In [ ]:

```
"""
##################  Week 11 & 12 Final Term End Project - Milestone 5 ##################

### Merging the Data and Storing in a Database/Visualizing Data
# Now that you have cleaned and transformed your 3 datasets, you need to load them into a database. You can cho
# You will want to load each dataset into SQL Lite as an individual table and then you must join the datasets t
# Once all the data is merged together in your database, create 5 visualizations that demonstrate the data you
# You should have at least 2 visualizations that have data from more than one source (meaning, if you have 3 ta
# also welcome to use your consolidated dataset that you created in the previous step, if you do that, you have



##################      Some great Learnings from (Milestone - 5) exercise.   ##################

The Data Wrangling course provided me with valuable learning experiences. I gained a deeper understanding of th
Data Science project. This course has significantly boosted my confidence in handling data clean-up and formula
While I had some prior experience with pandas and Python language at work and in the DSC530 (Exploratory Data A
as well as numpy and matplotlib. The course has covered essential topics such as Fuzzy Matching, Hierarchical I
skill set. The textbooks associated with the course were particularly helpful. Additionally, the weekly posts o
learning process engaging and dynamic. I made a conscious effort to take notes on intriguing topics, further en
file datasets and API datasets was easy part. I found web scraping little tricky as it relies on html DOM (elem
websites to find any discrepancy. Data cleaning, adding column names, standardizing names etc. tasks were reall
The visualizations created with Seaborn and Matplotlib libraries were exceptionally well-crafted, providing me 
visualization has always posed a challenge for me, but I am confident that with experience, this skill will imp
to be a significant aspect of the overall learning experience. One specific challenge I faced was positioning t
relying on Google as a constant resource proved to be immensely helpful, reinforcing the notion that it remains

"""



from __future__ import print_function
from itertools import zip_longest

import csv
import logging
import sys
import numpy as np
import pandas as pd
import random
import thinkplot
import thinkstats2
import datetime
import regression
import statsmodels.formula.api as smf
import statsmodels.api as sm
import matplotlib.pyplot as plt
import math
from bs4 import BeautifulSoup
import pandas as pd
import re
from datetime import date
from Levenshtein import distance
import plotly.express as px
import warnings
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
import re
import json
import tweepy
from requests_oauthlib import OAuth1Session
import os
import json
from textblob import TextBlob
import urllib.request, urllib.parse, urllib.error
import requests
import datadotworld as dw
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt
import seaborn as sns
from bs4 import BeautifulSoup as bs
%matplotlib inline
warnings.filterwarnings('ignore', category=FutureWarning)
import sqlite3

######### MileStone - 2 #################################

def ReadData_MileStone2(filename):
    ### Read in the Airbnb data set (given as a .csv file) from the local directory

    dfData = pd.read_csv(filename)

    # printing the headers
    names = []
    for line in dfData:
        var=line.split(":")[0]
        names.append(var)
    #print("Printing existing headers :", names);

    return dfData

# to Replace Headers
def Replace_Headers_MileStone2(df, filename):
    df.head()

    # adding header
    headerList = ['Airbnb_Id', 'Listing_URL', 'Scrape_Id', 'Last_Scraped','Source','Name','Description','Neighb
'Host_Is_Superhost','Host_Thumbnail_URL','Host_Picture_URL', 'Host_Neighbourhood', 'Host_Listings_Count', 'Host_
'Neighbourhood_Group_Cleansed', 'Latitude', 'Longitude', 'Property_type', 'Room_type', 'Accommodates', 'Bathroo
'Maximum_Maximum_Nights', 'Minimum_Nights_Avg_ntm', 'Maximum_Nights_Avg_ntm', 'Calendar_Updated', 'Has_Availabi
'Availability_365', 'Calendar_Last_Scraped', 'Number_of_reviews', 'Number_of_reviews_ltm', 'Number_of_reviews_l
'Review_scores_communication', 'Review_scores_location', 'Review_scores_value', 'License', 'Instant_bookable', 
'Calculated_host_listings_count_shared_rooms', 'Reviews_per_month']

    # converting data frame to csv
    df.to_csv(filename, header=headerList, index=False)

    # display modified csv file
    modifiedData = pd.read_csv(filename)
    #print('\nModified file:')
    #print(modifiedData)
    return modifiedData

# Format data into a more readable format
def Format_Data_MileStone2(df):

    # format the values as currency in thousands
    df["Scrape_Id"] = df["Scrape_Id"] / 1000
    df["Scrape_Id"] = df["Scrape_Id"].map("${:,.0f}K".format)

    df["Airbnb_Id"] = df["Airbnb_Id"] / 1000
    df["Airbnb_Id"] = df["Airbnb_Id"].map("${:,.0f}K".format)

    df['Last_Scraped'] =  pd.to_datetime(df['Last_Scraped'])
    df['Calendar_Last_Scraped'] = pd.to_datetime(df['Calendar_Last_Scraped'])

    # replace blank and Null values with 'None'
    df.fillna('None', inplace=True)
    df.to_csv("filename.csv", index=False)

    #print(df)
    return df

# Identify outliers and bad data
# Find duplicates
def Identify_Outliers_Duplicates_MileStone2(df):
    #drop the unnecessary columns
    df = df.drop(columns=(['Availability_60', 'Availability_90', 'Availability_365','Calendar_Last_Scraped', 'M
    df.describe()
    # print(df)

    df.head()
    #print("Scrape Id is duplictated - {}".format(any(df.Scrape_Id.duplicated())))
    #print("Name is duplictated - {}".format(any(df.Name.duplicated())))
    #print("Host Url is duplictated - {}".format(any(df.Host_URL.duplicated())))
    #print("Host Name is duplictated - {}".format(any(df.Host_Name.duplicated())))
    #print("Host Id is duplictated - {}".format(any(df.Host_Id.duplicated())))
    #print("Host Location is duplictated - {}".format(any(df.Host_Location.duplicated())))
    #print("Host Neighbourhood is duplictated - {}".format(any(df.Host_Neighbourhood.duplicated())))
    #print("Host Verifications is duplictated - {}".format(any(df.Host_Verifications.duplicated())))

