0.0
0.2
0.4
+1.5410304e9

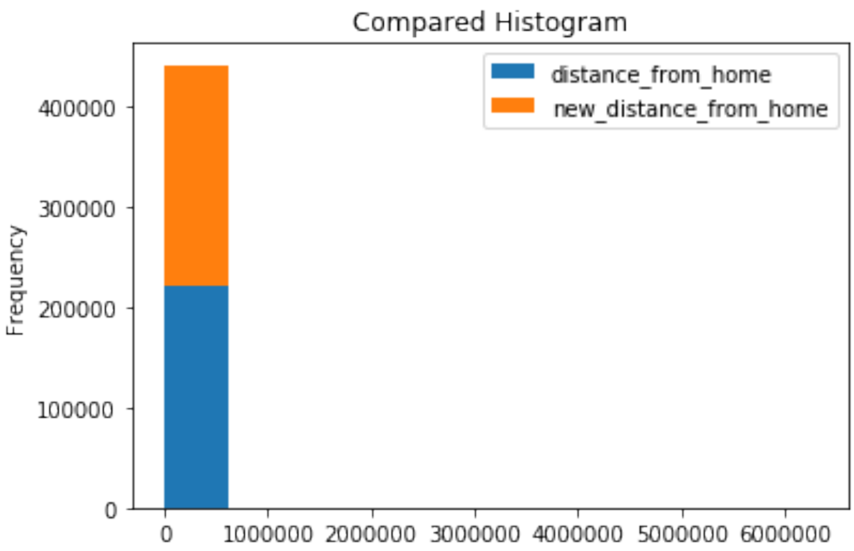

Compared Box plot
1e9
1.62
1.60
1.58
1.56
1.54
1.52
1.50
1.48
1.46
date_range_end
new_date_range_end

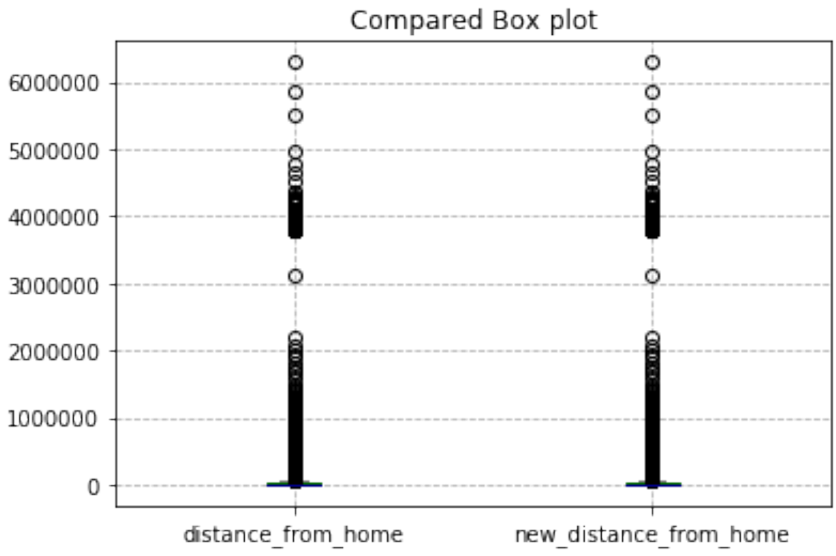

Compared Histogram
raw_visit_count
new_raw_visit_count
Frequency
400000
300000
200000
100000
0
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000

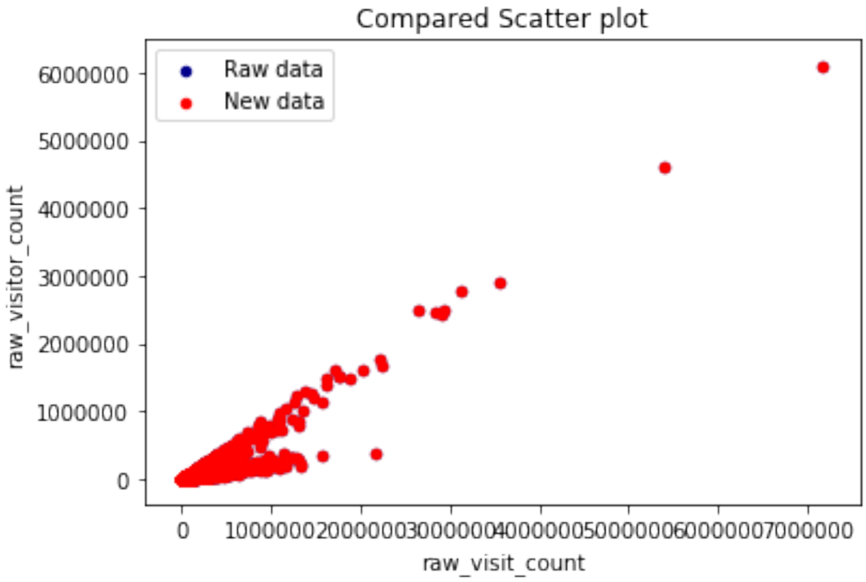

Compared Box plot
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
raw_visit_count
new_raw_visit_count

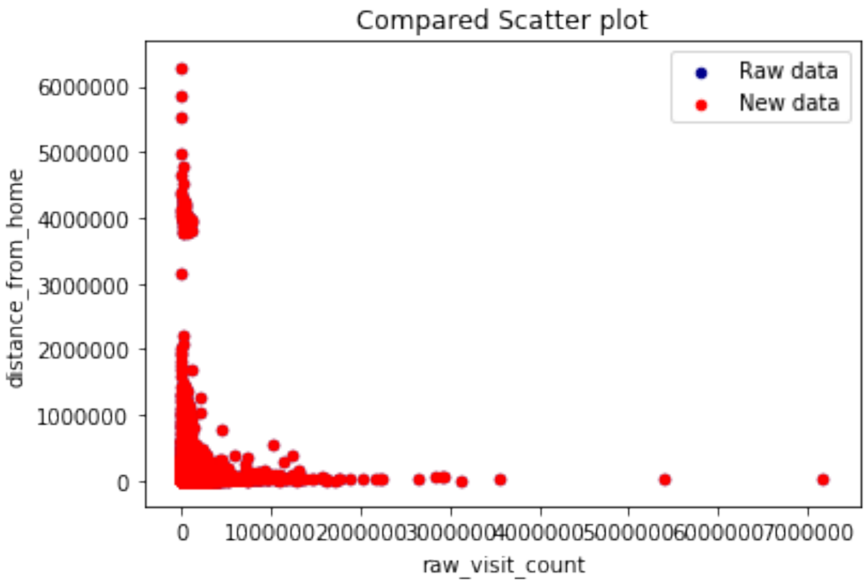

Compared Histogram
raw_visitor_count
new_raw_visitor_count
Frequency
400000
300000
200000
100000
0
0
1000000
2000000
3000000
4000000
5000000
6000000


Compared Box plot
6000000
5000000
4000000
3000000
2000000
1000000
0
raw_visitor_count
new_raw_visitor_count


Compared Histogram
distance_from_home
new_distance_from_home
Frequency
400000
300000
200000
100000
0
0
1000000
2000000
3000000
4000000
5000000
6000000


Compared Box plot
6000000
5000000
4000000
3000000
2000000
1000000
0
distance_from_home
new_distance_from_home





2.3 Fill in missing values by the correlation of the attribute

Random Forest Regressor

Missing attributes: census_block_group (1) | raw_visit_count (106) | raw_visitor_count (106) | distance_from_home (217)

In [13]:
```
from sklearn.ensemble import RandomForestRegressor
def set_missing_values(df, complete_index):

    missing_index = complete_index[0]

    # Take out the existing numerical data (no NaN) and throw them
in Random Forest Regressor
    train_df = df[complete_index]
    # known & unknow values
    known_values = np.array(train_df[train_df[missing_index].notnul
l()])
    unknow_values = np.array(train_df[train_df[missing_index].isnul
l()])

     # y is the know missing_index
    y = known_values[:, 0]

    # X are the features
    X = known_values[:, 1:]

    # fit
    rfr = RandomForestRegressor(random_state=0, n_estimators=2000,
n_jobs=-1)
    rfr.fit(X, y)

    # predict
    predicted_values = rfr.predict(unknow_values[:, 1:])

    print('fill in values of %s:'%missing_index, predicted_values)
     # fill in with predicted values
    df.loc[ (df[missing_index].isnull()), missing_index] = predicte
d_values
```

First, fill in missing values of '**census_block_group**' using complete attribute data ('date_range_start' & 'date_range_end')

