General information:

Author:

Mikko Saukkoriipi 013877851

Date:

14 Dec 2020

Background of the project:

This project is done for the course Distributed Data Infrastructure fall 2020. The course is part of the Masters of Data Science Programme, and the primary purpose of the project was to practice the use of the GraphLab, which was replaced by Turi Create.

General stucture:

This notebook follows the structure of the given questions in the same order as they are. In other words, each question has been answered in its own chapter. At the end of each chapter, there is a text answer to the given research question.

Project questions:

Distributed Data Infrastructures, Fall 2020, Project 2

Assignment In this project, you are supposed to use GraphLab to analyze data sets. GraphLab is a high performance, open-source, big data framework which we have discussed in the course. We provide you with a data set and you should run GraphLab either on AWS or locally to analyze the data set.

The dataset was released by Telecom Italia for their Open Big Data Challenge in 2014. It contains telecommunication records, weather, air quality, electricity consumption for city of Milan and province of Trentino in Italy in November and December 2013. You can find detailed description of the dataset from the paper: https://www.nature.com/articles/sdata201555.

Requirements You need write a program that uses GraphLab to provide answers to the following questions.

First three questions are worth 2 points total.

1. Find the most congested communication period of the day in Milan and Trentino.
2. List top 5 Italian provinces which are most called by residents of Milan and Trentino on average.
3. List top 5 languages tweeted by distinct users in Milan. How popular is Finnish as a tweeting language in Milan?

The following questions are worth 3 points total.

1. Compare call and internet activity between 24th, 25th and 26th December to 26th, 27th, 28th November for Milan. Plot the distribution.
2. Find correlation between user communication activity and different weather conditions (e.g. rain, snow etc.) in Milan and Trentino.

The final questions are worth 2 points each.

1. Plot the heatmap of user telecommunication activity for both Milan and Trentino. Do you observe any shift in communication pattern of users between day and night? (A typical day time is between 8AM to 8PM)
2. Investigate and plot the correlation between air quality and weather (temperature, sunshine, precipitation, etc.).

Read needed libraries before starting to answer the question

In [242]:
```
#Imports
import pandas as pd
from turicreate import SFrame, SGraph, Vertex, Edge
import arrow
import glob2
import time
from datetime import datetime as dt
import turicreate.aggregate as agg
import matplotlib.pyplot as plt
import seaborn as sns
import geopandas as gpd
import descartes
import datetime
```

1. Find the most congested communication period of the day in Milan and Trentino.

Answer to this question is based on the telecommunication data for Milan and Trentino on 24th December 2013. Answer will be the most congested hour in the given day. We will use combination of SMS, call and internet traffic data to find most congested communication period.

I have used a single day in this exercise because I have answered the question on my laptop. The size of the single-day telecom data for one data is the range between 270Mb to 350Mb and using multiple days in the calculation is a heavy and slow process for a laptop. Simultaneously, using multiple days would only change the file reading, but otherwise, code would be same.

Text answer to the question is given at the end of the code.

Telecommunication data has folllowing information:

1. Square id: identification string of a given square of Milan/Trentino GRID;
2. Time Interval: start interval time expressed in milliseconds. The end interval time can be obtained by adding 600,000 milliseconds (10 min) to this value;
3. Country code: the phone country code of the nation.
4. SMS-in activity: activity proportional to the amount of received SMSs inside a given Square id and during a given Time interval. The SMSs are sent from the nation identified by the Country code.
5. SMS-out activity: activity proportional to the amount of sent SMSs inside a given Square id during a given Time interval. The SMSs are received in the nation identified by the Country code;
6. Call-in activity: activity proportional to the amount of received calls inside the Square id during a given Time interval. The calls are issued from the nation identified by the Country code;
7. Call-out activity: activity proportional to the amount of issued calls inside a given Square id during a given Time interval. The calls are received in the nation identified by the Country code;
8. Internet traffic activity: number of CDRs generated inside a given Square id during a given Time interval. The Internet traffic is initiated from the nation identified by the Country code;

Define function to find top n number of peak hours

In [243]:
```
# Define function to find peak hour for the day
def peak_n_hours(sframe, n):
    # Rename columns
    sframe = sframe.rename({'0': 'SquareID', '1':'Time', '2':"country_code", "3":"SMS
    # Add hour column
    sframe['hour'] = sframe['Time'].apply(lambda t: dt.utcfromtimestamp(t/1e3).strfti

    # Calculate hourly activity
    sf_hourly = sframe.groupby(key_column_names = 'hour', operations= {'smsin':agg.SUM
                               'smsout': agg.SUM('SMSout'), 'callin':agg.SUM('call_in'),
                               'callout': agg.SUM('call_out'), 'int_traffic': agg.SUM('i

    # Calculate sum of the activity volumns
    sf_hourly['total'] = sf_hourly['smsin']+sf_hourly['smsout']+sf_hourly['callin']+s

    # Drop other columns except hour and total
    sf = sf_hourly["hour", "total"]

    # Sort sf_hourly and fet 5 largest values
    sf = sf.sort('total', ascending=False)

    # Return n first hours and their total activity
    return sf.head(n)
```

Read file and then use peak_n_hours function to fins most congested hours.

File is initially read with Pandas, because Turi is not able to read single csv file with different length of rows. After file is read with pandas it is converted to Turi SFrame.

In [244]:
```
# Set path to files
milano_dec24_path = 'data/milano_telecom_dec/sms-call-internet-mi-2013-12-14.txt'
trentino_dec24_path = 'data/trento_telecom_dec/sms-call-internet-tn-2013-12-14.txt'

# Read the files. SFrame is not able to handle rows with different lengths,
# so first read data with pandas and then convert to SFrame
milano = pd.read_csv(milano_dec24_path, sep="\t", header=None).fillna(0)
milano = SFrame(milano)
trentino = pd.read_csv(trentino_dec24_path, sep="\t", header=None).fillna(0)
trentino = SFrame(trentino)

# Run function and print n=5 busiest communication hours
print("The most congested communication hours of the 24th Dec 2013 in Milan")
print(peak_n_hours(milano, 5))
print("The most congested communication hours of the 24th Dec 2013 in Trentino")
print(peak_n_hours(trentino, 5))
```

```
The most congested communication hours of the 24th Dec 2013 in Milan
+------+-------------------+
| hour |       total       |
+------+-------------------+
|  16  | 5416257.583677531 |
|  11  | 5371330.078746542 |
|  17  | 5344031.796636861 |
|  10  | 5337831.324568588 |
|  15  | 5284227.558210146 |
+------+-------------------+
[5 rows x 2 columns]

The most congested communication hours of the 24th Dec 2013 in Trentino
+------+--------------------+
| hour |       total        |
+------+--------------------+
|  16  | 1036600.3784612629 |
|  10  | 1029579.4795295168 |
|  17  | 1011900.8027854251 |
|  11  | 1007262.7792211829 |
|  15  | 1006217.5522763213 |
+------+--------------------+
[5 rows x 2 columns]
```

Answer to research question 1:

We can see from the results, that in the 24th December 2013 the most congested communication hour in both cities was from 4pm to 5pm UTC time , which is in Italian time 5pm to 6pm.

2. List top 5 Italian provinces which are most called by residents of Milan and Trentino on average.

I will answer this question based on the "Milan/Trentino to provinces" telecommunication dataset on 24th December 2013. Using only a single day is the same as in exercise 1. The reason is that I used only a laptop for computation, and the fact that using multiple days would not change the code. This code is also convertible to use multiple days dataset by only changing the file reading process.

Milan/trentino to provinces dataset has following columns:

1. Square id: identification string of a given square of Milan/Trentino GRID;
2. Time Interval: Start interval time expressed in milliseconds. The end interval time can be obtained by adding 600,000 milliseconds (10 min) to this value;
3. Square to Province Inter: Value representing the interaction between the Square id and the Province. It is proportional to the number of calls exchanged between callers, which are located in the Square id, and receivers located in the Province;
4. Province to Square Inter: Value representing the interaction between the Square id and the Province. It is proportional to the number of calls exchanged between callers, which are located in the Province, and receivers located in the Square id.
5. Province: the name of the Italian province.

Define function to find most called n number of provinces.

Used information is defined in the column "square to province".

In [245]:
```
def most_called_provinces(sframe, n):
    # Rename columns
    sframe = sframe.rename({'0': 'SquareID', '1':'Province', '2':"Time", "3":"SqrtToPr

    # Drop columns where ProvinceToSqrt=0. These rows are incoming calls from province
    # and we want only count calls to provinces.
    sframe = sframe[sframe["SqrtToProvince"] > 0]

    # Count by provinces
    famous_prov = sframe.groupby(key_column_names = 'Province', operations= {'count':

    # Sort by count and return top5 provinces
    return famous_prov.sort('count', ascending=False).head(n)
```

Read files and use most_called_provinces to find most called provinces.

As in the question 1, we will first read file to Pandas, because Turi is not able to handle different length of the rows. After file is succesfully read to Pandas DataFrame, we will convert it to Turi SFrame.

In [246]:
```
# Measure how long it takes
start = time.time()

# Set path to files
milano_dec24_path = 'data/milano_provinces_telecom/mi-to-provinces-2013-12-24.txt'
trentino_dec24_path = 'data/trento_provinces_dec/tn-to-provinces-2013-12-24.txt'

# Read the files. SFrame is not able to handle rows with different lengths,
# so first read data with pandas and then convert to SFrame
milano = pd.read_csv(milano_dec24_path, sep="\t", header=None).fillna(0)
milano = SFrame(milano)
trentino = pd.read_csv(trentino_dec24_path, sep="\t", header=None).fillna(0)
trentino = SFrame(trentino)

# Run function and print top 5 provinces most called by residents
start = time.time()
print("The most called provinces from Milano on the 24th Dec 2013")
print(most_called_provinces(milano, 5))
print("The most called provinces from Trentino on the 24th Dec 2013")
print(most_called_provinces(trentino, 5))

# Print time how long it took
end = time.time()
print("Time to run was", round(end - start), "seconds")
```

```
The most called provinces from Milano on the 24th Dec 2013
+-----------------------+--------+
|        Province       | count  |
+-----------------------+--------+
|         MILANO        | 970483 |
| MONZA E DELLA BRIANZA | 217406 |
|         PAVIA         | 175663 |
|         VARESE        | 85739  |
|         NAPOLI        | 84398  |
+-----------------------+--------+
[5 rows x 2 columns]

The most called provinces from Trentino on the 24th Dec 2013
+---------------+--------+
|    Province   | count  |
+---------------+--------+
|     TRENTO    | 527030 |
| BOLZANO/BOZEN | 107719 |
|    BRESCIA    | 80951  |
|     VERONA    | 59680  |
|     MILANO    | 54876  |
+---------------+--------+
[5 rows x 2 columns]

Time to run was 2 seconds
```

Answer to research question 2:

We can see from the results that most of the calls happen inside the province. After this, there seem to be large neighboring provinces

3. List top 5 languages tweeted by distinct users in Milan. How popular is Finnish as a tweeting language in Milan?

To answer this question, I will use SocialPulse dataset. This is a Twitter dataset with all the needed information.