    ### Check if any essential column contains NaN.

    #print("The column Scrape Id contains NaN - %r " % df.Scrape_Id.isnull().values.any())
    #print("The column Name contains NaN - %s " % df.Name.isnull().values.any())
    #print("The column Host URL contains NaN - %s " % df.Host_URL.isnull().values.any())
    #print("The column Host Name contains NaN - %s " % df.Host_Name.isnull().values.any())
    #print("The column Host Id contains NaN - %s " % df.Host_Id.isnull().values.any())
    #print("The column Host Location contains NaN - %s " % df.Host_Location.isnull().values.any())
    #print("The column Host Neighbourhood contains NaN - %s " % df.Host_Neighbourhood.isnull().values.any())
    #print("The column Host Verifications contains NaN - %s " % df.Host_Verifications.isnull().values.any())


    ### Create a box plot to check for outliers.
    #plt.boxplot(df.Host_Id, notch=True)

    return df

# Fix casing or inconsistent values
def Fix_Casing_Inconsitentvalues_MileStone2(df):
    # Lower dest_region column
    df['Listing_URL'] = df['Listing_URL'].str.lower()
    df['Host_Response_Time'] = df['Host_Response_Time'].str.lower()
    df['Host_Name'] = df['Host_Name'].str.upper()

    # Verify changes have been effected
    print(df['Listing_URL'].unique())
    print(df['Host_Name'].unique())

    df.to_csv("filename.csv", index=False)

    #print(df)
    return df

# Conduct Fuzzy Matching
def Conduct_Fuzzy_Matching_MileStone2(df):
    host_location = 'Miami, FL1'
    for row in df['Host_Location']:
        print("{} {}".format(row, distance(host_location, row)))

def strip_whitespace_MileStone2(s):
    return s.strip()

def Perform_MileStone2_Operation():
     ### Read in the Airbnb project dataset (given as a .csv file) from the local direction:
    filename="listings.csv"
    df = ReadData_MileStone2(filename);

    ### To replace headers on the data sets
    dfData_MileStone2 = Replace_Headers_MileStone2(df, filename)

    ### Format data into a more readable format
    dfData_MileStone2 = Format_Data_MileStone2(dfData_MileStone2)

    ### Identify outliers and bad data and duplicate values
    dfData_MileStone2 = Identify_Outliers_Duplicates_MileStone2(dfData_MileStone2)

     ### Identify Fix casing and inconsitentvalues
    dfData_MileStone2 = Fix_Casing_Inconsitentvalues_MileStone2(dfData_MileStone2)

    dfData_MileStone2 = dfData_MileStone2.head(10)
    dfData_MileStone2.insert(0, 'customerNo', range(1, 1 + len(dfData_MileStone2)))

    dfData_MileStone2.at[5 ,'customerNo']=1
    dfData_MileStone2.at[6 ,'customerNo']=1
    return dfData_MileStone2

######### MileStone - 3 #################################


def ReadData_MileStone3():
    # Load the HTML dom structure in soup library which will parse the elements
    # Inspecting the web page in chrome developer view shows that the table where data is present has CSS class
    #table_MN = pd.read_html('http://ufcstats.com/fight-details/bb15c0a2911043bd')
    #print(f'Total tables: {len(table_MN)}')
    #dfData = table_MN[0].head()
    #display(dfData)
    #return dfData

    url = requests.get('http://ufcstats.com/fight-details/bb15c0a2911043bd')
    soup = bs(url.content, 'lxml')
    table = soup.select_one(
    '.js-fight-section:has(p:-soup-contains("Significant Strikes")) + table')

    dfData = pd.DataFrame(
        [[i.text.strip() for i in table.select(f'tr:nth-child(1) td p:nth-child({n+1})')]
          for n, _ in enumerate(table.select('tr:nth-child(1) > td:nth-child(1) > p'))], columns=[i.text.strip(

    #display(dfData)
    return dfData

# Column names contains extra \n at the begining. I will perform following cleanup on column names
# Remove "\n" from the begining of name
# Replace spaces " " and parentheis "(" and dashes "-" with _ (underscores)
# Change the case of names to lowercase
# This function will change the parameter passed to follow naming standard
def Naming_Convention_MileStone3(name):
   char_to_replace = {
       " ":"_",
       "-":"_",
       "(":"_",
       ")":"_"
   }
   # strip method will remove \n and any leading/trailing whitespaces
   better_name = name.strip().lower()
   # Replace string place holders according to values in dictionary
   better_name = better_name.translate(str.maketrans(char_to_replace))
   return better_name.rstrip('_')

# Define a function to find similar names
def Find_Similar_Name_MileStone3(name, choices):
    return process.extractOne(name, choices)

def Format_Data_Readable_Format_MileStone3(dfData):
    #print("dtypes ",dfData.dtypes)
```

```python
    #print(dfData)
    # Replace empty string ('') with np.nan before convertion
    dfData['ufcfighter_name']=dfData.ufcfighter_name.replace('',np.nan)
    dfData['sig_str']=dfData.sig_str.replace('',np.nan)
    dfData['sig_str_percentage']=dfData.sig_str_percentage.replace('',np.nan)
    dfData['head_count']=dfData.head_count.replace('',np.nan)
    dfData['body_count']=dfData.body_count.replace('',np.nan)
    dfData['leg_count']=dfData.leg_count.replace('',np.nan)
    dfData['distance_count']=dfData.distance_count.replace('',np.nan)
    dfData['clinch_count']=dfData.clinch_count.replace('',np.nan)
    dfData['ground_count']=dfData.ground_count.replace('',np.nan)

    # Removing any leading and trailing spaces and coverting state names to uppercase
    dfData['ufcfighter_name'] = dfData.ufcfighter_name.str.strip()
    dfData['sig_str'] = dfData.sig_str.str.strip()
    dfData['sig_str_percentage'] = dfData.sig_str_percentage.str.strip()
    dfData['head_count'] = dfData.head_count.str.strip()
    dfData['body_count'] = dfData.body_count.str.strip()
    dfData['ufcfighter_name'].head()

    return dfData

def Find_Duplicates_MileStone3(dfData):

    # To find duplicates on specific column 'procurement_id'
    #print("dfData.duplicated column 'ufcfighter_name' ", dfData.duplicated(subset=['ufcfighter_name']))
    #print("dfData.duplicated column 'sig_str' ", dfData.duplicated(subset=['sig_str']))
    #print("dfData.duplicated column 'sig_str_percentage' ", dfData.duplicated(subset=['sig_str_percentage']))
    #print("dfData.duplicated column 'head_count' ", dfData.duplicated(subset=['head_count']))
    #print("dfData.duplicated column 'body_count' ", dfData.duplicated(subset=['body_count']))
    #print("dfData.duplicated column 'leg_count' ", dfData.duplicated(subset=['leg_count']))
    #print("dfData.duplicated column 'distance_count' ", dfData.duplicated(subset=['distance_count']))
    #print("dfData.duplicated column 'clinch_count' ", dfData.duplicated(subset=['clinch_count']))
    #print("dfData.duplicated column 'ground_count' ", dfData.duplicated(subset=['ground_count']))

    #print("ufcfighter name is duplictated - {}".format(any(dfData.ufcfighter_name.duplicated())))
    #print("sig_str is duplictated - {}".format(any(dfData.sig_str.duplicated())))
    #print("sig_str_percentage is duplictated - {}".format(any(dfData.sig_str_percentage.duplicated())))
    #print("head_count is duplictated - {}".format(any(dfData.head_count.duplicated())))
    #print("body_count is duplictated - {}".format(any(dfData.body_count.duplicated())))
    #print("leg_count is duplictated - {}".format(any(dfData.leg_count.duplicated())))

    # DROP duplicates data
    dfData.drop_duplicates(subset=['sig_str_percentage'], keep='last')