In [14]:
```
new_data1 = data.copy()
set_missing_values(new_data1, ['census_block_group','date_range_start', 'date_range_end'])
new_data1.info()
```

```
fill in values of census_block_group: [2.87094781e+11]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220735 entries, 0 to 220734
Data columns (total 13 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   census_block_group        220735 non-null  float64
 1   date_range_start          220735 non-null  int64  
 2   date_range_end            220735 non-null  int64  
 3   raw_visit_count           220629 non-null  float64
 4   raw_visitor_count         220629 non-null  float64
 5   visitor_home_cbgs         220735 non-null  object 
 6   visitor_work_cbgs         220735 non-null  object 
 7   distance_from_home        220518 non-null  float64
 8   related_same_day_brand    220735 non-null  object 
 9   related_same_month_brand  220735 non-null  object 
 10  top_brands                220735 non-null  object 
 11  popularity_by_hour        220735 non-null  object 
 12  popularity_by_day         220735 non-null  object 
dtypes: float64(4), int64(2), object(7)
memory usage: 21.9+ MB
```

Second, fill in missing values of '**raw_visit_count**' using complete attribute data ('census_block_group' & 'date_range_start' & 'date_range_end')

In [15]:
```
new_data2 = new_data1.copy()
set_missing_values(new_data2, ['raw_visit_count', 'census_block_group', 'date_range_start', 'date_range_end'])
new_data2.info()
```

```
fill in values of raw_visit_count: [  7595.2457625     390.8717654
9   6497.85978906   1679.73450438
  12424.67968533   2771.83898465   3175.01409559  20572.75689855
    959.42987457  17022.25064952   2771.83898465  22016.07476694
 108579.88057456  12342.91229594   7688.83296719  14935.19125142
  12538.9684042   31906.87031033  21718.26844761   6717.99120972
   7854.15384241   4397.63862799  11074.71467845  37358.4800974
   2771.83898465  23116.57418909  15599.64176444   9513.23076504
   8756.59135326   8994.01865417  14287.08457106   8129.06377278
    658.07459942   2771.83898465  10135.51862857  80253.77036845
    631.08822489  22650.52523098  13808.11110321  37325.43464444
   2771.83898465   6512.46067736  54968.34996003  54968.34996003
 152677.64288805  18680.97897143  39196.13980331   7713.21051824
   1123.53470738   8840.10063222   7387.96028531  12581.16296011
  41323.37505575  27056.42601274  17583.26356352  18578.82119463
   2771.83898465  21718.26844761  42068.93086611    546.51578028
    215.89442727   8062.97575914  54968.34996003  24794.63086888
   4774.25449235    546.51578028   4988.69023456   5245.22784376
   5574.99468512   2004.09105253  27270.69847214    959.42987457
   5361.4439047    9882.26061239   5304.78975862   1123.53470738
    546.51578028    215.89442727  51320.6475632     631.08822489
  50836.29621795   7883.48651486  33623.03987672   5411.04562186
    215.89442727   5917.13570479  11627.22206282   8893.59940194
  11642.22956104    546.51578028   9835.13637055   3577.72610227
  38767.96847116  73000.83720207   9202.60161634   7494.18474876
   6195.26882012  22016.07476694   9763.14115572   1549.23292321
    215.89442727  35764.32759741  51113.43493318   1245.44578273
  14166.40524535  21769.34990809]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220735 entries, 0 to 220734
Data columns (total 13 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   census_block_group        220735 non-null  float64
 1   date_range_start          220735 non-null  int64
 2   date_range_end            220735 non-null  int64
 3   raw_visit_count           220735 non-null  float64
 4   raw_visitor_count         220629 non-null  float64
 5   visitor_home_cbgs         220735 non-null  object
 6   visitor_work_cbgs         220735 non-null  object
 7   distance_from_home        220518 non-null  float64
 8   related_same_day_brand    220735 non-null  object
 9   related_same_month_brand  220735 non-null  object
 10  top_brands                220735 non-null  object
 11  popularity_by_hour        220735 non-null  object
 12  popularity_by_day         220735 non-null  object
dtypes: float64(4), int64(2), object(7)
memory usage: 21.9+ MB
```

Third, fill in missing values of '**raw_visitor_count**' using complete attribute data ('raw_visit_count', 'census_block_group' & 'date_range_start' & 'date_range_end')

In [16]:
```
new_data3 = new_data2.copy()
set_missing_values(new_data3, ['raw_visitor_count', 'raw_visit_count', 'census_block_group', 'date_range_start', 'date_range_end'])
new_data3.info()
```

```
fill in values of raw_visitor_count: [ 1718.647       157.3315
1674.584       600.4095
  2900.6495      718.63496667   943.7705      4482.751
   442.48225893  4361.456       718.63496667  5051.921
 23490.9115      3242.861      1831.737625    3665.2365
  2190.667       5913.2225     4021.231       1260.548
  2367.226       1080.2965     3126.1265      4785.112
   718.63496667  5613.2465     3763.572       1791.554
  2353.2965      1691.1835     3784.687       1945.1135
   335.7635      718.63496667  2435.214      17517.518
   152.8315      4634.4085     2853.149       6848.433
   718.63496667  1619.3775    10317.231      10317.231
 27499.4775      4670.5255     8227.563       2014.72
   383.582       2070.7325     2106.0225      2145.8725
  9139.1095      6320.7565     3741.115       4135.2375
   718.63496667  4021.231      8761.919        302.916
   133.389       1869.9795    10317.231       6149.85
  1129.4415       302.916      1526.462       1407.4125
  1606.2385       612.705      6855.2765       442.48225893
  1270.682       2606.3305     1437.45         383.582
   302.916        133.389     10697.375        152.8315
  9997.079       1542.055      6719.085       1550.51
   133.389       1527.251      2803.1405      2849.454
  3086.51         302.916      2343.806        995.841
  9062.684      12188.147      2481.954       1718.3285
  1695.0635      5051.921      2507.1875       536.9565
   133.389       6180.6415    11164.5365       572.7105
  2073.498       4482.804     ]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220735 entries, 0 to 220734
Data columns (total 13 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   census_block_group        220735 non-null  float64
 1   date_range_start          220735 non-null  int64
 2   date_range_end            220735 non-null  int64
 3   raw_visit_count           220735 non-null  float64
 4   raw_visitor_count         220735 non-null  float64
 5   visitor_home_cbgs         220735 non-null  object
 6   visitor_work_cbgs         220735 non-null  object
 7   distance_from_home        220518 non-null  float64
 8   related_same_day_brand    220735 non-null  object
 9   related_same_month_brand  220735 non-null  object
 10  top_brands                220735 non-null  object
 11  popularity_by_hour        220735 non-null  object
 12  popularity_by_day         220735 non-null  object
dtypes: float64(4), int64(2), object(7)
memory usage: 21.9+ MB
```

Finally, fill in missing values of 'distance_from_home' using complete attribute data ('raw_visit_count', 'raw_visitor_count','census_block_group' & 'date_range_start' & 'date_range_end')

In [17]:
```
new_data = new_data3.copy()
set_missing_values(new_data, ['distance_from_home', 'raw_visitor_co
unt', 'raw_visit_count', 'census_block_group', 'date_range_start',
'date_range_end'])
new_data.info()

new_data_frequency = {key: new_data[key].value_counts() for key in
new_data.columns}
```

```
fill in values of distance_from_home: [1.24429215e+06 1.26321029e+
06 8.05086950e+03 7.35529740e+04
 7.14460520e+04 7.29168506e+05 3.94099665e+05 6.75989450e+03
 2.63665135e+04 7.75278850e+03 2.31526960e+04 1.06108755e+04
 2.62418195e+04 1.17695460e+04 1.77376283e+05 2.40199513e+05
 1.77652120e+05 8.66888171e+05 2.12964020e+04 2.41587610e+04
 6.76732485e+04 1.90962520e+04 1.22679382e+06 4.67829939e+05
 8.98527800e+03 3.19718530e+04 4.52585712e+05 4.39696173e+05
 3.57506875e+04 4.69177934e+05 4.68390224e+05 1.34856325e+04
 6.81400400e+03 4.44432475e+05 3.89385205e+05 8.99344650e+03
 4.46916729e+05 1.98744945e+05 7.11636865e+05 1.81265190e+04
 5.58958260e+04 4.13033339e+05 1.07479891e+05 1.77317155e+04
 6.52561544e+05 6.09299321e+05 2.40858748e+05 8.13717307e+05
 4.72683755e+05 2.28378050e+04 4.37019900e+03 5.45845510e+04
 3.20497571e+05 1.41880720e+04 6.23953199e+05 3.37132737e+05
 1.24749289e+06 9.14668250e+03 3.96533389e+05 1.01312620e+04
 1.10077685e+05 9.66420542e+05 2.42669325e+04 1.17648652e+05
 6.36296073e+05 3.13351435e+04 6.52799850e+03 6.71647400e+03
 5.35360900e+03 5.79292350e+05 6.02803345e+04 3.15062375e+04
 2.41569130e+04 1.36950495e+04 3.11793219e+05 8.69775610e+04
 1.02361831e+06 4.11128479e+05 6.80667338e+05 3.40794900e+04
 4.68304186e+05 1.73900205e+04 1.91620986e+05 6.33570042e+05
 3.39499020e+04 5.90554240e+05 7.95714750e+03 3.08781005e+04
 1.24650705e+06 5.70858159e+05 1.64001765e+04 1.25465026e+06
 2.15982355e+04 4.63418630e+04 7.67712407e+05 3.97397912e+05
 1.79113432e+05 1.76956455e+04 6.90466000e+03 1.12986770e+04
 5.41307080e+04 8.90316850e+03 2.82031108e+05 3.05084623e+05
 2.46899895e+04 2.14531106e+05 6.79205382e+05 2.23648610e+04
 6.12106050e+03 4.32993874e+05 1.53517705e+04 7.10645990e+04
 1.30845855e+04 9.80532050e+03 6.39573419e+05 3.04845000e+04
 5.71796000e+03 1.87390620e+04 5.62654735e+04 1.73104690e+04
 3.07379125e+04 5.71796000e+03 5.95188045e+04 2.40432365e+04
 7.11911720e+04 1.14651805e+04 1.07893075e+04 5.31441420e+04
 1.45523810e+04 4.24883115e+04 6.27794500e+04 1.17243665e+04
 1.30510300e+04 1.50768390e+04 3.53207615e+06 5.71796000e+03
 5.20886850e+04 6.54338465e+04 1.42642780e+04 7.76467650e+03
 2.27816925e+04 6.15660915e+04 8.89875450e+03 4.84131431e+05
 5.71796000e+03 3.31578685e+04 1.01923120e+04 7.08203750e+03
 4.82233545e+04 2.26195300e+04 5.34190000e+03 5.71796000e+03
```