The SocialPulse dataset contains geolocalized tweets originated from Milan between November 1, 2013 and January 1st, 2014.

1. user: anonymized Twitter username;
2. language: language of the Tweet, where und means undefined;
3. municipality: the municipality in which the tweet has been probably created.
4. timestamp: Tweet timestamp;
5. geometry latitude: approximate position of the tweet, in geoJSON format. Error <600 m.
6. geometry longitude: approximate position of the tweet, in geoJSON format. Error <600 m.

Read the file, filter by municipality and then group and count by language

In [247]:
```
# Set path to file
milano_path = 'data/social_pulse/social_pulse_milano.csv'

# Read the files
milano = SFrame.read_csv(milano_path, sep="\t", header=None)

# Rename columns
milano = milano.rename({'X1': 'user', 'X2':'language', 'X3':"municipality", "X4":"time

# Print unique values in municipality column
#print(list(milano["municipality"].unique()))

# Filter municiplatilties column to include only Milano
milano = milano[milano["municipality"] == "Milano"]

# Count language
languages_count = milano.groupby(key_column_names = 'language', operations= {'count':

# Sort languages by count
languages_count = languages_count.sort('count', ascending=False)

# Print 15 most populat tweeting languages
languages_count.print_rows(num_rows=15)
```

```
Finished parsing file /Users/saukk/Desktop/Distributed data infrastucture/GraphLab project/data/social_pulse/social_pulse_milano.csv
Parsing completed. Parsed 100 lines in 0.166448 secs.
------------------------------------------------------
Inferred types from first 100 line(s) of file as
column_type_hints=[str,str,str,int,float,float]
If parsing fails due to incorrect types, you can correct
the inferred type list above and pass it to read_csv in
the column_type_hints argument
------------------------------------------------------
Finished parsing file /Users/saukk/Desktop/Distributed data infrastucture/GraphLab project/data/social_pulse/social_pulse_milano.csv
Parsing completed. Parsed 269290 lines in 0.177033 secs.
+----------+--------+
| language | count  |
+----------+--------+
|    it    | 146546 |
|    en    | 42958  |
|    es    |  6971  |
|    tl    |  6502  |
|    pt    |  4611  |
|    so    |  3650  |
|    fr    |  3457  |
|    id    |  2949  |
|   und    |  2732  |
|    tr    |  2005  |
|    de    |  1962  |
|    ar    |  1800  |
|    ro    |  1540  |
|    nl    |  1510  |
|    fi    |  1467  |
+----------+--------+
[45 rows x 2 columns]
```

Answer to research question 3:

From the table, we can see that the Italian language is the most common in the municipality of Milan. The second most popular language is English, that is vastly more popular than the rest of the languages. Finnish is in the list in place 15.

4. Compare call and internet activity between 24th, 25th and 26th December to 26th, 27th, 28th November for Milan. Plot the distribution.

To answer this question, I will calculate hourly call and internet activity for each day, and then make histogram and plot of hourly activities in given days in November and December.

Telecommunication data has folllowing information:

1. Square id: identification string of a given square of Milan/Trentino GRID;
2. Time Interval: start interval time expressed in milliseconds. The end interval time can be obtained by adding 600,000 milliseconds (10 min) to this value;
3. Country code: the phone country code of the nation.
4. SMS-in activity: activity proportional to the amount of received SMSs inside a given Square id and during a given Time interval. The SMSs are sent from the nation identified by the Country code.
5. SMS-out activity: activity proportional to the amount of sent SMSs inside a given Square id during a given Time interval. The SMSs are received in the nation identified by the Country code;
6. Call-in activity: activity proportional to the amount of received calls inside the Square id during a given Time interval. The calls are issued from the nation identified by the Country code;
7. Call-out activity: activity proportional to the amount of issued calls inside a given Square id during a given Time interval. The calls are received in the nation identified by the Country code;
8. Internet traffic activity: number of CDRs generated inside a given Square id during a given Time interval. The Internet traffic is initiated from the nation identified by the Country code;

Define function to read multiple files in the same folder in given date interval

In [248]:
```
def create_hourly_telecom_SFrame(paths, start_date, end_date):
    # Measure time how long it takes. Set start time.
    start = time.time()

    # Filter paths by start_date and end_date
    # Path example: sms-call-internet-mi-2013-12-01.txt
    wanted_paths = []
    for path in paths:
        date = int(path[-6:-4])
        if (date <= end_date) and (date >= start_date):
            wanted_paths.append(path)

    # Read the files and append dataframes to result df
    result_df = pd.DataFrame(columns = ["day", "SquareID", "Time", "country_code", "SM
    for path in wanted_paths:
        day = path[-6:-4]
        df = pd.read_csv(path, sep="\t", header=None)
        df = df.fillna(0)
        df.columns = ["SquareID", "Time", "country_code", "SMSin", "SMSout", "call_in
        df["day"] = day
        result_df= result_df.append(df)

    # Convert Pandas dataframe to SFrame
    sf = SFrame(data=result_df)

    # Drop columns SquareID and country_code
    sf = sf.remove_column('SquareID')
    sf = sf.remove_column('country_code')

    # Add hour column and add 1 hour to time (from UTC to Rome time).
    sf["hour"] = sf["Time"].apply(lambda timestamp: arrow.get(timestamp+3600000).forma

    # Group by day and hour, and sum of each activity in each hour and day
    sf_hourly = sf.groupby(['day', 'hour'], {'callin':agg.SUM('call_in'),
                                             'callout': agg.SUM('call_out'),
                                             'int_traffic': agg.SUM('internet_traffic

    # Calculate sum of callin and callout
    sf_hourly["calls"] = sf_hourly["callin"]+sf_hourly["callout"]

    # Drop columns SquareID and country_code
    sf_hourly = sf_hourly.remove_column('callin')
    sf_hourly = sf_hourly.remove_column('callout')

    # Sort by day and hour
    sf_hourly = sf_hourly.sort(['day', 'hour'], ascending=True)

    # Set hour and day type to int
    sf_hourly["day"] = sf_hourly["day"].astype(int)
    sf_hourly["hour"] = sf_hourly["hour"].astype(int)

    # Print time how long it took
    end = time.time()
    print("Time to run function was", round(end - start), "seconds")

    # Return results
    return sf_hourly
```

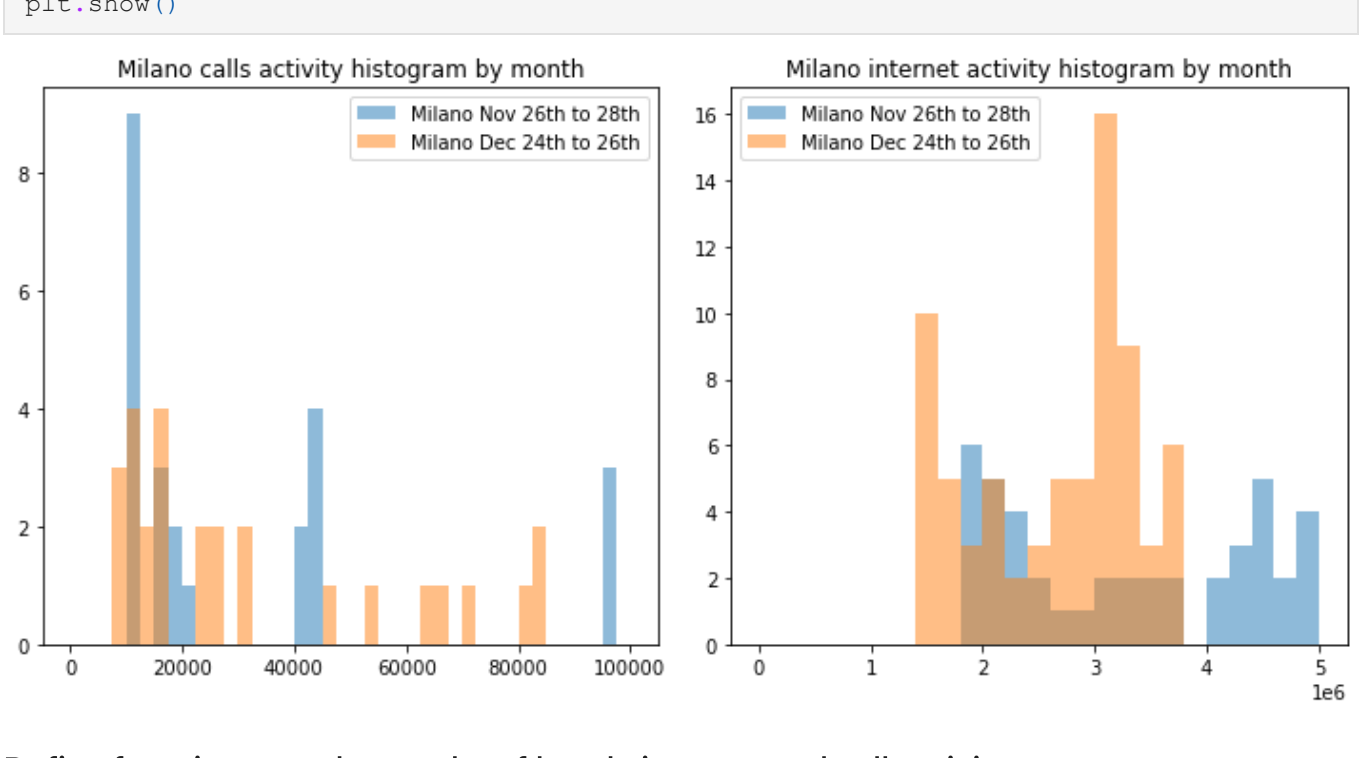
Read the files using ceate hourly telecom SFrame with given days