    # By default it removes duplicate rows based on all columns.
    dfData.drop_duplicates()

    return dfData

def Identify_Outliers_BadData_MileStone3(dfData):

    ### Check if any essential column contains NaN.

    print("The column ufcfighter_name contains NaN - %r " % dfData.ufcfighter_name.isnull().values.any())
    print("The column sig_str contains NaN - %s " % dfData.sig_str.isnull().values.any())
    print("The column sig_str_percentage contains NaN - %s " % dfData.sig_str_percentage.isnull().values.any())
    print("The column head_count contains NaN - %s " % dfData.head_count.isnull().values.any())
    print("The column body_count contains NaN - %s " % dfData.body_count.isnull().values.any())
    print("The column leg_count contains NaN - %s " % dfData.leg_count.isnull().values.any())

    # Create a box plot to check for outliers.
    plt.boxplot(dfData.head_count, notch=True)

    # Find outliers and view the data distribution using a histogram
    fig = px.histogram(dfData, x='head_count')
    fig.show()

    # Find multivariate outliers using a scatter plot
    fig_1 = px.scatter(x=dfData['head_count'], y=dfData['head_count'])
    fig_1.show()

def Fix_Casing_InconsistentValues_MileStone3(dfData):

    # Lower name column. Fix capitalization inconsistencies
    dfData['ufcfighter_name'] = dfData['ufcfighter_name'].str.lower()
    dfData['sig_str'] = dfData['sig_str'].str.lower()
    dfData['sig_str_percentage'] = dfData['sig_str_percentage'].str.lower()
    dfData['head_count'] = dfData['head_count'].str.lower()
    dfData['body_count'] = dfData['body_count'].str.lower()

    # Verify changes have been effected
    dfData_name = dfData['ufcfighter_name'].unique()

    # Fix whitespace if any

    dfData_name.sort()
    #print("dfData_name sort : ", dfData_name)
    # Remove white spaces from `dest_size`
    dfData['ufcfighter_name'] = dfData['ufcfighter_name'].str.strip()
    dfData['sig_str'] = dfData['sig_str'].str.strip()
    dfData['sig_str_percentage'] = dfData['sig_str_percentage'].str.strip()
    dfData['head_count'] = dfData['head_count'].str.strip()
    dfData['body_count'] = dfData['body_count'].str.strip()
    # Verify changes have been effected
    #print(dfData['ufcfighter_name'].unique())

    return dfData

def Conduct_Fuzzy_Matching_MileStone3(dfData):

    #Use fuzzy matching to correct inconsistent data entry

    # get all the unique values in the 'dest_region' column
    dfData_name = dfData['procurement_num'].unique()
    # sort them alphabetically and then take a closer look
    dfData_name.sort()

    vendorname = 'JES CONSTRUCTION INC.'
    for row in dfData['vendor_name']:
        print("{} {}".format(row, distance(vendorname, row)))

    # Create a list of choices for name matching
    procurement_choices = dfData['procurement_id'].tolist()

    # Apply the function to each row in the 'Name' column
    result = dfData['procurement_id'].apply(lambda x: Find_Similar_Name_MileStone3(x, procurement_choices))

    # Display the DataFrame with closest matches
    #print('result :', result)

    return dfData

def Perform_MileStone3_Operation():
    dfData_MileStone3 =ReadData_MileStone3()

    #print('dfData.head() :',dfData_MileStone3.head())

    # 'Fighter', 'KD', 'Sig._str.', 'Sig._str._%', 'Total_str.', 'Td', 'Td_%', 'Sub._att', 'Rev.', 'Ctrl'

    # replace long column names to shorter ones
    dfData_MileStone3.rename(columns={"Fighter" : Naming_Convention_MileStone3("UFCfighter_Name"),
                            "Sig. str" : Naming_Convention_MileStone3("Sig_str"),
                            "Sig. str. %" : Naming_Convention_MileStone3("Sig_str_percentage"),
                            "Head" : Naming_Convention_MileStone3("Head_Count"),
                            "Body" : Naming_Convention_MileStone3("Body_Count"),
                            "Leg" : Naming_Convention_MileStone3("Leg_Count"),
                            "Distance" : Naming_Convention_MileStone3("Distance_Count"),
                            "Clinch" : Naming_Convention_MileStone3("Clinch_Count"),
                            "Ground" : Naming_Convention_MileStone3("Ground_Count")
                            }, inplace=True)


    # Check if any of the column contains missing value NaN
    #print("dfData.isna().sum(axis=0) :", dfData_MileStone3.isna().sum(axis=0))

    # remove spaces in columns name and replace with underscores
    dfData_MileStone3.columns = dfData_MileStone3.columns.str.replace(' ','_')
    #print("df.columns remove spaces :", dfData_MileStone3.columns)

    # changing all colunmn names to lower case
    dfData_MileStone3.rename(columns=str.lower, inplace=True)
    #print("df.columns lower case :", dfData_MileStone3.columns)

    ### Format data into a more readable format
    dfData_MileStone3 = Format_Data_Readable_Format_MileStone3(dfData_MileStone3)

    ### Identify duplicates
    dfData_MileStone3 = Find_Duplicates_MileStone3(dfData_MileStone3)

    ### Identify outliers and bad data
    ###Identify_Outliers_BadData_MileStone3(dfData_MileStone3)

    ### Fix casing or inconsistent values
    dfData_MileStone3 = Fix_Casing_InconsistentValues_MileStone3(dfData_MileStone3)

    #print('dfData :', dfData_MileStone3)
    ### Conduct Fuzzy Matching
    #Conduct_Fuzzy_Matching_MileStone3(dfData)

    dfData_MileStone3 = dfData_MileStone3.head(10)
    dfData_MileStone3.insert(0, 'customerNo', range(1, 1 + len(dfData_MileStone3)))

    return dfData_MileStone3
```

```
######### MileStone - 4 #################################


with open('APIkeys.json') as f:
        keys = json.load(f)
        data_world_api_token = keys['data-world-token']

serviceurl = 'https://donnees-data.tpsgc-pwgsc.gc.ca/br1/delaipaiement-promptpayment/delaipaiement-promptpaymen

def ReadData_MileStone4():
    # header values to be passed in HTTP POST request
    # fetch data in json format
    headers = {
     "Content-type":"application/json",