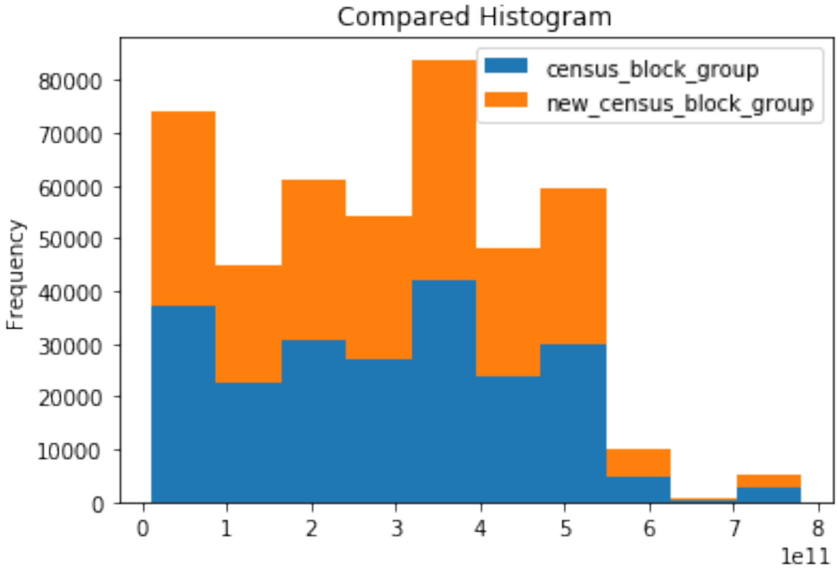
```
 1.24195785e+04 5.83249900e+03 5.83249900e+03 1.68151720e+04
 1.08937447e+05 6.47775100e+03 1.15646145e+04 1.13836835e+04
 1.24397480e+04 1.30659410e+04 3.96057920e+04 7.36913150e+03
 5.53806520e+04 1.01151405e+05 2.55391160e+04 5.71796000e+03
 4.24883115e+04 4.32947585e+04 4.91420708e+06 6.16309623e+05
 9.24476055e+04 5.83249900e+03 1.69314110e+04 7.49819650e+03
 4.91420708e+06 1.48663925e+04 8.24116050e+03 1.45145380e+04
 7.01977850e+03 2.46212930e+04 1.73104690e+04 8.99368800e+03
 3.98524680e+04 1.24734235e+04 1.13836835e+04 4.91420708e+06
 6.16309623e+05 7.45167115e+04 7.08203750e+03 2.51771515e+04
 2.72403375e+04 4.22693110e+04 1.44209575e+04 6.16309623e+05
 1.56799500e+04 2.01758775e+04 1.31173795e+04 1.28782910e+04
 4.91420708e+06 1.09557999e+06 1.32335610e+04 2.05369883e+05
 2.07319195e+04 1.04340320e+04 9.55683400e+03 1.25885610e+04
 5.95188045e+04 9.52717800e+03 3.39949136e+05 6.16309623e+05
 7.93635295e+04 4.56803250e+04 2.04009217e+05 3.23703176e+05
 5.74114330e+04]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220735 entries, 0 to 220734
Data columns (total 13 columns):
 #   Column                   Non-Null Count   Dtype  
---  ------                   --------------   -----  
 0   census_block_group       220735 non-null  float64
 1   date_range_start         220735 non-null  int64  
 2   date_range_end           220735 non-null  int64  
 3   raw_visit_count          220735 non-null  float64
 4   raw_visitor_count        220735 non-null  float64
 5   visitor_home_cbgs        220735 non-null  object 
 6   visitor_work_cbgs        220735 non-null  object 
 7   distance_from_home       220735 non-null  float64
 8   related_same_day_brand   220735 non-null  object 
 9   related_same_month_brand 220735 non-null  object 
 10  top_brands               220735 non-null  object 
 11  popularity_by_hour       220735 non-null  object 
 12  popularity_by_day        220735 non-null  object 
dtypes: float64(4), int64(2), object(7)
memory usage: 21.9+ MB
```

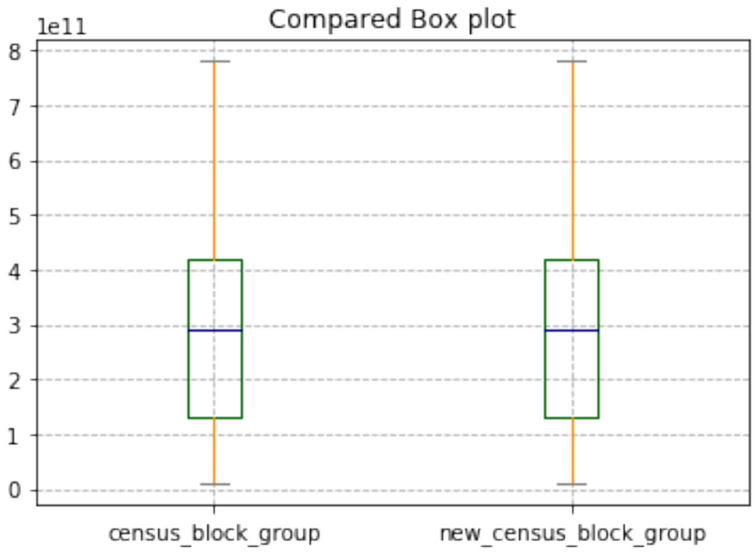
compare with raw data

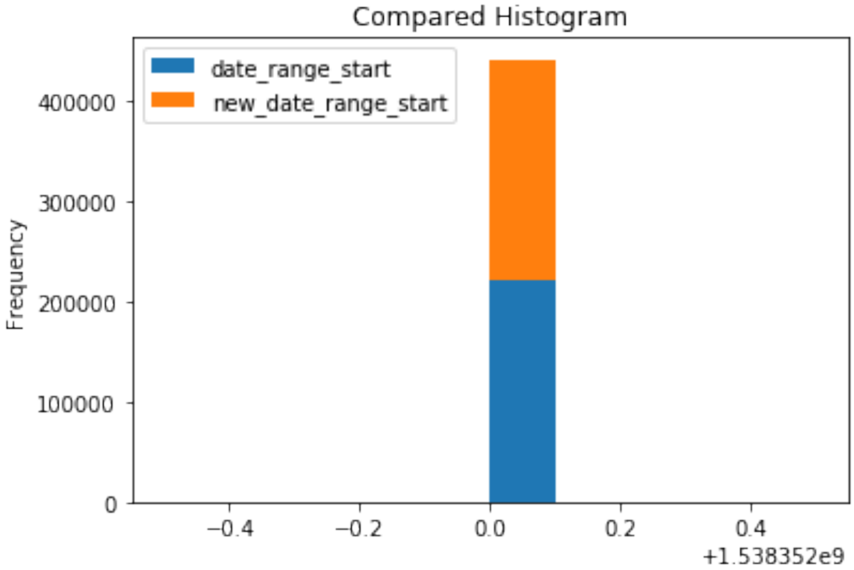
There is no missing data of nominal data, so we need no visualization numerical data compared visualization