In [249]:
```
# Execute the glob function that returns a list of filepaths
data_paths_milano_nov = glob2.glob("data/milano_telecom_nov/*")
data_paths_milano_dec = glob2.glob("data/milano_telecom_dec/*")

# Use create_telecom_SFrame to read paths and join the files within given dates
milano_nov = create_hourly_telecom_SFrame(data_paths_milano_nov, 26, 28)
milano_dec = create_hourly_telecom_SFrame(data_paths_milano_dec, 24, 26)

# Print first 5 rows
milano_nov.print_rows(num_rows=5)
milano_dec.print_rows(num_rows=5)
```

```
Time to run function was 88 seconds
Time to run function was 76 seconds
+-----+------+--------------------+--------------------+
| day | hour |    int_traffic     |       calls        |
+-----+------+--------------------+--------------------+
|  26 |  0   | 3081805.3777902434 | 41192.73183245973  |
|  26 |  1   | 2572701.778685294  | 17706.488070584164 |
|  26 |  2   | 2231849.13408716   | 11778.291443263028 |
|  26 |  3   | 2032399.1755711331 | 10178.701211725669 |
|  26 |  4   | 1922521.8709400047 | 11383.318199818374 |
+-----+------+--------------------+--------------------+
[72 rows x 4 columns]

+-----+------+--------------------+--------------------+
| day | hour |    int_traffic     |       calls        |
+-----+------+--------------------+--------------------+
|  24 |  0   | 2479071.569021002  | 54213.38695708153  |
|  24 |  1   | 2100878.8674795628 | 26984.249997574632 |
|  24 |  2   | 1831355.1988663676 | 17032.582469058943 |
|  24 |  3   | 1680874.7424091802 | 11930.500057642956 |
|  24 |  4   | 1549273.9723427454 | 12418.661768335747 |
+-----+------+--------------------+--------------------+
[72 rows x 4 columns]
```

Make call and internet activity histograms of both months to see the distribution

In [250]:
```
fig, ax = plt.subplots(1, 2, tight_layout=True, figsize=(10, 5))

plt.subplot(1, 2, 1)
bins = list(range(0, 100000 + 2500, 2500))
plt.rcParams["figure.figsize"] = (6, 6)
plt.hist(milano_nov["calls"], bins=bins, alpha=0.5)
plt.hist(milano_dec["calls"], bins=bins, alpha=0.5)
plt.legend(['Milano Nov 26th to 28th', "Milano Dec 24th to 26th"], loc ="upper right")
plt.title("Milano calls activity histogram by month")

plt.subplot(1, 2, 2)
bins = list(range(0, 5000000 + 200000, 200000))
plt.rcParams["figure.figsize"] = (6, 6)
plt.hist(milano_nov["int_traffic"], bins=bins, alpha=0.5)
plt.hist(milano_dec["int_traffic"], bins=bins, alpha=0.5)
plt.legend(['Milano Nov 26th to 28th', "Milano Dec 24th to 26th"], loc ="upper left")
plt.title("Milano internet activity histogram by month")
plt.show()
```

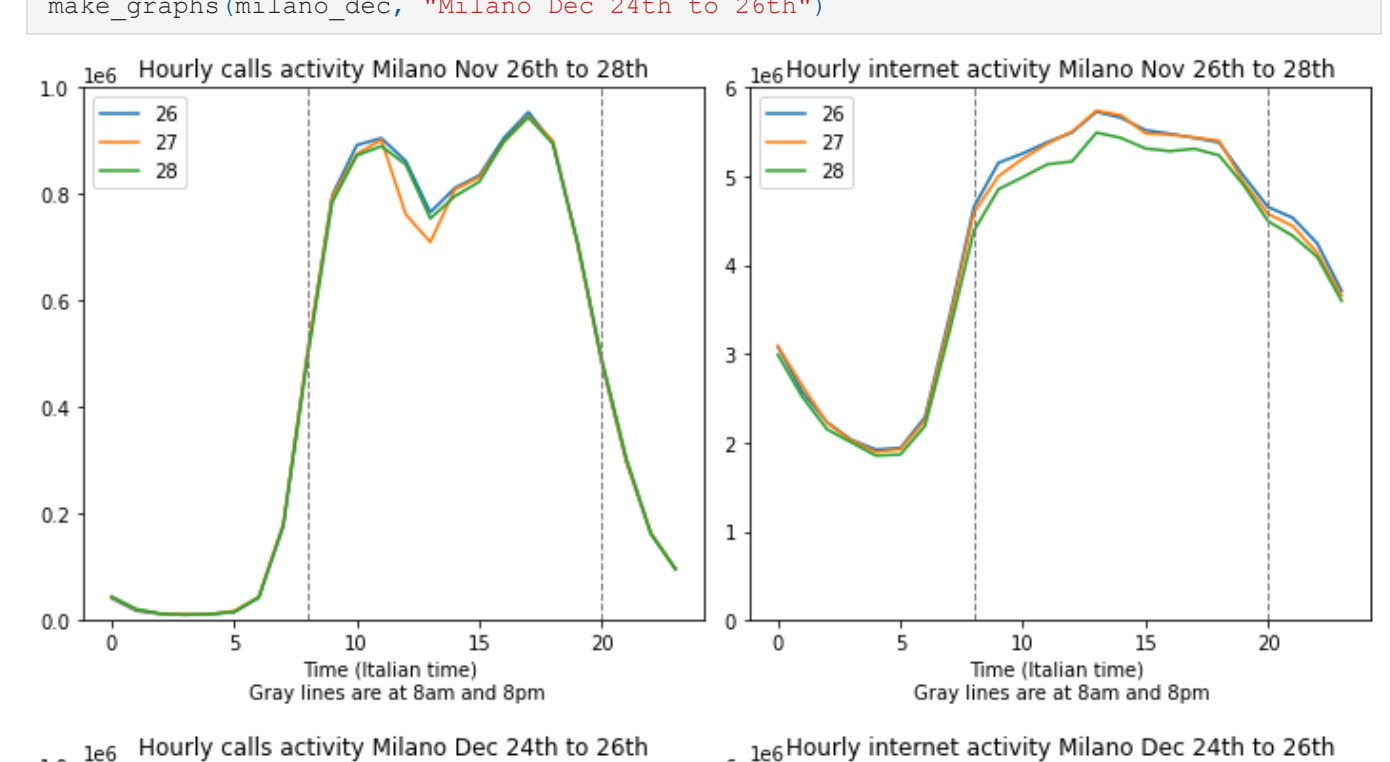


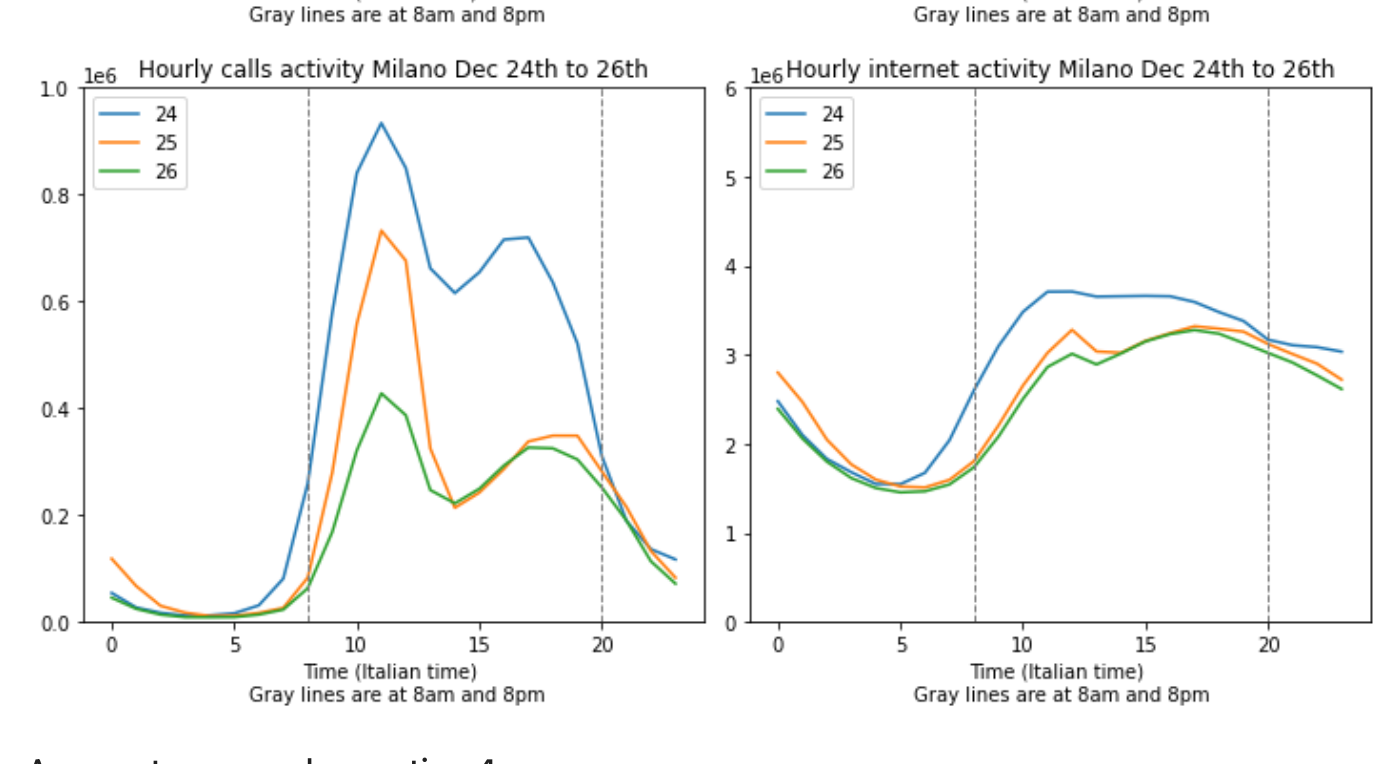
Define function to make graphs of hourly internet and call activity