     "Accept":"application/json",
     "Authorization":data_world_api_token
    }

    # endpoint URL for fecthing data from
    resp = requests.get(serviceurl,headers=headers)
    dfData = json.loads(resp.text)
    return dfData;

# This function will change the parameter passed to follow naming standard
def Naming_Convention_MileStone4(name):
   char_to_replace = {
       " ":"_",
       "-":"_",
       "(":"_",
       ")":"_"
   }
   # strip method will remove \n and any leading/trailing whitespaces
   better_name = name.strip().lower()
   # Replace string place holders according to values in dictionary
   better_name = better_name.translate(str.maketrans(char_to_replace))
   return better_name.rstrip('_')

# Define a function to find similar names
def Find_Similar_Name_MileStone4(name, choices):
    return process.extractOne(name, choices)

def Format_Data_Readable_Format_MileStone4(dfData):
    #print("dtypes ",dfData.dtypes)
    dfData.insert(0, 'procurement_num', range(880, 880 + len(dfData)))

    #print(dfData)
    # Replace empty string ('') with np.nan before convertion
    dfData['procurement_id']=dfData.procurement_id.replace('',np.nan)
    dfData['project_name']=dfData.project_name.replace('',np.nan)
    dfData['vendor_name']=dfData.vendor_name.replace('',np.nan)
    dfData['payment_date']=dfData.payment_date.replace('',np.nan)
    dfData['proper_invoice_received_date']=dfData.proper_invoice_received_date.replace('',np.nan)

    # Remove any any text starting from parenthisis "("" to end using regex
    dfData['vendor_name'] = dfData.vendor_name.apply(lambda x: re.sub("\(.*", "",x))

    # Removing any leading and trailing spaces and coverting state names to uppercase
    dfData['procurement_id'] = dfData.procurement_id.str.strip()
    dfData['vendor_name'] = dfData.vendor_name.str.strip()
    dfData['payment_date'] = dfData.payment_date.str.strip()
    dfData['proper_invoice_received_date'] = dfData.proper_invoice_received_date.str.strip()
    dfData['procurement_id'].head()

    return dfData

def Find_Duplicates_MileStone4(dfData):

    dfData.head(10)
    # To find duplicates on specific column 'procurement_id'
    #print("dfData.duplicated column 'procurement_id' ", dfData.duplicated(subset=['procurement_id']))
    #print("dfData.duplicated column 'project_name' ", dfData.duplicated(subset=['project_name']))
    #print("dfData.duplicated column 'vendor_name' ", dfData.duplicated(subset=['vendor_name']))
    #print("dfData.duplicated column 'payment_date' ", dfData.duplicated(subset=['payment_date']))
    #print("dfData.duplicated column 'proper_invoice_received_date' ", dfData.duplicated(subset=['proper_invoic

    #print("Procurement Id is duplictated - {}".format(any(dfData.procurement_id.duplicated())))
    #print("Project Name is duplictated - {}".format(any(dfData.project_name.duplicated())))
    #print("Vendor Name Url is duplictated - {}".format(any(dfData.vendor_name.duplicated())))
    #print("Payment Date is duplictated - {}".format(any(dfData.payment_date.duplicated())))
    #print("Proper Invoice Received dDte is duplictated - {}".format(any(dfData.proper_invoice_received_date.du

    # DROP duplicates data
    dfData.drop_duplicates(subset=['procurement_id', 'project_name', 'vendor_name', 'payment_date', 'proper_inv

    # By default it removes duplicate rows based on all columns.
    dfData.drop_duplicates()

    return dfData
```

```python
def Identify_Outliers_BadData_MileStone4(dfData):

    ### Check if any essential column contains NaN.

    print("The column Procurement Id contains NaN - %r " % dfData.procurement_id.isnull().values.any())
    print("The column Project Name contains NaN - %s " % dfData.project_name.isnull().values.any())
    print("The column Vendor Name contains NaN - %s " % dfData.vendor_name.isnull().values.any())
    print("The column Payment Date contains NaN - %s " % dfData.payment_date.isnull().values.any())
    print("The column Proper Invoice Received Date contains NaN - %s " % dfData.proper_invoice_received_date.is

    # Create a box plot to check for outliers.
    plt.boxplot(dfData.procurement_num, notch=True)

    # Find outliers and view the data distribution using a histogram
    fig = px.histogram(dfData, x='procurement_num')
    fig.show()

    # Find multivariate outliers using a scatter plot
    fig_1 = px.scatter(x=dfData['procurement_num'], y=dfData['procurement_num'])
    fig_1.show()

def Fix_Casing_InconsistentValues_MileStone4(dfData):

    # Lower name column. Fix capitalization inconsistencies
    dfData['procurement_id'] = dfData['procurement_id'].str.lower()
    dfData['project_name'] = dfData['project_name'].str.lower()
    dfData['vendor_name'] = dfData['vendor_name'].str.lower()
    dfData['payment_date'] = dfData['payment_date'].str.lower()
    dfData['proper_invoice_received_date'] = dfData['proper_invoice_received_date'].str.lower()

    # Verify changes have been effected
    dfData_name = dfData['procurement_num'].unique()

    # Fix whitespace if any

    dfData_name.sort()
    #print("dfData_name sort : ", dfData_name)

    # Remove white spaces from `dest_size`
    dfData['procurement_id'] = dfData['procurement_id'].str.strip()
    dfData['project_name'] = dfData['project_name'].str.strip()
    dfData['vendor_name'] = dfData['vendor_name'].str.strip()
    dfData['payment_date'] = dfData['payment_date'].str.strip()
    dfData['proper_invoice_received_date'] = dfData['proper_invoice_received_date'].str.strip()
    # Verify changes have been effected
    #print(dfData['procurement_num'].unique())

    return dfData

def Conduct_Fuzzy_Matching_MileStone4(dfData):

    #Use fuzzy matching to correct inconsistent data entry

    # get all the unique values in the 'dest_region' column
    dfData_name = dfData['procurement_num'].unique()
    # sort them alphabetically and then take a closer look
    dfData_name.sort()

    vendorname = 'JES CONSTRUCTION INC.'