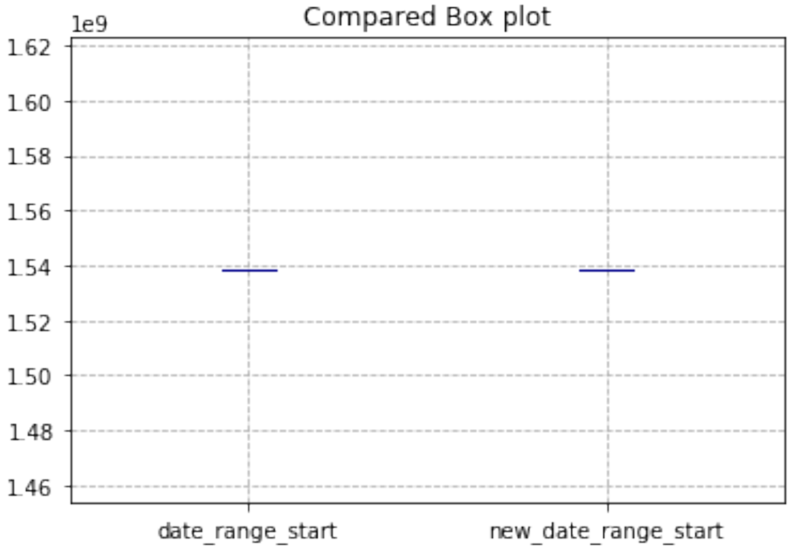
- Compared histogram
- Compared box plot
- Compared scatter plot

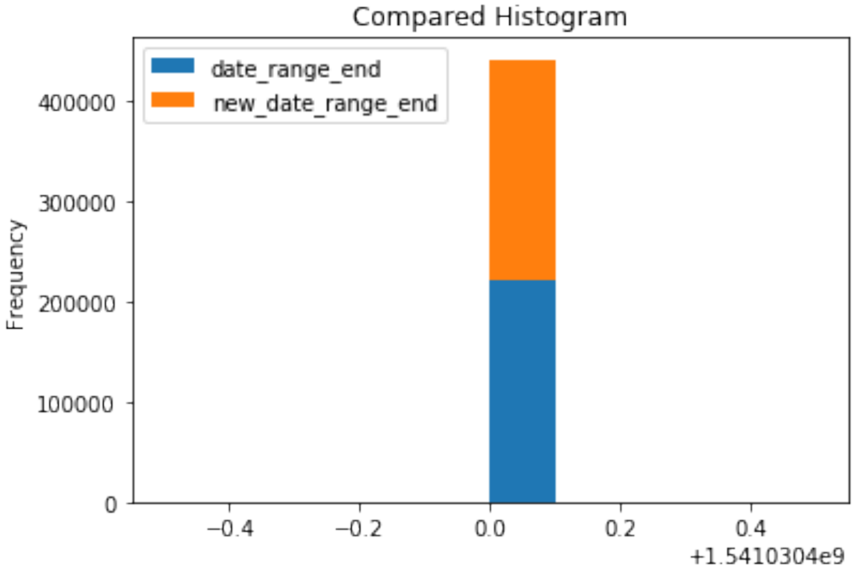
In [19]:
```
for i in numerical_index:
    histogram_compare(data[i], new_data[i])
    box_plot_compare(data[i], new_data[i])

scatter_plot_compare(data, new_data, 'raw_visit_count', 'raw_visitor_count')
scatter_plot_compare(data, new_data, 'raw_visit_count', 'distance_from_home')
```

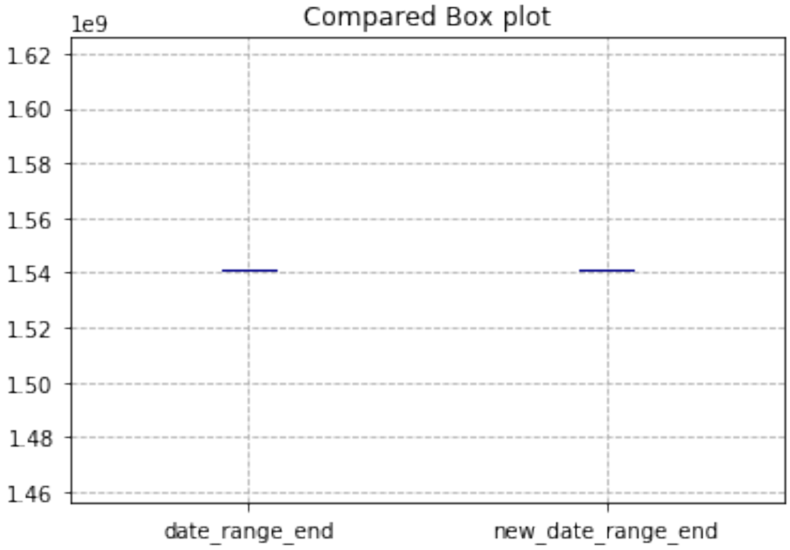


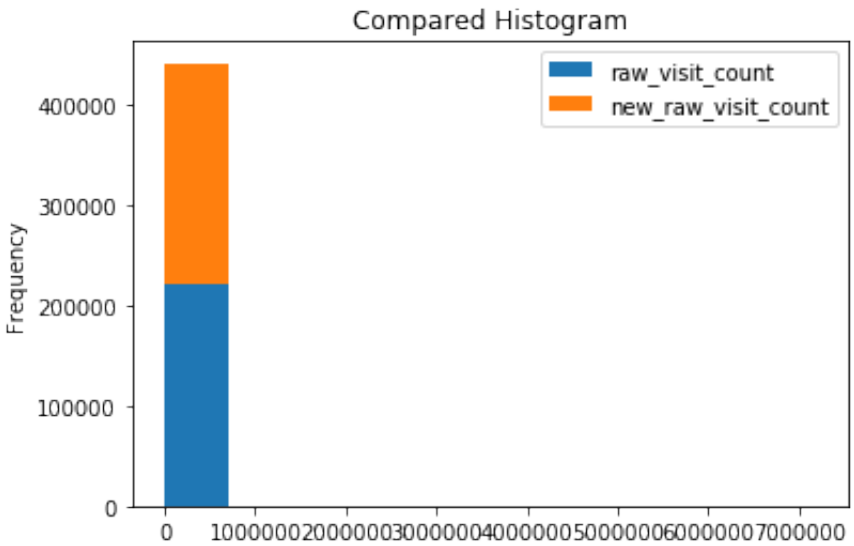


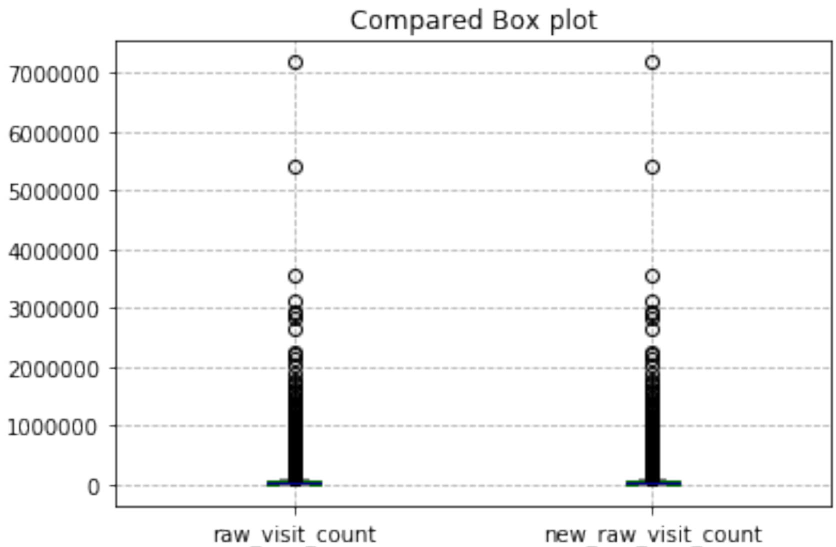

Compared Histogram
date_range_start
new_date_range_start
Frequency
400000
300000
200000
100000
0
−0.4
−0.2
0.0
0.2
0.4
+1.538352e9