In [251]:
```
# Make histograms
def make_graphs(SFrame, title):
    # Make days list and sort it
    days = list(SFrame["day"].unique())
    days = sorted(days)

    # Make Trentino histograms. One for each precipitation intensity class.
    fig, ax = plt.subplots(1, 2, tight_layout=True, figsize=(10, 5))

    # Graph1
    plt.subplot(1, 2, 1)
    # Make hourly calls graph. Add vertical lines to 8am and 8pm
    for day in days:
        sf_day = SFrame[SFrame["day"] == day]
        plt.plot(sf_day["hour"], sf_day["calls"])
    plt.ylim(0, 1000000)
    plt.xlabel('Time (Italian time) \n Gray lines are at 8am and 8pm')
    plt.axvline(x=8, color='gray', linestyle='--', lw=1)
    plt.axvline(x=20, color='gray', linestyle='--', lw=1)
    plt.title('Hourly calls activity {}'.format(title))
    plt.legend([days[0], days[1], days[2]], loc ="upper left")
    #plt.show()

    # Graph2
    plt.subplot(1, 2, 2)
    # Make hourly internet activity graph
    for day in days:
        sf_day = SFrame[SFrame["day"] == day]
        plt.plot(sf_day["hour"], sf_day["int_traffic"])
    plt.ylim(0, 6000000)
    plt.xlabel('Time (Italian time) \n Gray lines are at 8am and 8pm')
    plt.axvline(x=8, color='gray', linestyle='--', lw=1)
    plt.axvline(x=20, color='gray', linestyle='--', lw=1)
    plt.title('Hourly internet activity {}'.format(title))
    plt.legend([days[0], days[1], days[2]], loc ="upper left")
    plt.show()
```

Run the make graphs function

In [252]:
```
make_graphs(milano_nov, "Milano Nov 26th to 28th")
make_graphs(milano_dec, "Milano Dec 24th to 26th")
```





Answer to research question 4:

From the results and graphs, we can see a significant difference in calls and internet activity between November 26th to 28th and December 24th to 26th. We can see from the histogram that there are more hectic hours in both classes in November. When we compare the hourly activity graphs, we can see how call and internet activity is lower during Christmas than in November. This can be considered to be expected result, because most of the people are on holiday from the 24th to December 26th.

5. Find correlation between user communication activity and different weather conditions (e.g. rain, snow etc.) in Milan and Trentino.

As precipitation intensity is the only common information we have from the Milano and Trentino, we will use that. Both datasets use a different number of precipitation intensity groups, and they are defined in different scales. For this reason, we will study the correlation between precipitation and communication activity separately in Milan and Trentino.

Because this data is from 2013, the internet was not an important communication channel. Because of this reason, we will calculate communication activity with the following formula:

total_activity = sms_in + sms_out + call_in + call_out

The days we will use are same as in part 4: November 26th to 28th and December 24th to 26th. Time will be in 10 minutes intervals.

Milano weather dataset columns:

1. Timestamp: timestamp value with the following format: YYYYMMDDHHmm;
2. Square id: id of a given square of Milan/Trentino GRID;
3. Intensity: intensity value of the precipitation. It is a value between 0 and 3;
4. Coverage: percentage value of the quadrant covered by the precipitation;
5. Type: type of the precipitation. It is a value between 0 and 2.

Milano intensity classes are defined as follow:

1. Absent: precipitation quantity equal to 0 mm/h. Defined as type 0;
2. Slight: precipitation quantity equal in [0,2] mm/h. Defined as type 1;
3. Moderate: precipitation quantity equal in [2,10] mm/h. Defined as type 2;
4. Heavy: precipitation quantity equal to in [10,100] mm/h. Defined as type 3.

Trentino weather dataset columns:

1. Timestamp: timestamp value with the following format: YYYYMMDDHHmm;
2. Square id: id of a given square of Milan/Trentino GRID;
3. Intensity: intensity value of the precipitation. It is a value between 0 and 18.

Trentino precipitation intensity values means following:

1. very slight: precipitation intensity defined [1,3] meaning an amount of [0.20,2.0] mm/hr.
2. slight: precipitation intensity defined [4,6] meaning an amount of [2.0,7.0] mm/hr.
3. moderate: precipitation intensity defined [7,9] meaning an amount of [7.0,16.0] mm/hr.
4. heavy: precipitation intensity defined [10,12] meaning an amount of [16.0,30.0] mm/hr.
5. very heavy: precipitation intensity defined [13,15] meaning an amount of [30.0,70.0] mm/hr.
6. extreme: precipitation intensity defined [16,18] meaning an amount of more than 70 mm/hr.

First define function to read multiple telecom files and join them to singe SFrame

We will create a new telecom joining function. We did a similar function in part 4, but this time, we also need the timestamps and square ID's to join telecom data to weather data. Also, because the weather can change multiple times in an hour, we don't want to use hourly data, but instead, original data in 10 minutes interval.

In the function user can define the date interval he want to use.

In [253]:
```
# Define function to read multiple telecom files and append them to one SFrame
def create_telecom_SFrame(paths, start_date, end_date):
    # Measure time how long it takes. Set start time.
    start = time.time()

    # Filter paths by start_date and end_date
    # Path example: sms-call-internet-mi-2013-12-01.txt
    wanted_paths = []
    for path in paths:
        date = int(path[-6:-4])
        if (date <= end_date) and (date >= start_date):
            wanted_paths.append(path)

    # Read the files and append dataframes to result df
    result_df = pd.DataFrame(columns = ["time", "SquareID", "total"])
    for path in wanted_paths:
        df = pd.read_csv(path, sep="\t", header=None)
        df = df.fillna(0)
        df.columns = ["SquareID", "time", "country_code", "SMSin", "SMSout", "call_in
        df["total"] = df["SMSin"]+df["SMSout"]+df['call_in']+df['call_out']
        df = df.drop(columns=["SMSin", "SMSout", 'call_in', 'call_out', "inte
        result_df = result_df.append(df)

    # Convert Pandas dataframe to SFrame
    sf = SFrame(data=result_df)

    # Change timestamp type to same format as with weather data. Drop old time colum
    sf["timestamp"] = sf["time"].apply(lambda timestamp: arrow.get(timestamp).format(
    sf = sf.remove_column('time')

    # Print time how long it took
    end = time.time()
    print("Time to run function was", round(end - start), "seconds")

    # Return results
    return sf
```

Use create_telecom_SFrame to read multiple files, then join monthly files, one to Milano and one to Trentino

In [254]:
```
# Set paths to telecom files
data_paths_milano_nov = glob2.glob("data/milano_telecom_nov/*")
data_paths_milano_dec = glob2.glob("data/milano_telecom_dec/*")
data_paths_trentino_nov = glob2.glob("data/trentino_telecom_nov/*")
data_paths_trentino_dec = glob2.glob("data/trento_telecom_dec/*")

# Use create_telecom_SFrame to read paths and join the files within given dates
milano_tele_nov = create_telecom_SFrame(data_paths_milano_nov, 24, 28)
milano_tele_dec = create_telecom_SFrame(data_paths_milano_dec, 24, 26)
trentino_tele_nov = create_telecom_SFrame(data_paths_trentino_nov, 24, 26)
trentino_tele_dec = create_telecom_SFrame(data_paths_trentino_dec, 24, 26)

# Append Milano_tele_nov and Milano_tele_dec
# Do same to trentino telecom data
milano_tele = milano_tele_nov.append(milano_tele_dec)
trentino_tele = trentino_tele_nov.append(trentino_tele_dec)

# Print 5 first rows
milano_tele.print_rows(num_rows=5)
trentino_tele.print_rows(num_rows=5)
```

```
Time to run function was 16 seconds
Time to run function was 14 seconds
Time to run function was 3 seconds
Time to run function was 31 seconds
+----------+----------------------+--------------+
| SquareID |        total         |  timestamp   |
+----------+----------------------+--------------+
|    1     | 0.05701025935414147  | 201311252300 |
|    1     | 0.5391165221186242   | 201311252300 |
|    1     | 0.0035746201209887   | 201311252300 |
|    1     | 0.5444784935531237   | 201311252310 |
|    1     | 0.4054790801879873   | 201311252300 |
+----------+----------------------+--------------+
[30497069 rows x 3 columns]

+----------+----------------------+--------------+
| SquareID |        total         |  timestamp   |
+----------+----------------------+--------------+
|  10000   | 0.5710468694019556   | 201312272300 |
|  10000   | 0.5719633713248610   | 201312272310 |
|  10000   | 0.9560052219963053   | 201312272320 |
|  10000   | 0.04166027556299599  | 201312272330 |
|  10000   | 0.14472894701995652  | 201312272350 |
+----------+----------------------+--------------+
[36823804 rows x 3 columns]
```