    for row in dfData['vendor_name']:
        print("{} {}".format(row, distance(vendorname, row)))

    # Create a list of choices for name matching
    procurement_choices = dfData['procurement_id'].tolist()

    # Apply the function to each row in the 'Name' column
    result = dfData['procurement_id'].apply(lambda x: find_similar_name(x, procurement_choices))

    # Display the DataFrame with closest matches
    #print('result :', result)

    return dfData

def Perform_MileStone4_Operation():
    jsondata = ReadData_MileStone4();
    # Normalize semi-structured JSON data into a flat table.
    df = pd.json_normalize(jsondata['data'])
    dfData_MileStone4 = pd.DataFrame(df)
    #print('dfData.shape :', dfData_MileStone4.shape)
    #print(dfData)

    # First row of the data is for Total of all states. As I am interested in only state level records, let's d
    dfData_MileStone4.drop(axis=0,index=0,inplace=True)
    dfData_MileStone4.shape

     # printing the headers
    names = []
    for line in dfData_MileStone4:
        var=line.split(":")[0]
        names.append(var)

    #print("Printing existing headers :", names);

    # replace long column names to shorter ones
    dfData_MileStone4.rename(columns={"procurement-id_id-approvisionnement" : Naming_Convention_MileStone4("pro
                            "Project-number_Numéro-de-projet" : Naming_Convention_MileStone4("project_name"),
                            "Vendor-name_Nom-du-fournisseur" : Naming_Convention_MileStone4("vendor_name"),
                            "Payment-date_Date-de-paiement" : Naming_Convention_MileStone4("payment_date"),
                            "Proper-Invoice-Received-Date_date-de-réception-de-la-facture-en-règle" : Naming_Co
                            }, inplace=True)


    # Check if any of the column contains missing value NaN
    #print("dfData_MileStone4.isna().sum(axis=0) :", dfData_MileStone4.isna().sum(axis=0))

    # remove spaces in columns name and replace with underscores
    dfData_MileStone4.columns = dfData_MileStone4.columns.str.replace(' ','_')
    #print("df.columns remove spaces :", dfData_MileStone4.columns)

    # changing all colunmn names to lower case
    dfData_MileStone4.rename(columns=str.lower, inplace=True)
    #print("df.columns lower case :", dfData_MileStone4.columns)

    #print(dfData_MileStone4)

    ### Format data into a more readable format
    dfData_MileStone4 = Format_Data_Readable_Format_MileStone4(dfData_MileStone4)

    ### Identify duplicates
    dfData_MileStone4 = Find_Duplicates_MileStone4(dfData_MileStone4)

    ### Identify outliers and bad data
    #Identify_Outliers_BadData(dfData_MileStone4)

    ### Fix casing or inconsistent values
    dfData_MileStone4 = Fix_Casing_InconsistentValues_MileStone4(dfData_MileStone4)

    ### Conduct Fuzzy Matching
    #Conduct_Fuzzy_Matching(dfData_MileStone4)

    dfData_MileStone4 = dfData_MileStone4.head(10)
    dfData_MileStone4.insert(0, 'customerNo', range(1, 1 + len(dfData_MileStone4)))

    dfData_MileStone4.at[5 ,'vendor_name']= 'the state group inc.'
    dfData_MileStone4.at[5 ,'vendor_name']= 'the state group inc.'
    dfData_MileStone4.at[9 ,'vendor_name']= 'the state group inc.'

    return dfData_MileStone4


######### MileStone - 5 #################################

### Merging the Data and Storing in a Database/Visualizing Data
# Now that you have cleaned and transformed your 3 datasets, you need to load them into a database. You can cho
# You will want to load each dataset into SQL Lite as an individual table and then you must join the datasets t
# Once all the data is merged together in your database, create 5 visualizations that demonstrate the data you 
# You should have at least 2 visualizations that have data from more than one source (meaning, if you have 3 ta
# also welcome to use your consolidated dataset that you created in the previous step, if you do that, you have


#conn = sqlite3.connect("justicesystem.db")
# Using in-memory database
conn = sqlite3.connect(':memory:')
# check if connection is successful by creating cursor
def chk_conn(conn):
    try:
      conn.cursor()
      return True
    except Exception as ex:
      return False

def main():
    print('Inside Main function')

    ##########  Operations for MileStone 2 ###################

    dfData_MileStone2 = Perform_MileStone2_Operation()

    ##########  Operations for MileStone 3 ###################

    dfData_MileStone3 = Perform_MileStone3_Operation()

    ##########  Operations for MileStone 4 ###################

    dfData_MileStone4 = Perform_MileStone4_Operation()

    ################ Operations for MileStone 5 ####################
```

```
print('chk_conn(conn) :',chk_conn(conn))

# Storing second dataset dataframe to "dfData_mileStone2" table
dfData_MileStone2.to_sql(name='dfData_mileStone2', con=conn, index=False)
p2 = pd.read_sql('select * from dfData_mileStone2', conn)
#print ('p2.head(5) :', p2.head(5))
display(p2)

# Storing second dataset dataframe to "dfData_mileStone3" table
dfData_MileStone3.to_sql(name='dfData_mileStone3', con=conn, index=False)
p3 = pd.read_sql('select * from dfData_mileStone3', conn)
#print ('p3.head(5) :', p3.head(5))
display(p3)

# Storing first dataset dataframe to "dfData_MileStone4" table
dfData_MileStone4.to_sql(name='dfData_MileStone4', con=conn, index=False)
p4 = pd.read_sql('select * from dfData_MileStone4', conn)
display(p4)
#print('p4.head(5) : ',p4.head(5))


#### [1] First of all, I am interested to check relation between Review_scores_value and Review_per_month.
#### Both fields are in first dataset. So will use only first dataset.

dfData_MileStone2['Review_scores_value'] = dfData_MileStone2['Review_scores_value'].astype(int)
dfData_MileStone2['Reviews_per_month'] = dfData_MileStone2['Reviews_per_month'].astype(int)

ax = sns.lmplot(data=dfData_MileStone2, x="Review_scores_value", y="Reviews_per_month")
ax.set(xlabel='Review scores', ylabel='Reviews per month')
plt.show()