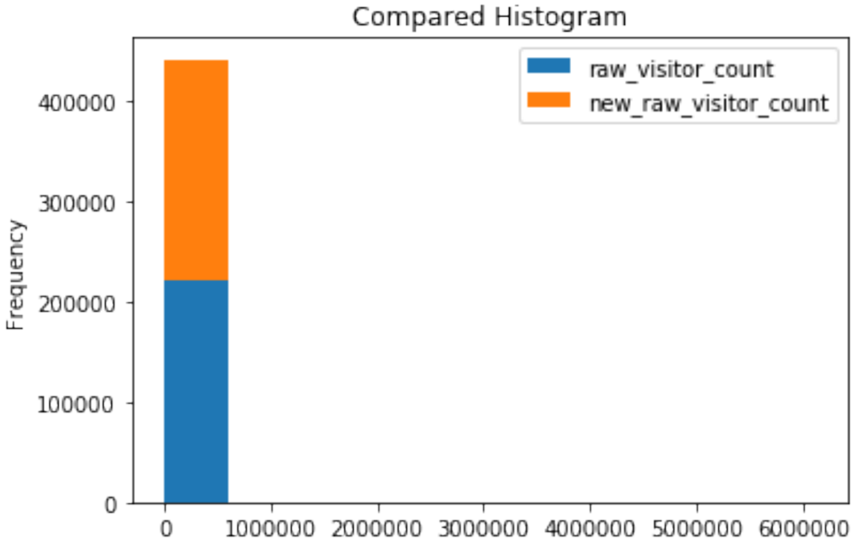

Compared Box plot
1e9
1.62
1.60
1.58
1.56
1.54
1.52
1.50
1.48
1.46
date_range_start
new_date_range_start

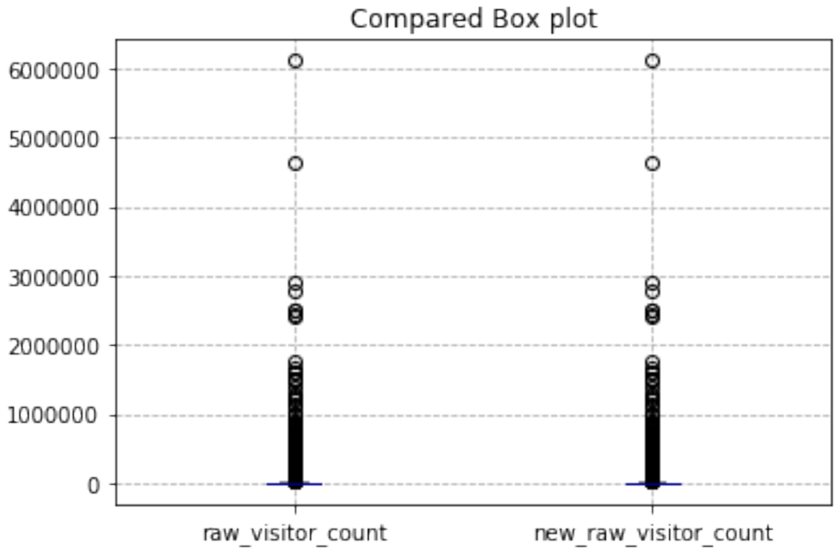

Compared Histogram
date_range_end
new_date_range_end
Frequency
400000
300000
200000
100000
0
−0.4
−0.2
0.0
0.2
0.4
+1.5410304e9

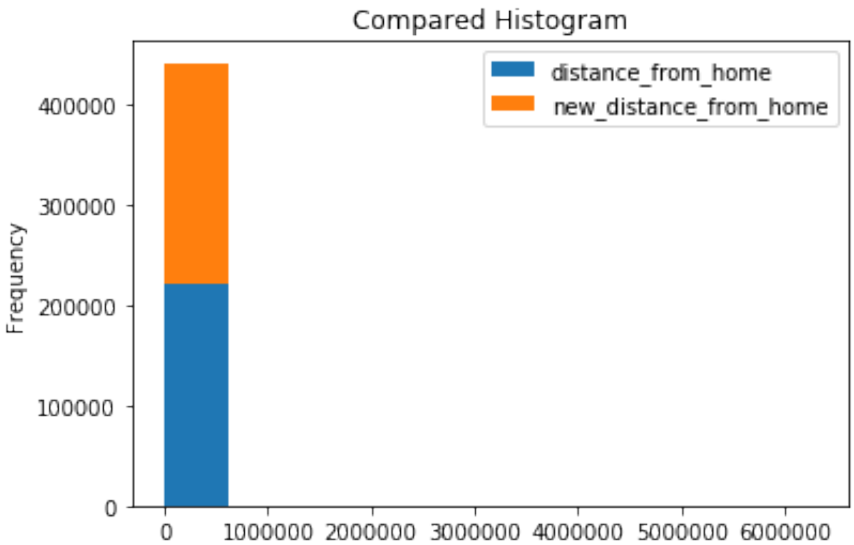

Compared Box plot
1e9
1.62
1.60
1.58
1.56
1.54
1.52
1.50
1.48
1.46
date_range_end
new_date_range_end

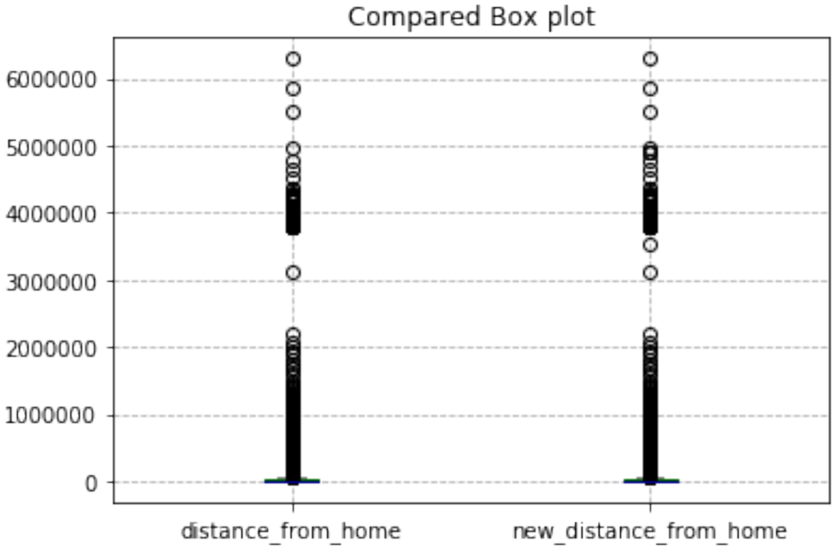

Compared Histogram
raw_visit_count
new_raw_visit_count
Frequency
400000
300000
200000
100000
0
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000

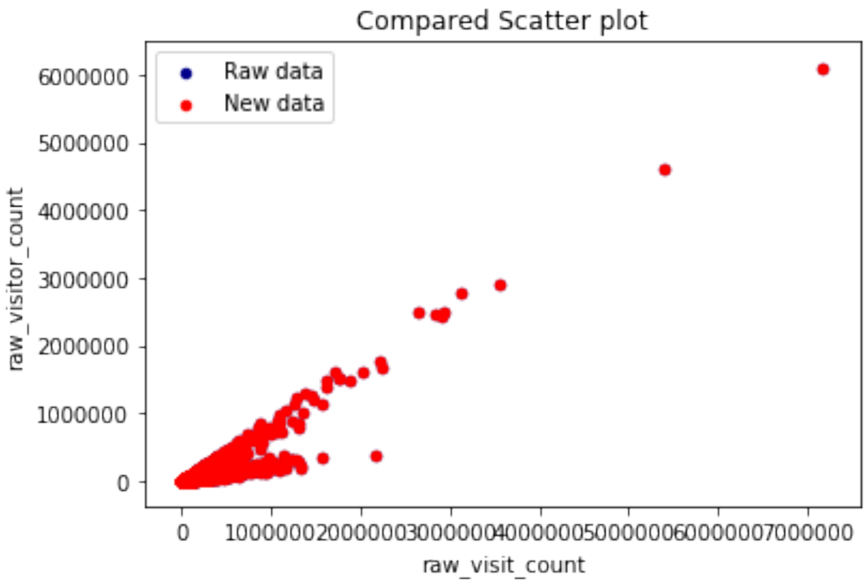

Compared Box plot
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
raw_visit_count
new_raw_visit_count

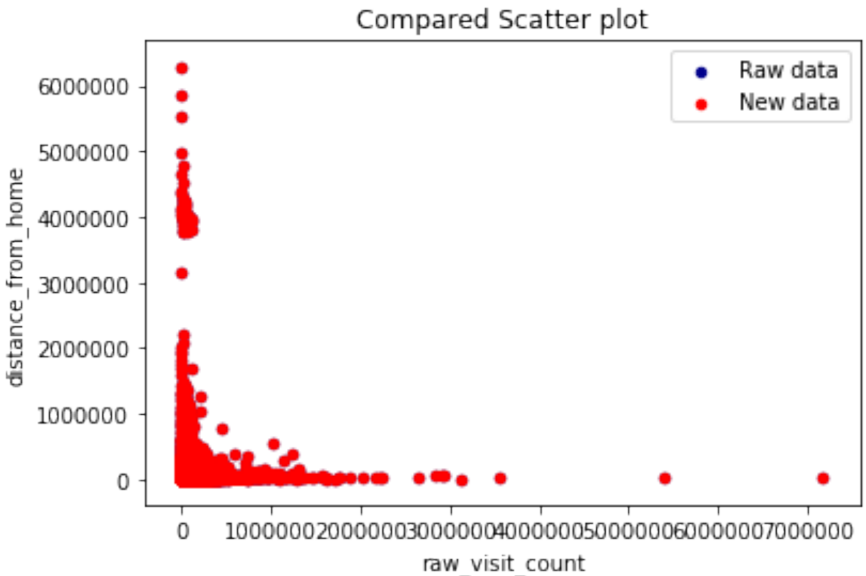

Compared Histogram
raw_visitor_count
new_raw_visitor_count
Frequency
0
100000
200000
300000
400000
0
1000000
2000000
3000000
4000000
5000000
6000000