Read weather data and join it to telecom data

In [255]:
```
# Set paths to weather files
milano_weather_path = 'data/MeteoMilano_01-11-13_01-01-14.csv'
trentino_weather_path = 'data/precipitation-trentino.csv'

# Read weather files
milano_weather = SFrame.read_csv(milano_weather_path, sep=",", header=None)
trentino_weather = SFrame.read_csv(trentino_weather_path, sep=",", header=None)

# Rename columns of the weather SFrames
milano_weather = milano_weather.rename({'X1':'timestamp', 'X2':'SquareID', 'X3':'inte
trentino_weather = trentino_weather.rename({'X1': 'timestamp', 'X2':'SquareID', 'X3':

# Drop coverage and type from Milano weather data, because we will only study
# how intensity affects to the user communication activity.
milano_weather = milano_weather.remove_column('coverage')
milano_weather = milano_weather.remove_column('type')

# Print first rows
```

```
milano_weather.print_rows(num_rows=5)
trentino_weather.print_rows(num_rows=5)
```

```
Finished parsing file /Users/saukk/Desktop/Distributed data infrastucture/GraphLab project/data/MeteoMilano_01-11-13_01-01-14.csv
Parsing completed. Parsed 100 lines in 0.011216 secs.
------------------------------------------------------
Inferred types from first 100 line(s) of file as
column_type_hints=[int,int,int]
If parsing fails due to incorrect types, you can correct
the inferred type list above and pass it to read_csv in
the column_type_hints argument
------------------------------------------------------
Finished parsing file /Users/saukk/Desktop/Distributed data infrastucture/GraphLab project/data/MeteoMilano_01-11-13_01-01-14.csv
Parsing completed. Parsed 33316 lines in 0.018 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastucture/GraphLab project/data/precipitation-trentino.csv
Parsing completed. Parsed 100 lines in 0.314223 secs.
------------------------------------------------------
Inferred types from first 100 line(s) of file as
column_type_hints=[int,int,int]
If parsing fails due to incorrect types, you can correct
the inferred type list above and pass it to read_csv in
the column_type_hints argument
------------------------------------------------------
Read 2489903 lines. Lines per second: 4.98608e+06
Finished parsing file /Users/saukk/Desktop/Distributed data infrastucture/GraphLab project/data/precipitation-trentino.csv
Parsing completed. Parsed 3489417 lines in 0.583172 secs.
+--------------+----------+-----------+
|  timestamp   | SquareID | intensity |
+--------------+----------+-----------+
| 201311010000 |    1     |     0     |
| 201311010000 |    2     |     0     |
| 201311010000 |    3     |     0     |
| 201311010000 |    4     |     0     |
| 201311010010 |    1     |     0     |
+--------------+----------+-----------+
[33316 rows x 3 columns]

+--------------+----------+-----------+
|  timestamp   | SquareID | intensity |
+--------------+----------+-----------+
| 201311010000 |   2383   |     1     |
| 201311010020 |   9452   |     1     |
| 201311010020 |   8278   |     0     |
| 201311010020 |   6871   |     2     |
| 201311010020 |   5689   |     3     |
+--------------+----------+-----------+
[3489417 rows x 3 columns]
```

Join telecom and weather data

```
# First set SFrames column timestamp and SquareID to int
milano_tele["timestamp"] = milano_tele["timestamp"].astype(int)
milano_tele["SquareID"] = milano_tele["SquareID"].astype(int)
milano_weather["timestamp"] = milano_weather["timestamp"].astype(int)
milano_weather["SquareID"] = milano_weather["SquareID"].astype(int)

trentino_tele["timestamp"] = trentino_tele["timestamp"].astype(int)
trentino_tele["SquareID"] = trentino_tele["SquareID"].astype(int)
trentino_weather["timestamp"] = trentino_weather["timestamp"].astype(int)
trentino_weather["SquareID"] = trentino_weather["SquareID"].astype(int)

# Join weather and telecom datasets
milano = milano_weather.join(milano_tele, on=['timestamp', 'SquareID'], how='inner')
trentino = trentino_weather.join(trentino_tele, on=['timestamp', 'SquareID'], how='in...

# Print first rows
milano.print_rows(num_rows=5)
trentino.print_rows(num_rows=5)
```

```
+--------------+----------+-----------+----------------------+
|  timestamp   | SquareID | intensity |        total         |
+--------------+----------+-----------+----------------------+
| 201311252300 |    1     |     1     | 0.05701250935247165  |
| 201311252300 |    1     |     0     |  0.5391185522366262  |
| 201311252310 |    1     |     1     | 0.003574622010998775 |
| 201311252310 |    1     |     0     |  0.5444784825551247  |
| 201311252320 |    1     |     0     |  0.4054790801987945  |
+--------------+----------+-----------+----------------------+
[10023 rows x 4 columns]

+--------------+----------+-----------+---------------------+
|  timestamp   | SquareID | intensity |        total        |
+--------------+----------+-----------+---------------------+
| 201312281010 |  10043   |     1     |         0.0         |
| 201312281530 |  10095   |     1     | 0.41318456643054676 |
| 201312281530 |  10095   |     1     |  5.493458678918411  |
| 201312281530 |  10095   |     1     |         0.0         |
| 201312281640 |  10100   |     1     |  1.4242910096578831 |
+--------------+----------+-----------+---------------------+
[358166 rows x 4 columns]
```

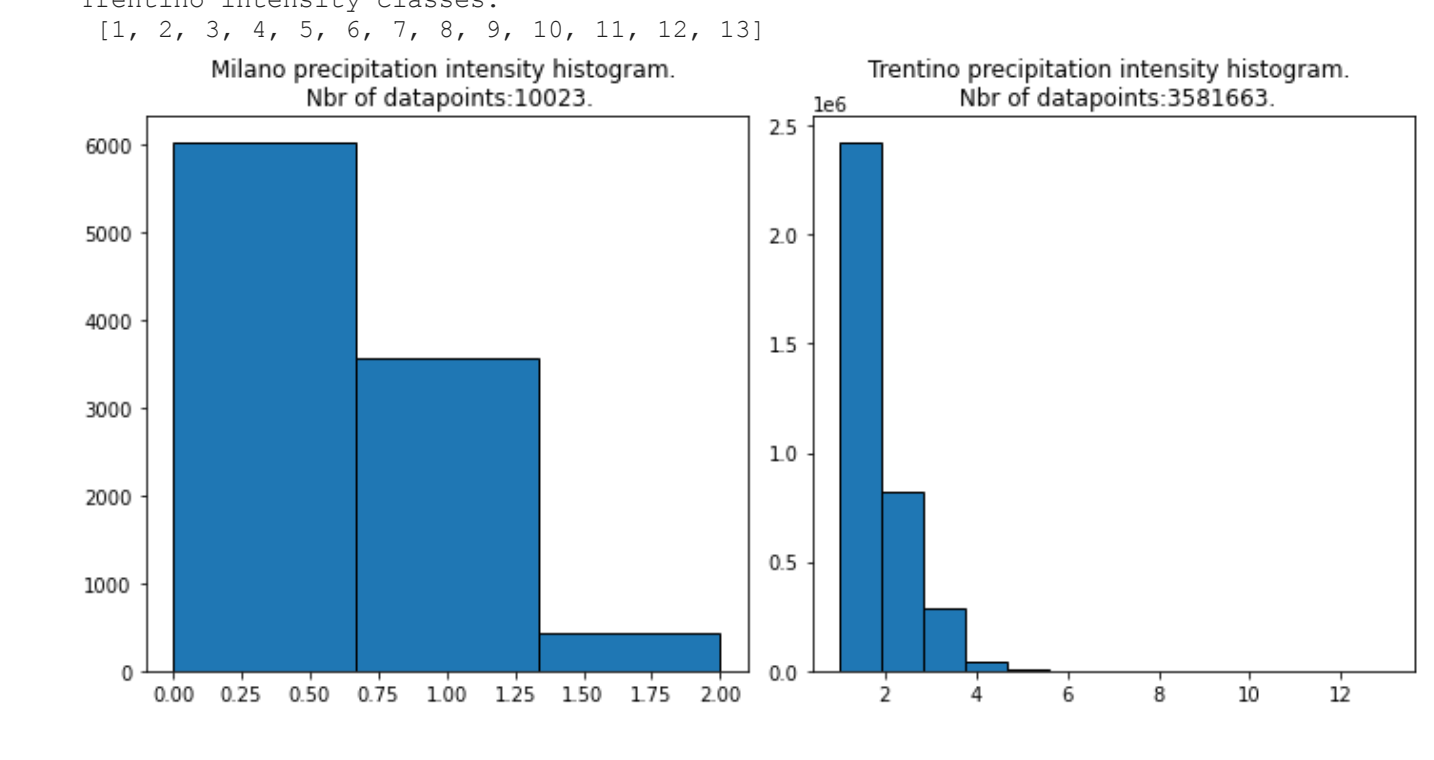
Check the intensity goups distribution in weather data

We can see that in Milan, there are no datapoints in intensity class 3 and in Trentino in intensity classes between 14 and 18. These classes mean really heavy raining.

We can also see that intensity value 0 is the most common class.

```
# Print unique intensity values in Milano and Trentino
# Used to make sure, that data description is correct.
print("Milano intensity classes: \n", sorted(list(milano["intensity"].unique())))
print("Trentino intensity classes: \n", sorted(list(trentino["intensity"].unique())))

# Make intensity histograms to see intensity distribution in Milano and Trentino.
# Milano and Trentino precipitation intensity has different scales.
# Milano scale is 0-4 and Trentino 1-18.
fig, ax = plt.subplots(1, 2, tight_layout=True, figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.hist(milano["intensity"], bins=3, edgecolor='black')
plt.title("Milano precipitation intensity histogram. \n Nbr of datapoints:{}.".format...
plt.subplot(1, 2, 2)
plt.hist(trentino["intensity"], bins=13, edgecolor='black')
plt.title("Trentino precipitation intensity histogram. \n Nbr of datapoints:{}.".for...
plt.show()
```

```
Milano intensity classes:
 [0, 1, 2]
Trentino intensity classes:
 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
```



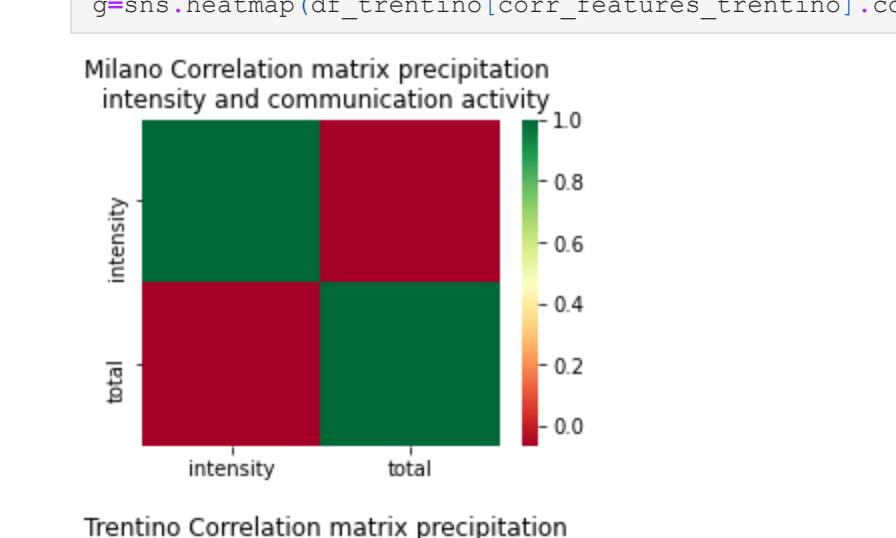
Try to find correlation values between precipitation intensity and total communication activity