#### [2] Now I want to see how 'Calculated host listings count' and 'Calculated host listings count entire
#### For that purpose, first I will join 2 datasets i.e.
#### 1. Calculated host listings count (File data #2)
#### 2. Calculated host listings count entire homes (File data #2)
#### 3. Vendor Name (API data #4)
#### I will plot side by side in horizontal bar chart.

# Fetching data from 2 tables by SQL query inner join
dfData = pd.read_sql('select mileStone2.customerNo, mileStone2.Calculated_host_listings_count,mileStone2.Ca
# selecting 10 records
dfData.head(10)
#display(dfData)
# Plotting the horizontal bar charts
ind = np.arange(len(dfData))
width = 0.4
fig, ax = plt.subplots()
ax.barh(ind, dfData.Calculated_host_listings_count, width, color='green', label='Calculated host listings c
ax.barh(ind + width, dfData.Calculated_host_listings_count_entire_homes, width, color='orange', label='Calc
ax.set(yticks=ind + width, yticklabels=dfData.vendor_name, ylim=[2*width - 1, len(dfData)])
ax.legend()
plt.show()


#### [3] Now I want to see how Review scores location and Review scores communication are related in differ
#### For that purpose, first I will join 2 datasets i.e.
#### 1. Review scores location (File data #2)
#### 2. Review scores communication (File data #2)
#### 3. Vendor Name (API data #4)
#### I will plot for 10 random states side by side in vertical bar chart.

dfData = pd.read_sql('select mileStone2.customerNo, mileStone4.vendor_name, mileStone2.Review_scores_commun
# selecting 10 records
dfData.head(10)
dfData.sample(10).plot.bar()


#### [4] Now I would like to check Calculated_host_listings_count,  Calculated_host_listings_count_entire_h
#   Calculated_host_listings_count_shared_rooms,    Reviews_per_month in Overall Unites States (combined fo

dfData = pd.read_sql('select mileStone2.customerNo, mileStone4.vendor_name, mileStone2.Calculated_host_list
dfData.dropna()
#display('Dataset for pie chart',dfData)

# Plot the pie chart
# make the plot
# Extract the sizes of the segments
#df_vendors = dfData.groupby(['vendor_name']).size()
#df_vendors.plot.pie(figsize=(4,4))
dfData.groupby(['vendor_name']).sum().plot(kind='pie', y='customerNo')

#### [5] Now I would also like to see any correlation various vendor_name.
#### For this purpose, I will join 2 datasets
#### 1. Listing_URL (Web Data #2)
#### 2. Vendor Name and procurement_id data (API Data #3)

#### Then I will plot HeatMap

dfData = pd.read_sql('select mileStone2.customerNo, mileStone4.vendor_name, mileStone2.Calculated_host_list
    #display('Dataset for heatmap chart',dfData)

    # Plotting HeatMap
    plt.figure(figsize=(8,8))
    sns.heatmap(dfData.corr(), annot = True, fmt='.1g', cmap= 'coolwarm')


if __name__ == "__main__":
    main()
```

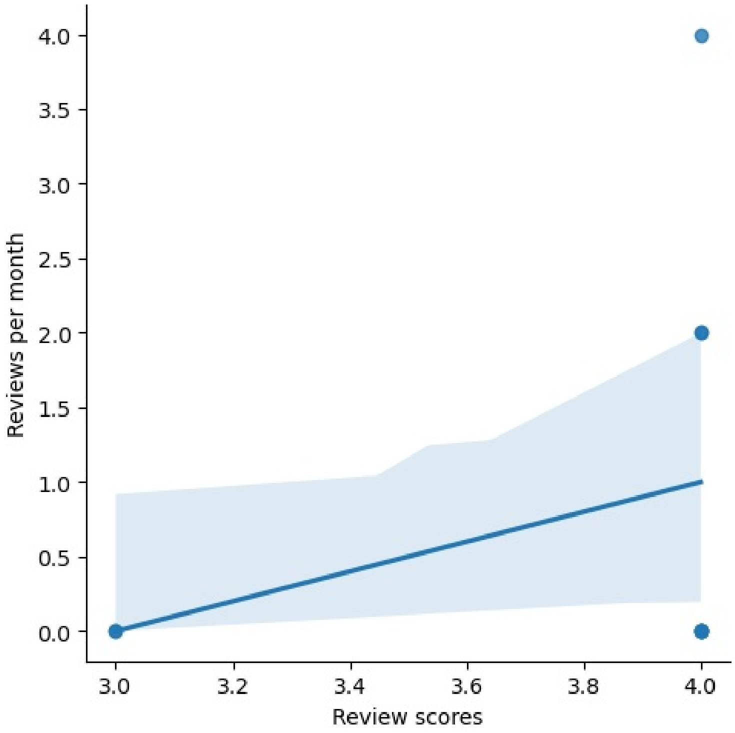
```
Inside Main function
['https://www.airbnb.com/rooms/958' 'https://www.airbnb.com/rooms/5858'
 'https://www.airbnb.com/rooms/8142' ...