Compared Box plot
6000000
5000000
4000000
3000000
2000000
1000000
0
raw_visitor_count
new_raw_visitor_count


Compared Histogram
distance_from_home
new_distance_from_home
Frequency
0
100000
200000
300000
400000
0
1000000
2000000
3000000
4000000
5000000
6000000


Compared Box plot
6000000
5000000
4000000
3000000
2000000
1000000
0
distance_from_home
new_distance_from_home


Compared Scatter plot
Raw data
New data
raw_visitor_count
6000000
5000000
4000000
3000000
2000000
1000000
0
0 1000000 2000000 3000000 4000000 5000000 6000000 7000000
raw_visit_count


Compared Scatter plot
Raw data
New data
distance_from_home
6000000
5000000
4000000
3000000
2000000
1000000
0
0 1000000 2000000 3000000 4000000 5000000 6000000 7000000
raw_visit_count

2.4 Fill in missing values by similarity between data objects

KNN

find most similar object and fill in missing valus

In [44]:
```
import sys
from impyute.imputation.cs import fast_knn

# start the KNN training
imputed_training=fast_knn(data[numerical_index].values, k=30)
imputed_data = pd.DataFrame(imputed_training, columns=numerical_ind
ex)
nn_data = data.copy()
for index in numerical_index:
    nn_data.loc[:, index] = imputed_data[index]
nn_data.info()

nn_data_frequency = {key: nn_data[key].value_counts() for key in nn
_data.columns}
```

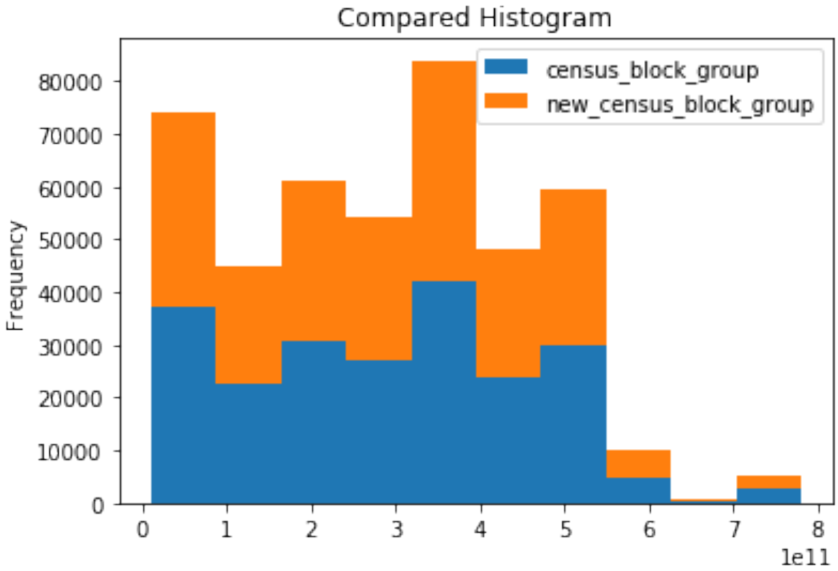
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220735 entries, 0 to 220734
Data columns (total 13 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   census_block_group        220735 non-null  float64
 1   date_range_start          220735 non-null  float64
 2   date_range_end            220735 non-null  float64
 3   raw_visit_count           220735 non-null  float64
 4   raw_visitor_count         220735 non-null  float64
 5   visitor_home_cbgs         220735 non-null  object
 6   visitor_work_cbgs         220735 non-null  object
 7   distance_from_home        220735 non-null  float64
 8   related_same_day_brand    220735 non-null  object
 9   related_same_month_brand  220735 non-null  object
 10  top_brands                220735 non-null  object
 11  popularity_by_hour        220735 non-null  object
 12  popularity_by_day         220735 non-null  object
dtypes: float64(6), object(7)
memory usage: 21.9+ MB
```

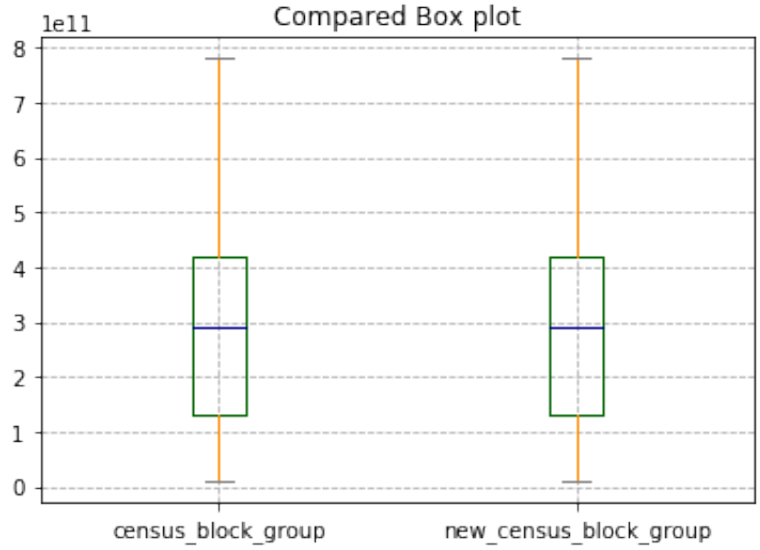
compare with raw data

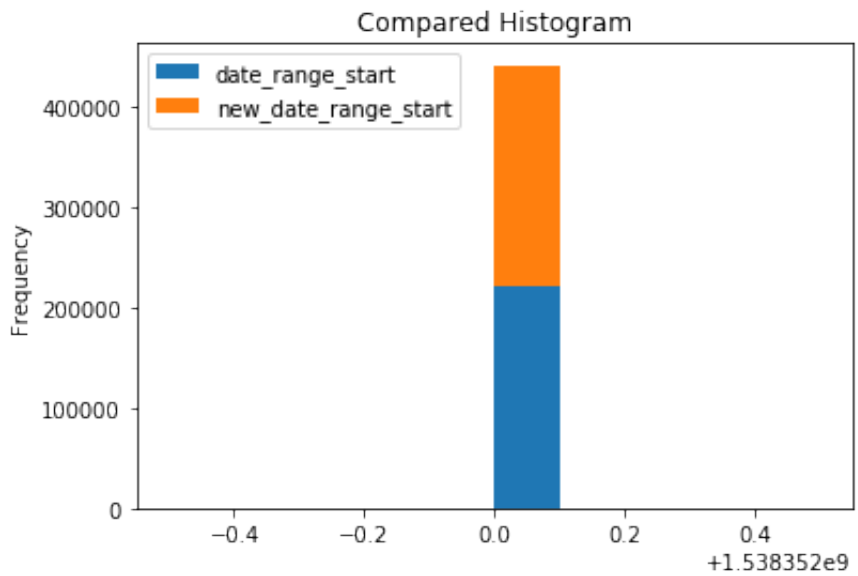
There is no missing data of nominal data, so we need no visualization numerical data compared visualization

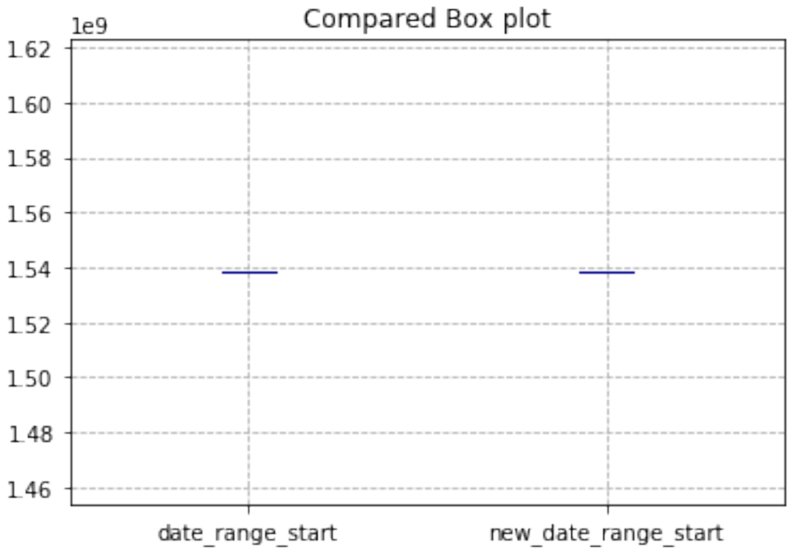
- Compared histogram
- Compared box plot
- Compared scatter plot

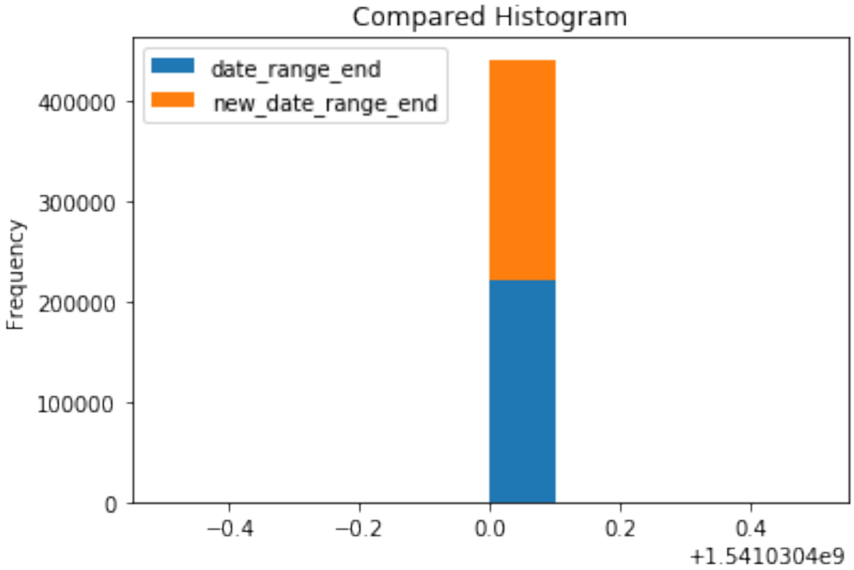
In [46]:
```
for i in numerical_index:
    histogram_compare(data[i], new_data[i])
    box_plot_compare(data[i], new_data[i])