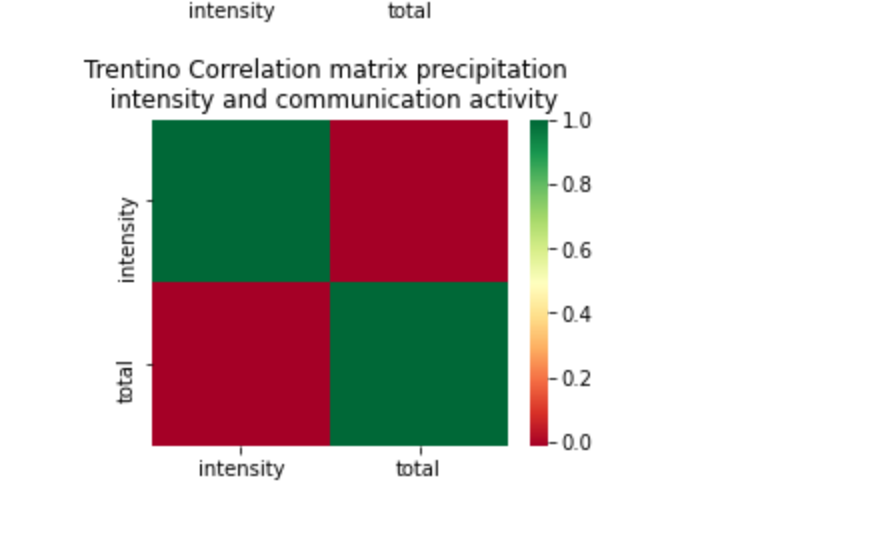
Result: Correlation value is -0.063, which is really low. According to this, there is no correlation between precipitation intensity and communication activity.

```
# Milano
df_milano = pd.DataFrame(milano[["intensity", "total"]])
print("Milano intensity vs activity correlation: \n", df_milano.corr(), "\n")

# Trentino
df_trentino = pd.DataFrame(trentino[["intensity", "total"]])
print("Trentino intensity vs activity correlation: \n", df_trentino.corr())
```

```
Milano intensity vs activity correlation:
            intensity     total
intensity   1.000000 -0.063826
total      -0.063826  1.000000

Trentino intensity vs activity correlation:
            intensity     total
intensity   1.000000 -0.012159
total      -0.012159  1.000000
```

```
# Milano
corr_milano = df_milano.corr()
corr_features_milano = corr_milano.index
plt.figure(figsize=(4,3))
plt.title("Milano Correlation matrix precipitation \n intensity and communication acti...
g=sns.heatmap(df_milano[corr_features_milano].corr(),annot=False,cmap="RdYlGn")

# Trentino
corr_trentino = df_trentino.corr()
corr_features_trentino = corr_trentino.index
plt.figure(figsize=(4,3))
plt.title("Trentino Correlation matrix precipitation \n intensity and communication ac...
g=sns.heatmap(df_trentino[corr_features_trentino].corr(),annot=False,cmap="RdYlGn")
```