 'https://www.airbnb.com/rooms/970412224633812937'
 'https://www.airbnb.com/rooms/970487367150776460'
 'https://www.airbnb.com/rooms/970585764773485914']
['HOLLY' 'PHILIP AND TANIA' 'AARON' ... 'SABA' 'GENNA' 'JAVIER']
chk_conn(conn) : True
```

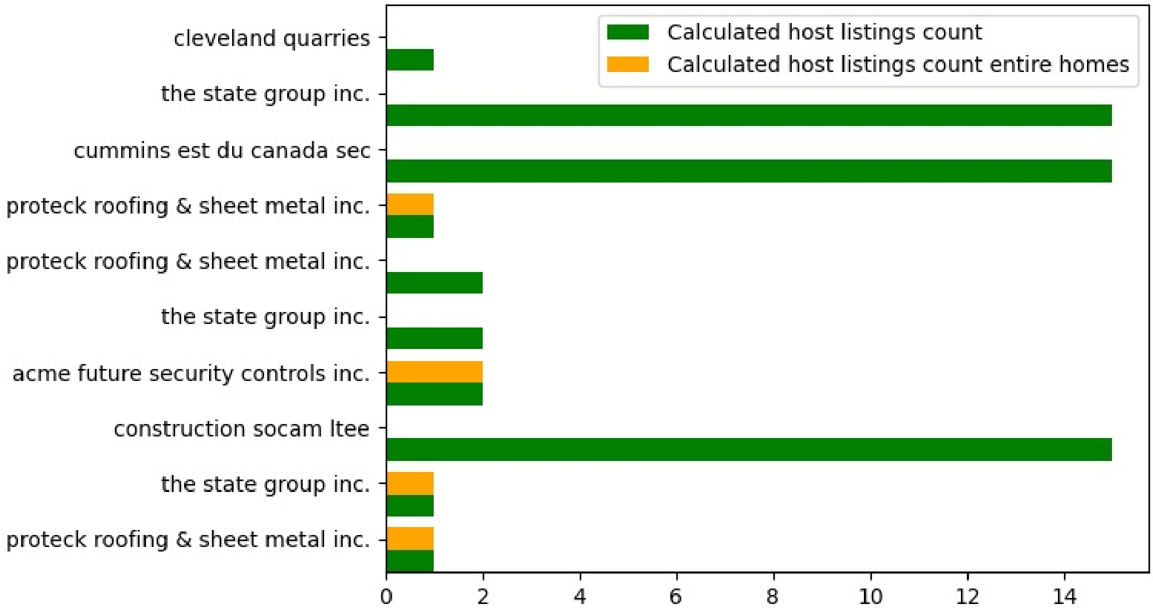
	customerNo	Airbnb_Id	Listing_URL	Scrape_Id	Last_Scraped	Source	Name	Description	Neighborhood_
0	1	$1K	https://www.airbnb.com/rooms/958	$20,230,900,000K	2023-09-02 00:00:00	city scrape	Serviced apartment in San Francisco · ★4.87 · ...	Our bright garden unit overlooks a lovely back...	Quiet cul de sa neighbor
1	2	$6K	https://www.airbnb.com/rooms/5858	$20,230,900,000K	2023-09-02 00:00:00	city scrape	Rental unit in San Francisco · ★4.88 · 2 bedro...	We live in a large Victorian house on a quiet ...	I lo neighborhood
2	3	$8K	https://www.airbnb.com/rooms/8142	$20,230,900,000K	2023-09-02 00:00:00	city scrape	Rental unit in San Francisco · ★4.70 · 1 bedro...	Nice and good public transportation. 7 minute...	N Juda Mu UCSF Shuttl
3	4	$8K	https://www.airbnb.com/rooms/8339	$20,230,900,000K	2023-09-02 00:00:00	city scrape	Condo in San Francisco · ★4.87 · 1 bedroom · 1...	Pls email before booking. <br />Interior featu...	
4	5	$9K	https://www.airbnb.com/rooms/8739	$20,230,900,000K	2023-09-02 00:00:00	city scrape	Condo in San Francisco · ★4.92 · 1 bedroom · 1...	Welcome to "The Mission," the sunniest neighbo...	Located betwee Street and D
5	1	$11K	https://www.airbnb.com/rooms/10537	$20,230,900,000K	2023-09-02 00:00:00	city scrape	Place to stay in San Francisco · ★4.94 · 1 bed...	Casa de Paz (House of Peace) is like staying w...	
6	1	$11K	https://www.airbnb.com/rooms/10578	$20,230,900,000K	2023-09-02 00:00:00	city scrape	Rental unit in San Francisco · ★4.93 · Studio ...	A cute studio with nice street views and lots ...	Very centrally l /><br />Fis
7	8	$12K	https://www.airbnb.com/rooms/12041	$20,230,900,000K	2023-09-02 00:00:00	city scrape	Rental unit in San Francisco · ★4.0 · 1 bedroo...	Nice and good public transportation. 7 minute...	N Juda Mu UCSF Shuttl