scatter_plot_compare(data, nn_data, 'raw_visit_count', 'raw_visitor
_count')
scatter_plot_compare(data, nn_data, 'raw_visit_count', 'distance_fr
om_home')
```

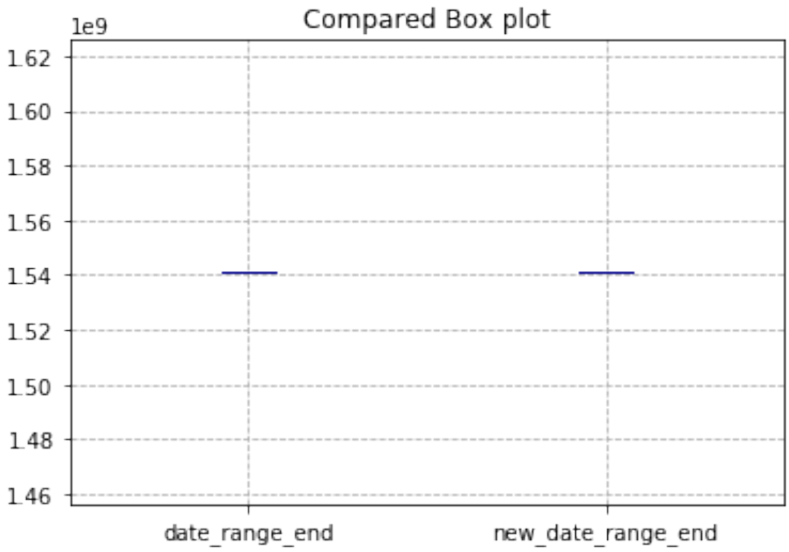


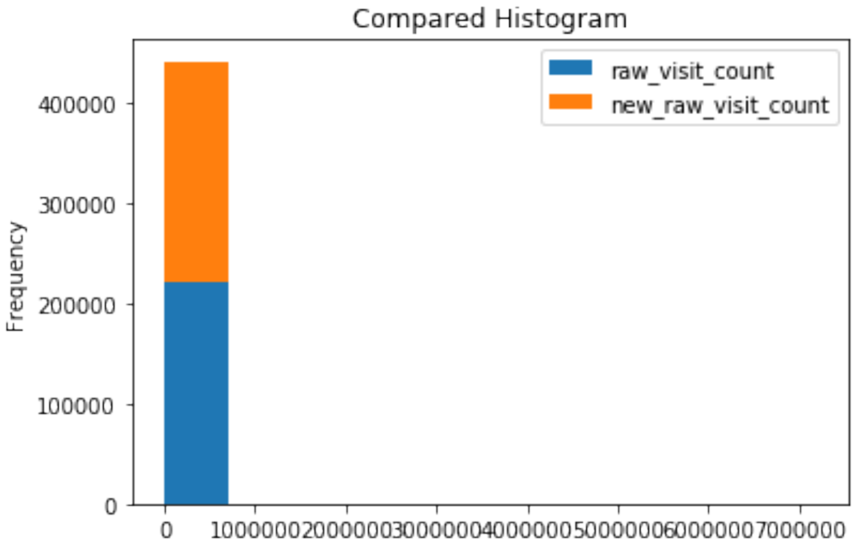

Compared Box plot
1e11
8
7
6
5
4
3
2
1
0
census_block_group
new_census_block_group

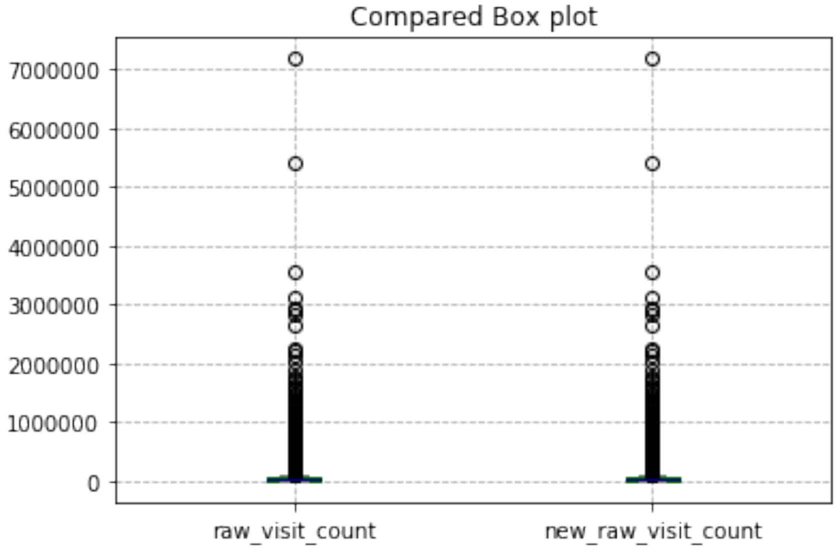

Compared Histogram
date_range_start
new_date_range_start
Frequency
400000
300000
200000
100000
0
−0.4
−0.2
0.0
0.2
0.4
+1.538352e9

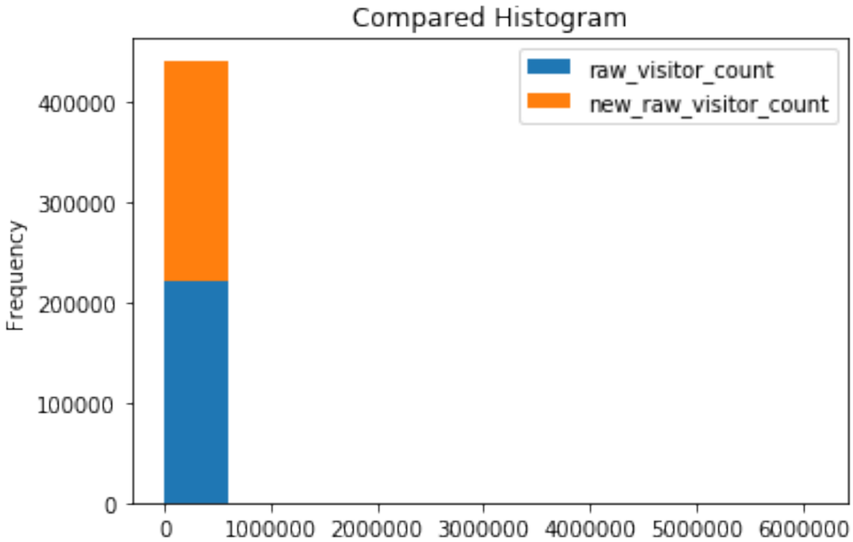

Compared Box plot
1e9
1.62
1.60
1.58
1.56
1.54
1.52
1.50
1.48
1.46
date_range_start
new_date_range_start

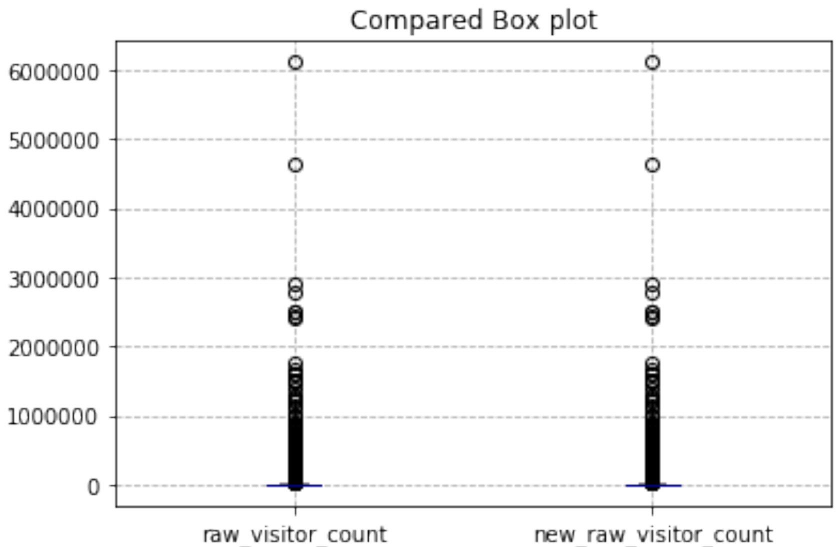

Compared Histogram
date_range_end
new_date_range_end
Frequency
400000
300000
200000
100000
0
−0.4
−0.2
0.0
0.2
0.4
+1.5410304e9

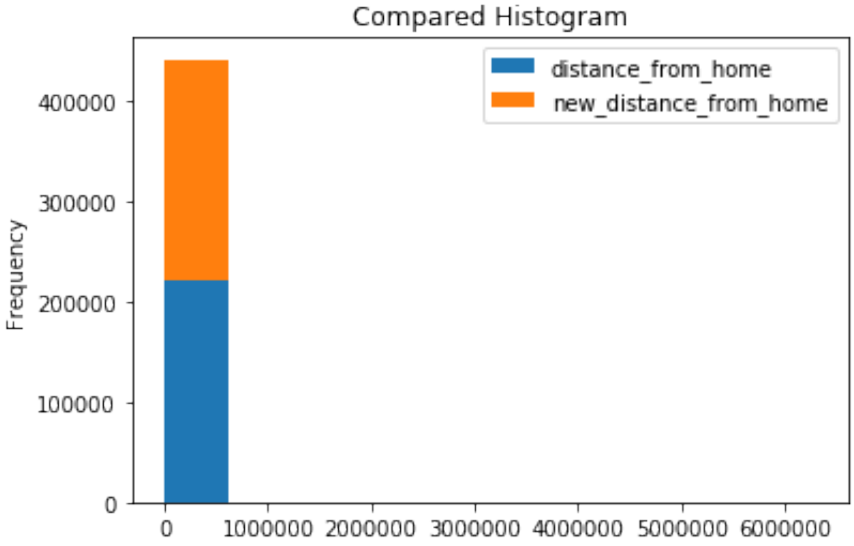

Compared Box plot
1e9
1.62
1.60
1.58
1.56
1.54
1.52
1.50
1.48
1.46
date_range_end
new_date_range_end

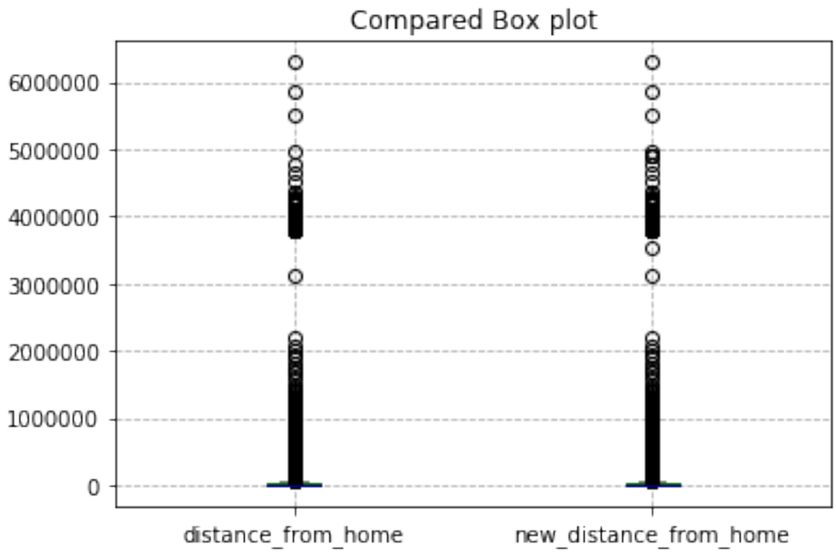

Compared Histogram
raw_visit_count
new_raw_visit_count
Frequency
400000
300000
200000
100000
0