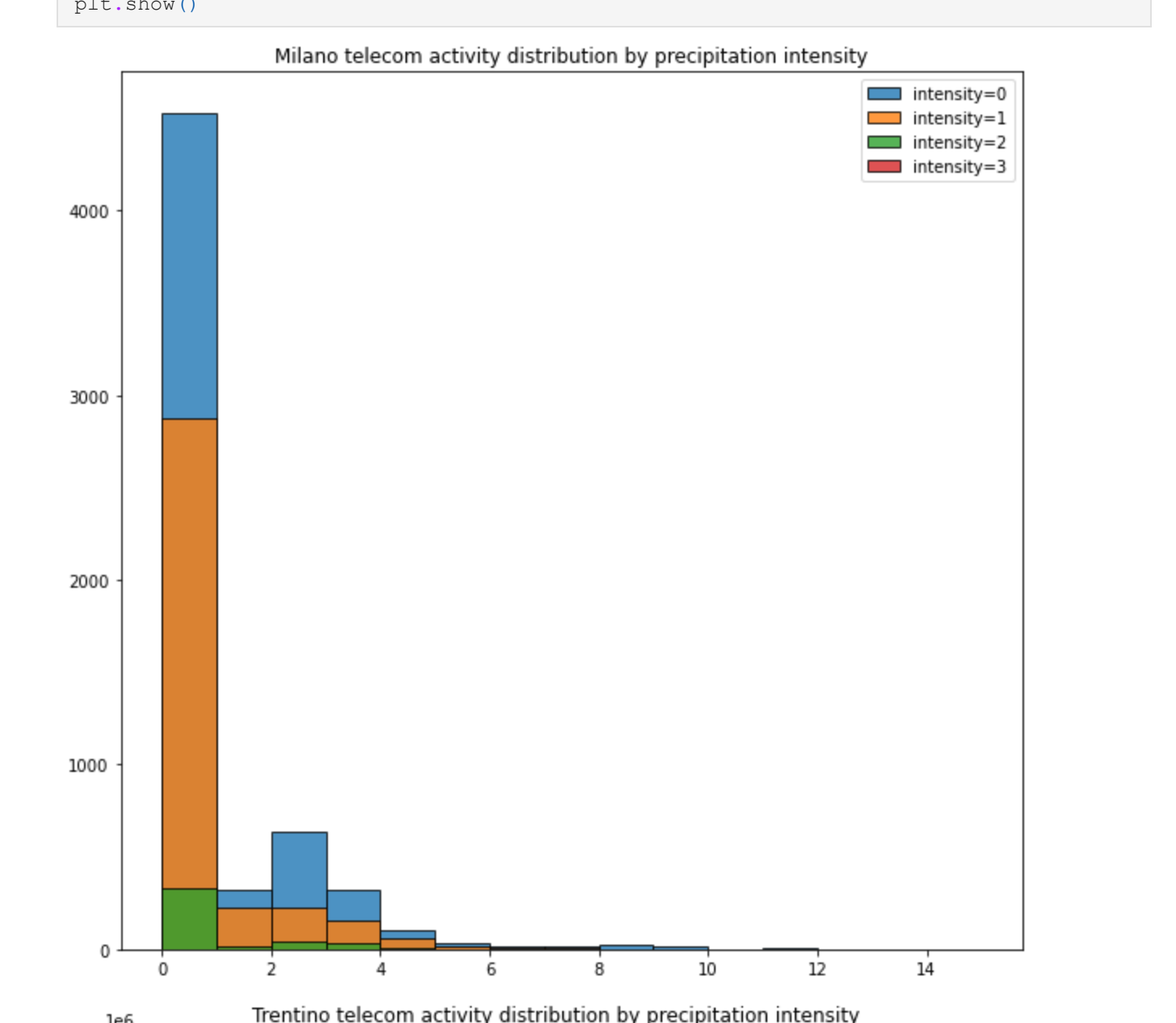


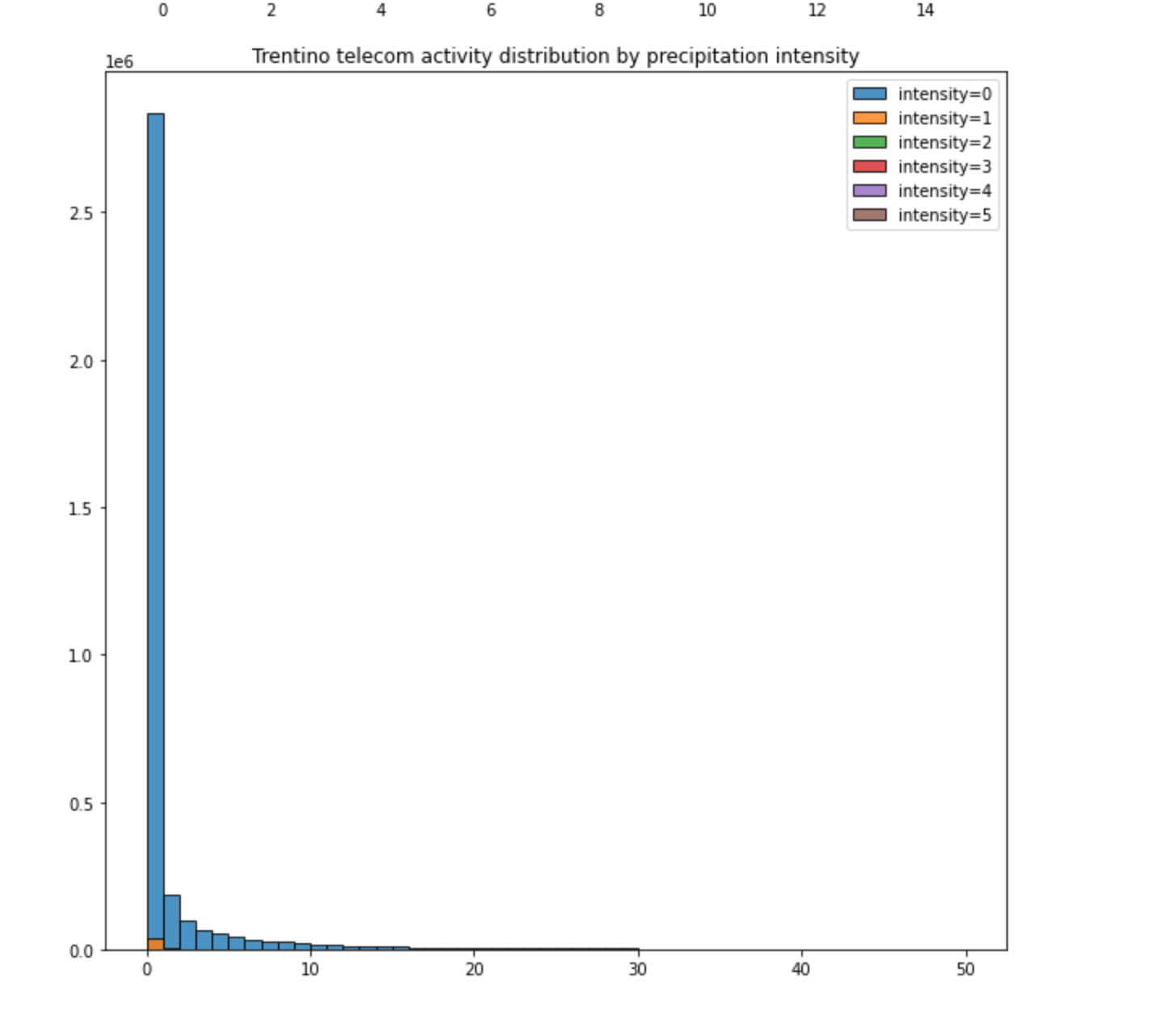


Make histograms of total activity with different precipitation intencities

Check if there are differences in total communication activity during different levels of precipitation intensity.

```
# Split Milano weather data, based on intensity, to 4 groups (intensity classes 0 to 3)
milano_w0 = milano.filter_by(0, 'intensity')
milano_w1 = milano.filter_by(1, 'intensity')
milano_w2 = milano.filter_by(2, 'intensity')
milano_w3 = milano.filter_by(3, 'intensity')

# Split Milano weather data, based on intensity, to 6 groups (intensity classes 1 to ...
trentino_w1 = trentino[(trentino['intensity'] >= 1) & (trentino['intensity'] <= 3)]
trentino_w2 = trentino[(trentino['intensity'] >= 4) & (trentino['intensity'] <= 6)]
trentino_w3 = trentino[(trentino['intensity'] >= 7) & (trentino['intensity'] <= 9)]
trentino_w4 = trentino[(trentino['intensity'] >= 10) & (trentino['intensity'] <= 12)]
trentino_w5 = trentino[(trentino['intensity'] >= 13) & (trentino['intensity'] <= 15)]
trentino_w6 = trentino[(trentino['intensity'] >= 16) & (trentino['intensity'] <= 18)]

# Plot Milano telecom activity histogram
bins = list(range(0, 15 + 1, 1))
plt.rcParams["figure.figsize"] = (10, 10)
plt.hist(milano_w0["total"], bins=bins, alpha=0.8, edgecolor='black')
plt.hist(milano_w1["total"], bins=bins, alpha=0.8, edgecolor='black')
plt.hist(milano_w2["total"], bins=bins, alpha=0.8, edgecolor='black')
plt.hist(milano_w3["total"], bins=bins, alpha=0.8, edgecolor='black')
plt.title("Milano telecom activity distribution by precipitation intensity")
plt.legend(["intensity=0", "intensity=1", "intensity=2", "intensity=3"], loc ="upper ri...
plt.show()

# Plot Trentino telecom activity histogram
bins = list(range(0, 50 + 1, 1))
plt.rcParams["figure.figsize"] = (10, 10)
plt.hist(trentino_w1["total"], bins=bins, alpha=0.8, edgecolor='black')
plt.hist(trentino_w2["total"], bins=bins, alpha=0.8, edgecolor='black')
plt.hist(trentino_w3["total"], bins=bins, alpha=0.8, edgecolor='black')
plt.hist(trentino_w4["total"], bins=bins, alpha=0.8, edgecolor='black')
plt.hist(trentino_w5["total"], bins=bins, alpha=0.8, edgecolor='black')
plt.hist(trentino_w6["total"], bins=bins, alpha=0.8, edgecolor='black')
plt.title("Trentino telecom activity distribution by precipitation intensity")
plt.legend(["intensity=0", "intensity=1", "intensity=2", "intensity=3", "intensity=4"
           ,"intensity=5", "intensity=6"], loc ="upper right")
plt.show()
```





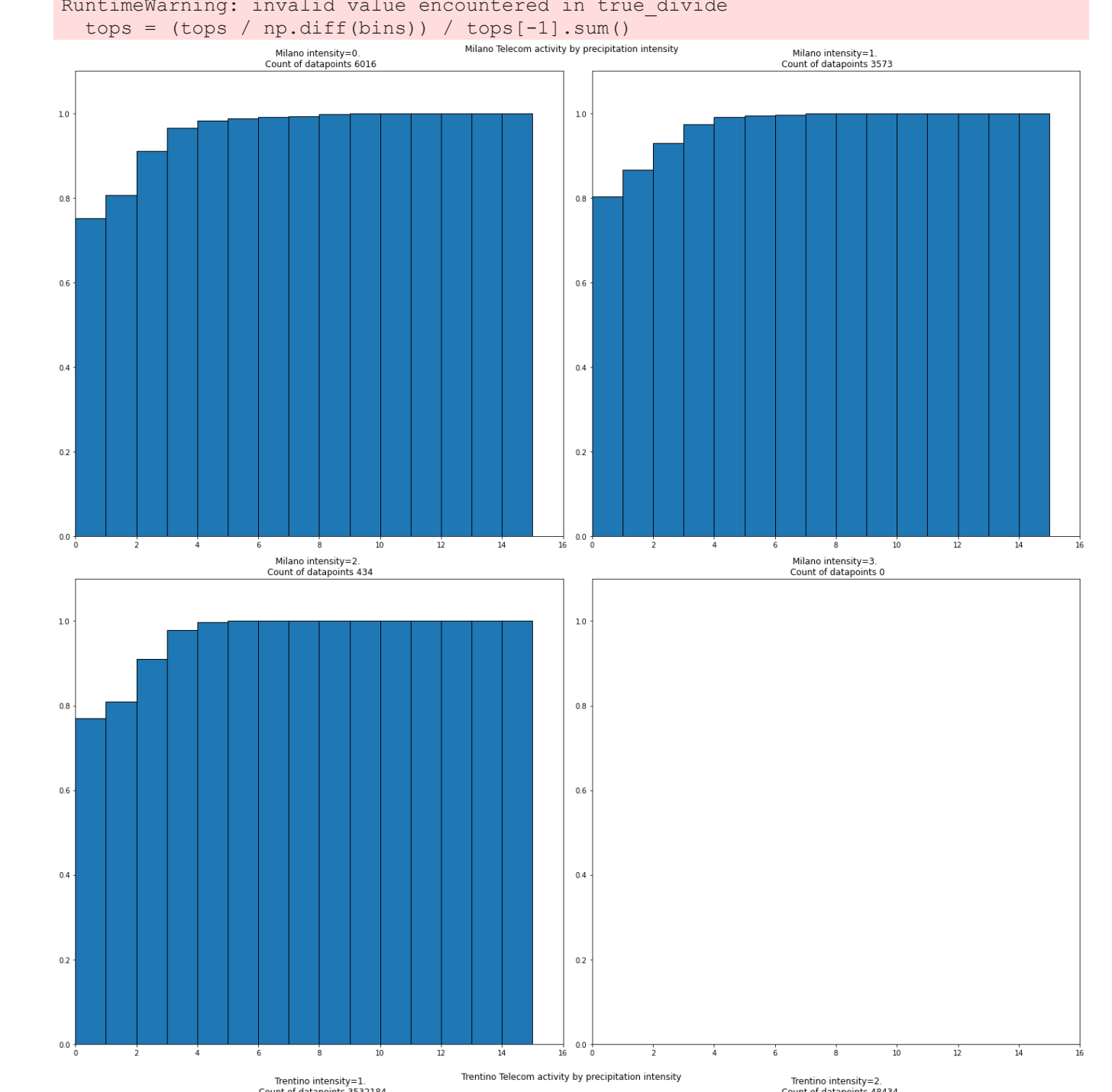
Plot same data as density histograms.

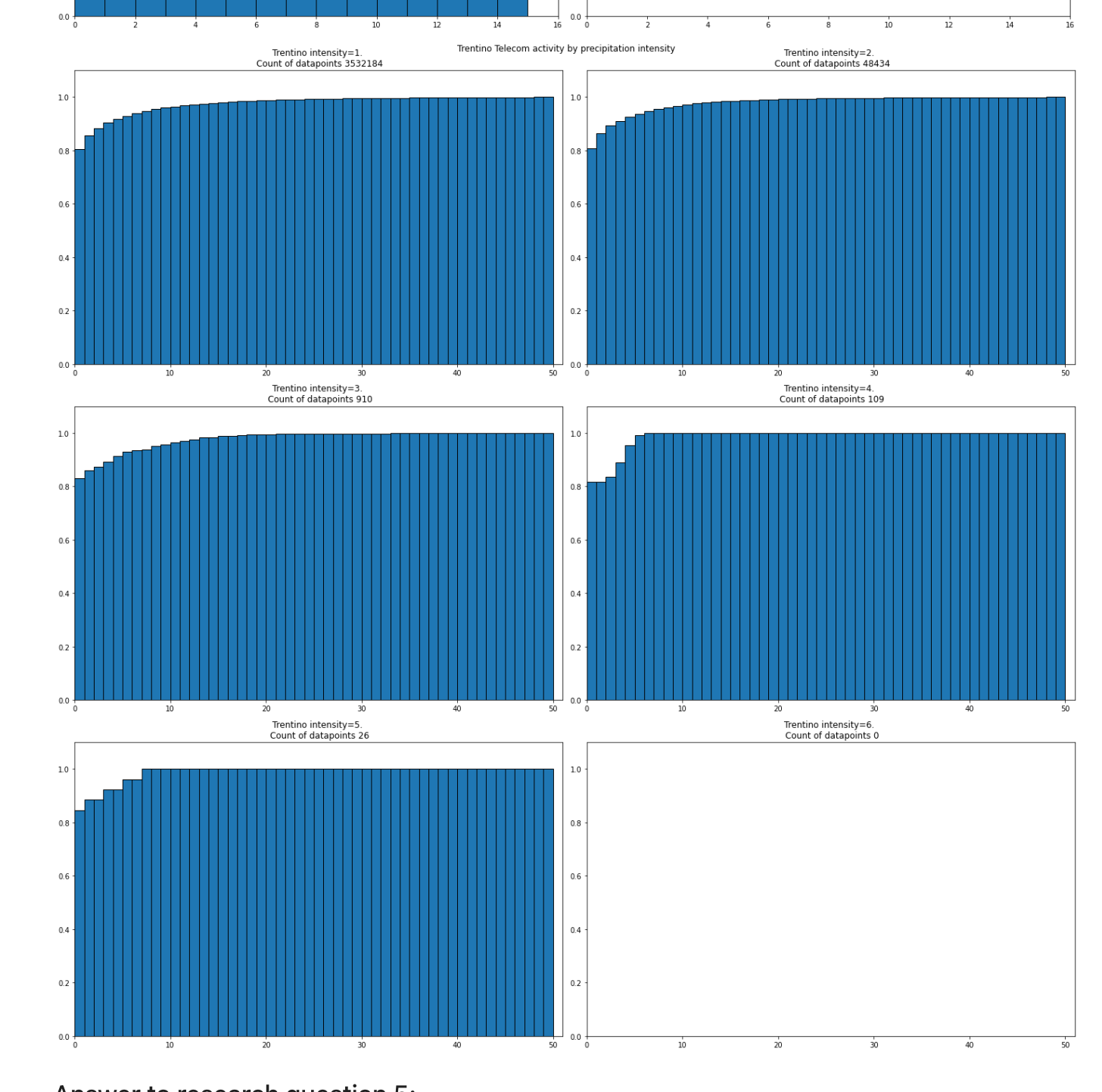
The previous plots were complicated to read, and they did not offer so much information. The main reasons behind this were that most of the activity values are small, and intensity=0 is tens of times more common than others.