8	9	$12K	https://www.airbnb.com/rooms/12042	$20,230,900,000K	2023-09-02 00:00:00	city scrape	Rental unit in San Francisco · ★3.20 · 1 bedro...	Settle down, S.F. resident, student, hospital,...	N Juda Mu UCSF Shuttl
9	10	$13K	https://www.airbnb.com/rooms/12522	$20,230,900,000K	2023-09-02 00:00:00	city scrape	Rental unit in San Francisco · ★4.93 · 1 bedro...	1895 Victorian flat w/ 12 ft ceilings. (No Lon...	

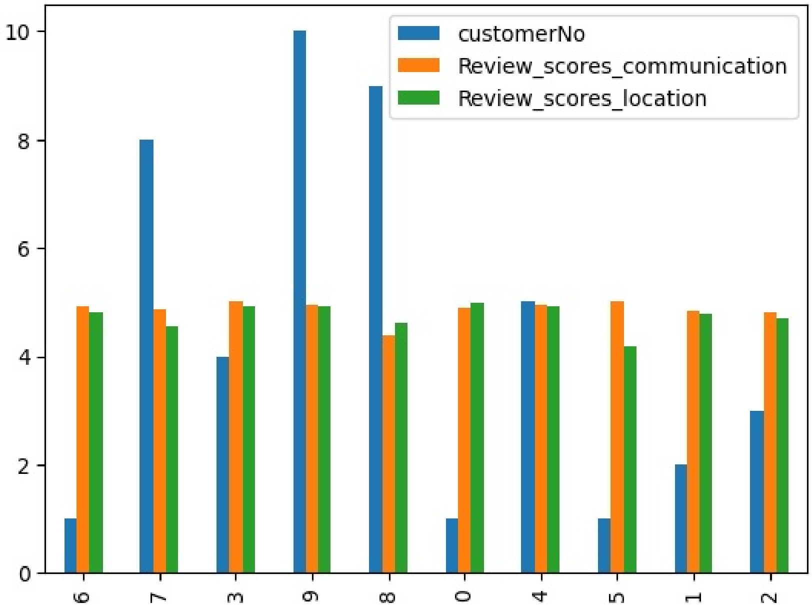
10 rows × 70 columns

	customerNo	ufcfighter_name	sig_str	sig_str_percentage	head_count	body_count	leg_count	distance_count	clinch_count	ground_coun
0	1	joanne wood	27 of 68	39%	8 of 36	3 of 7	16 of 25	26 of 67	1 of 1	0 of
1	2	taila santos	30 of 60	50%	21 of 46	3 of 7	6 of 7	19 of 42	0 of 0	11 of 1

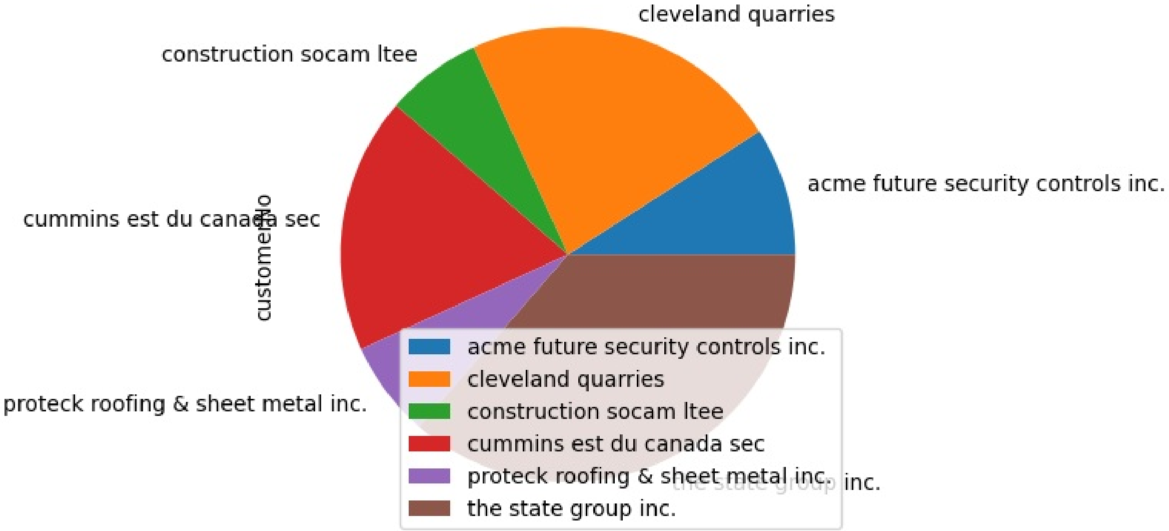
	customerNo	procurement_num	procurement_id	project_name	vendor_name	payment_date	proper_invoice_received_date
0	1	880	eq75470483	r.076458.002	proteck roofing & sheet metal inc.	2017-05-10	2017-03-17
1	2	881	eq75450756	r.012641.001	the state group inc.	2017-05-02	2017-04-06
2	3	882	ef171366	r.079823.001	construction socam ltee	2017-05-02	2017-04-06
3	4	883	ep75651517	r.009790.328	acme future security controls inc.	2017-05-05	2017-04-11
4	5	884	ep75051699	r.029435.327	the state group inc.	2017-05-09	2017-04-13
5	6	885	ej19622337	None	g a l power systems ottawa ltd	2017-05-16	2017-04-20
6	7	886	ee51771572	r.065090.101	jes construction inc.	2017-05-16	2017-04-20
7	8	887	ej19630410	None	cummins est du canada sec	2017-05-19	2017-04-25
8	9	888	ep67171354	r.068270.013	the state group inc.	2017-05-12	2017-05-09
9	10	889	ep75150193	r.011801.219	cleveland quarries	2017-05-12	2017-03-31



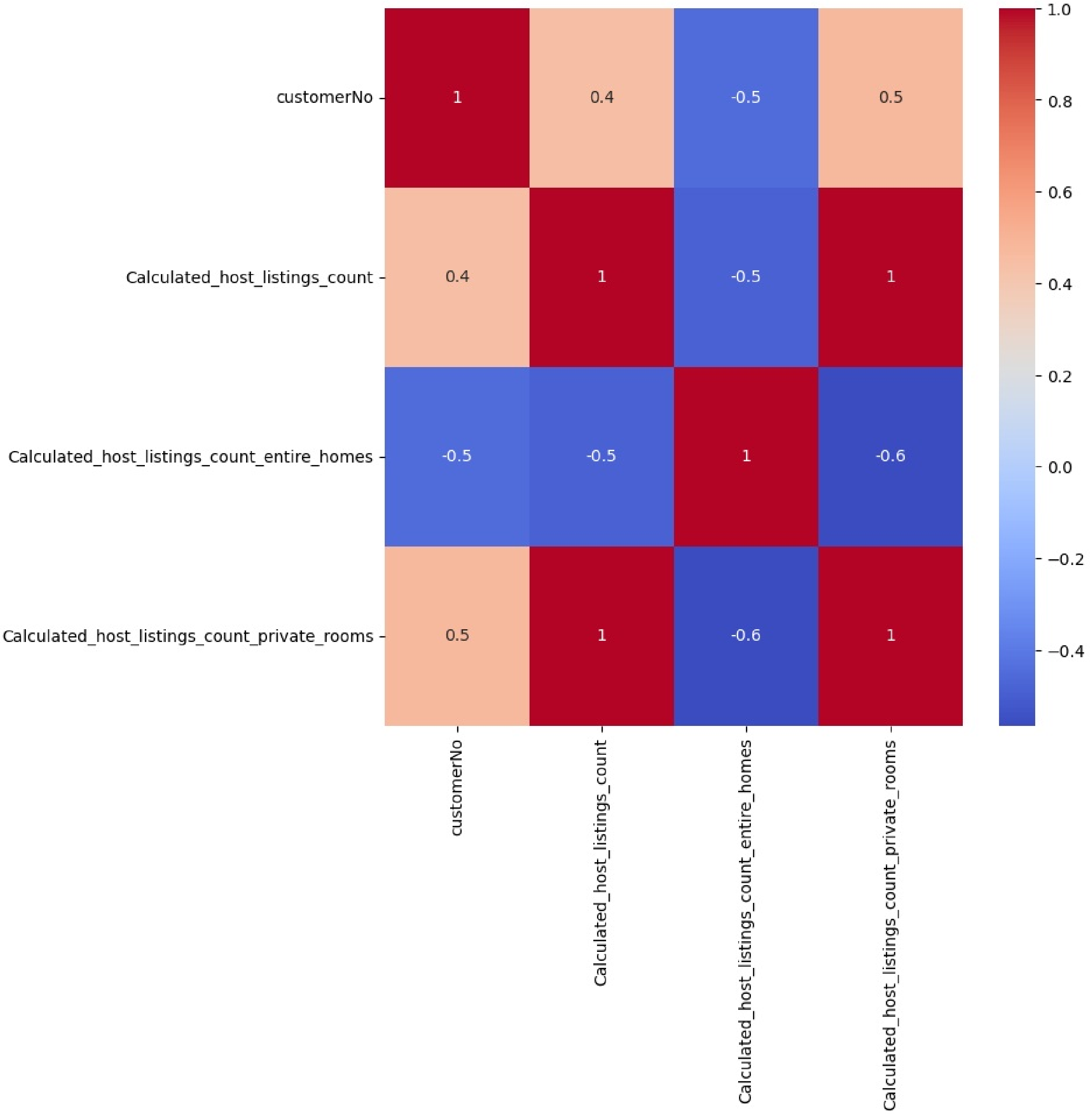




10
8
6
4
2
0
customerNo
Review_scores_communication
Review_scores_location
6
7
3
9
8
0
4
5
1
2



cleveland quarries
construction socam ltee
acme future security controls inc.
cummins est du canada sec
customerNo
proteck roofing & sheet metal inc.
the state group inc.
acme future security controls inc.
cleveland quarries
construction socam ltee
cummins est du canada sec
proteck roofing & sheet metal inc.
the state group inc.



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js