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000

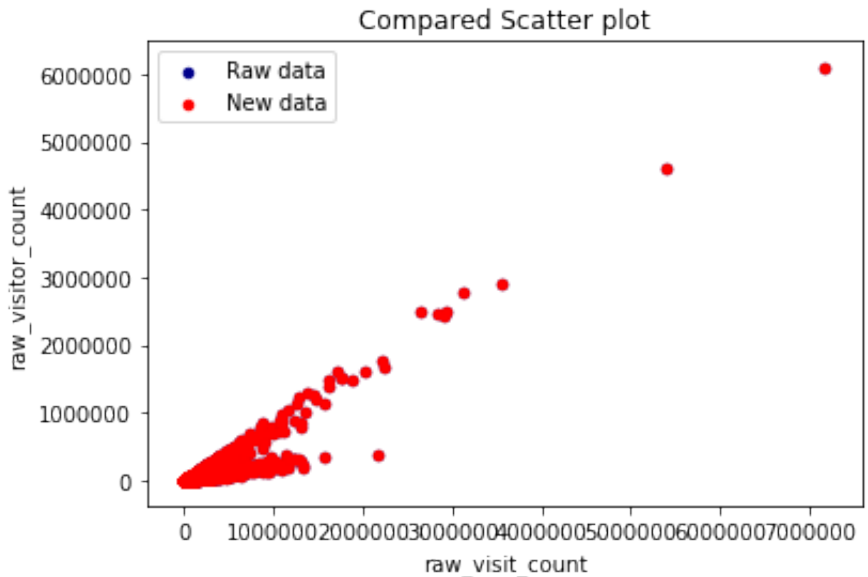

Compared Box plot
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
raw_visit_count
new_raw_visit_count

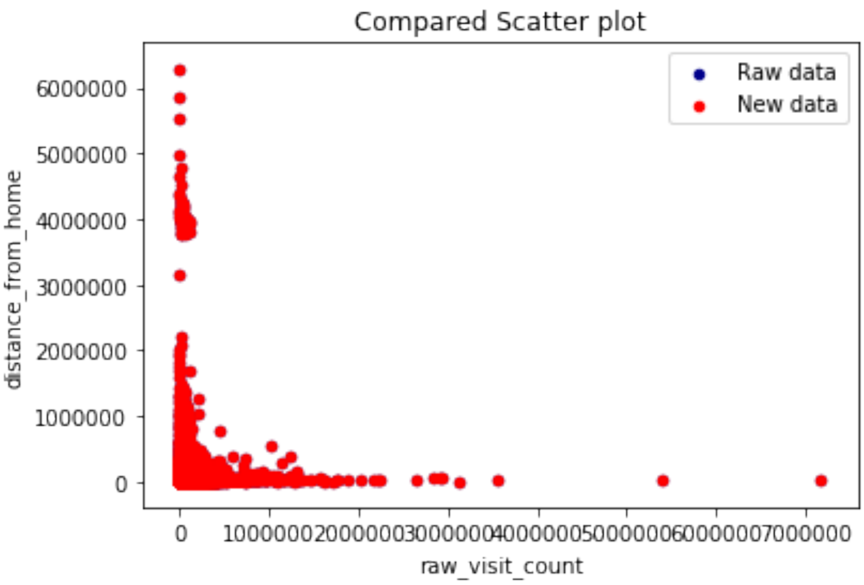

Compared Histogram
raw_visitor_count
new_raw_visitor_count
Frequency
400000
300000
200000
100000
0
0
1000000
2000000
3000000
4000000
5000000
6000000


Compared Box plot
6000000
5000000
4000000
3000000
2000000
1000000
0
raw_visitor_count
new_raw_visitor_count


Compared Histogram
distance_from_home
new_distance_from_home
Frequency
400000
300000
200000
100000
0
0
1000000
2000000
3000000
4000000
5000000
6000000


Compared Box plot
6000000
5000000
4000000
3000000
2000000
1000000
0
distance_from_home
new_distance_from_home


Compared Scatter plot
Raw data
New data
raw_visitor_count
6000000
5000000
4000000
3000000
2000000
1000000
0
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
raw_visit_count


Compared Scatter plot
Raw data
New data
distance_from_home
6000000
5000000
4000000
3000000
2000000
1000000
0
raw_visit_count
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000