Next, we will plot the same histograms as normalized density histograms. We will use normalized density histograms for two reasons. First, different classes have a different number of datapoints, which tends to scale results, which we do not want. Secondly, most of the data is in small values, making it challenging to read works from normal histograms.

```
# Set bins for first plot
bins = list(range(0, 15 + 1, 1))

# Make Milano histograms. One for each precipitation intensity class.
fig, ax = plt.subplots(2, 2, tight_layout=True, figsize=(20, 20))

# Histogram 1
plt.subplot(2, 2, 1)
plt.xlim(0, 16)
plt.ylim(0, 1.1)
plt.hist(milano_w0["total"], bins=bins, density=True, stacked=True, cumulative=True, e...
plt.title("Milano intensity=0. \n Count of datapoints {}". format(len(milano_w0)))

# Histogram 2
plt.subplot(2, 2, 2)
plt.xlim(0, 16)
plt.ylim(0, 1.1)
plt.hist(milano_w1["total"], bins=bins, density=True, stacked=True, cumulative=True, e...
plt.title("Milano intensity=1. \n Count of datapoints {}". format(len(milano_w1)))

# Histogram 3
plt.subplot(2, 2, 3)
plt.xlim(0, 16)
plt.ylim(0, 1.1)
plt.hist(milano_w2["total"], bins=bins, density=True, stacked=True, cumulative=True, e...
plt.title("Milano intensity=2. \n Count of datapoints {}". format(len(milano_w2)))

# Histogram 4
plt.subplot(2, 2, 4)
plt.xlim(0, 16)
plt.ylim(0, 1.1)
plt.hist(milano_w3["total"], bins=bins, density=True, stacked=True, cumulative=True, e...
plt.title("Milano intensity=3. \n Count of datapoints {}". format(len(milano_w3)))

plt.suptitle('Milano Telecom activity by precipitation intensity')
plt.show()

# Set bins for second plot
bins = list(range(0, 50 + 1, 1))

# Make Trentino histograms. One for each precipitation intensity class.
fig, ax = plt.subplots(3, 2, tight_layout=True, figsize=(20, 20))

# Histogram 1
plt.subplot(3, 2, 1)
plt.ylim(0, 1.1)
plt.xlim(0, 51)
plt.hist(trentino_w1["total"], bins=bins, density=True, stacked=True, cumulative=True,...
plt.title("Trentino intensity=1. \n Count of datapoints {}". format(len(trentino_w1))...

# Histogram 2
plt.subplot(3, 2, 2)
plt.ylim(0, 1.1)
plt.xlim(0, 51)
plt.hist(trentino_w2["total"], bins=bins, density=True, stacked=True, cumulative=True,...
plt.title("Trentino intensity=2. \n Count of datapoints {}". format(len(trentino_w2))...

# Histogram 3
plt.subplot(3, 2, 3)
plt.ylim(0, 1.1)
plt.xlim(0, 51)
plt.hist(trentino_w3["total"], bins=bins, density=True, stacked=True, cumulative=True,...
plt.title("Trentino intensity=3. \n Count of datapoints {}". format(len(trentino_w3))...

# Histogram 4
plt.subplot(3, 2, 4)
plt.ylim(0, 1.1)
plt.xlim(0, 51)
plt.hist(trentino_w4["total"], bins=bins, density=True, stacked=True, cumulative=True,...
plt.title("Trentino intensity=4. \n Count of datapoints {}". format(len(trentino_w4))...

# Histogram 5
plt.subplot(3, 2, 5)
plt.ylim(0, 1.1)
plt.xlim(0, 51)
plt.hist(trentino_w5["total"], bins=bins, density=True, stacked=True, cumulative=True,...
plt.title("Trentino intensity=5. \n Count of datapoints {}". format(len(trentino_w5))...

# Histogram 6
plt.subplot(3, 2, 6)
plt.ylim(0, 1.1)
plt.xlim(0, 51)
plt.hist(trentino_w6["total"], bins=bins, density=True, stacked=True, cumulative=True,...
plt.title("Trentino intensity=6. \n Count of datapoints {}". format(len(trentino_w6))...

plt.suptitle('Trentino Telecom activity by precipitation intensity')
plt.show()
```

```
/Users/saukk/opt/anaconda3/lib/python3.8/site-packages/matplotlib/axes/_axes.py:6664:
RuntimeWarning: invalid value encountered in true_divide
  tops = (tops / np.diff(bins)) / tops[-1].sum()
```





Answer to research question 5:

We can see from the results that there is no direct correlation between communication activity and precipitation intensity. The correlation value between these two numbers is -0.063 for Milano and -0.012 for Trentino, which is low. We used only six days from November and December, which may affect the results in the analysis. I tried to run my algorithm for a larger number of days, but my laptop could not handle that amount of data.

We can detect small differences in the density histograms when we compare them from different precipitation intensity times. There are more active communication periods when the precipitation intensity is low or not raining. The difference is not big, but it would make sense that people meet to communicate outdoors with each other more frequently, also with phones, when the weather is good.

6. Plot the heatmap of user telecommunication activity for both Milan and Trentino. Do you observe any shift in communication pattern of users between day and night? (A typical day time is between 8AM to 8PM)

In part 4, we made hourly graphs for telecommunication activity for different days. From those graphs, we could easily see how the activity depends on the time. We also saw how day affects to the activity, when one of the days was Christmas.

To add new information about the question, about communication patterns relationship to the time, we will make two map heatmaps of the telecommunication activity for both locations. The first map has activity from 8 am to 8 pm, and the second map has activity from 8 pm to 8am.

Telecommunication activity is calculated as the sum of the call and SMS activities. Then we will take the average total activity for each grid during AM and PM times. We will use average instead of sum because some grid locations may have more data points than others, which makes sum unreliable.

First prepare telecom data

```
# We will use same SFrame we made in the part 5.
milano_tele = milano_tele
trentino_tele = trentino_tele

# Add column am_pm, where am=(8am to 8pm) and pm=(8pm to 8am)
# First add column hour
milano_tele["timestamp"] = milano_tele["timestamp"].astype(str)
trentino_tele["timestamp"] = trentino_tele["timestamp"].astype(str)
milano_tele["hour"] = milano_tele["timestamp"].apply(lambda timestamp: timestamp[-4:-...
trentino_tele["hour"] = trentino_tele["timestamp"].apply(lambda timestamp: timestamp[...

# Next add col am_pm based on value in hour column
#lambda x: True if x % 2 == 0 else False
milano_tele["hour"] = milano_tele["hour"].astype(int)
trentino_tele["hour"] = trentino_tele["hour"].astype(int)
milano_tele["am_pm"] = milano_tele["hour"].apply(lambda hour: "am" if (hour <= 20 and...
trentino_tele["am_pm"] = trentino_tele["hour"].apply(lambda hour: "am" if (hour <= 20...

# Group by SquareID and am_pm, and find average activity.
# We will use AVG activity, because there may be different number of
# datapoints in different locations so AVG and Mean are best options.
milano_tele_am_pm = milano_tele.groupby(['SquareID', 'am_pm'], {'avg_total':agg.AVG('...
trentino_tele_am_pm = trentino_tele.groupby(['SquareID', 'am_pm'], {'avg_total':agg.A...

# Print first 5 rows
milano_tele_am_pm.print_rows(num_rows=5)
trentino_tele_am_pm.print_rows(num_rows=5)
```

```
+----------+-------+--------------------+
| SquareID | am_pm |     avg_total      |
+----------+-------+--------------------+
|   6693   |   am  |  3.64029210554621  |
|   1422   |   am  | 0.4693612318534571 |
|   765    |   am  | 3.720867447194179  |
|   1980   |   am  | 1.8467841892435237 |
|   3249   |   am  | 1.5367980047965475 |
+----------+-------+--------------------+
[20000 rows x 3 columns]

+----------+-------+--------------------+
| SquareID | am_pm |     avg_total      |
+----------+-------+--------------------+
|   6693   |   am  | 0.7245303753561196 |
|  10701   |   am  | 1.8849917472606825 |
|   5758   |   am  | 2.9425561068658643 |
|   4467   |   am  | 3.6736246928213645 |
|   6664   |   am  | 0.9505704427980467 |
+----------+-------+--------------------+
[12518 rows x 3 columns]
```

Read the grip data and join coordinated to telecomm data

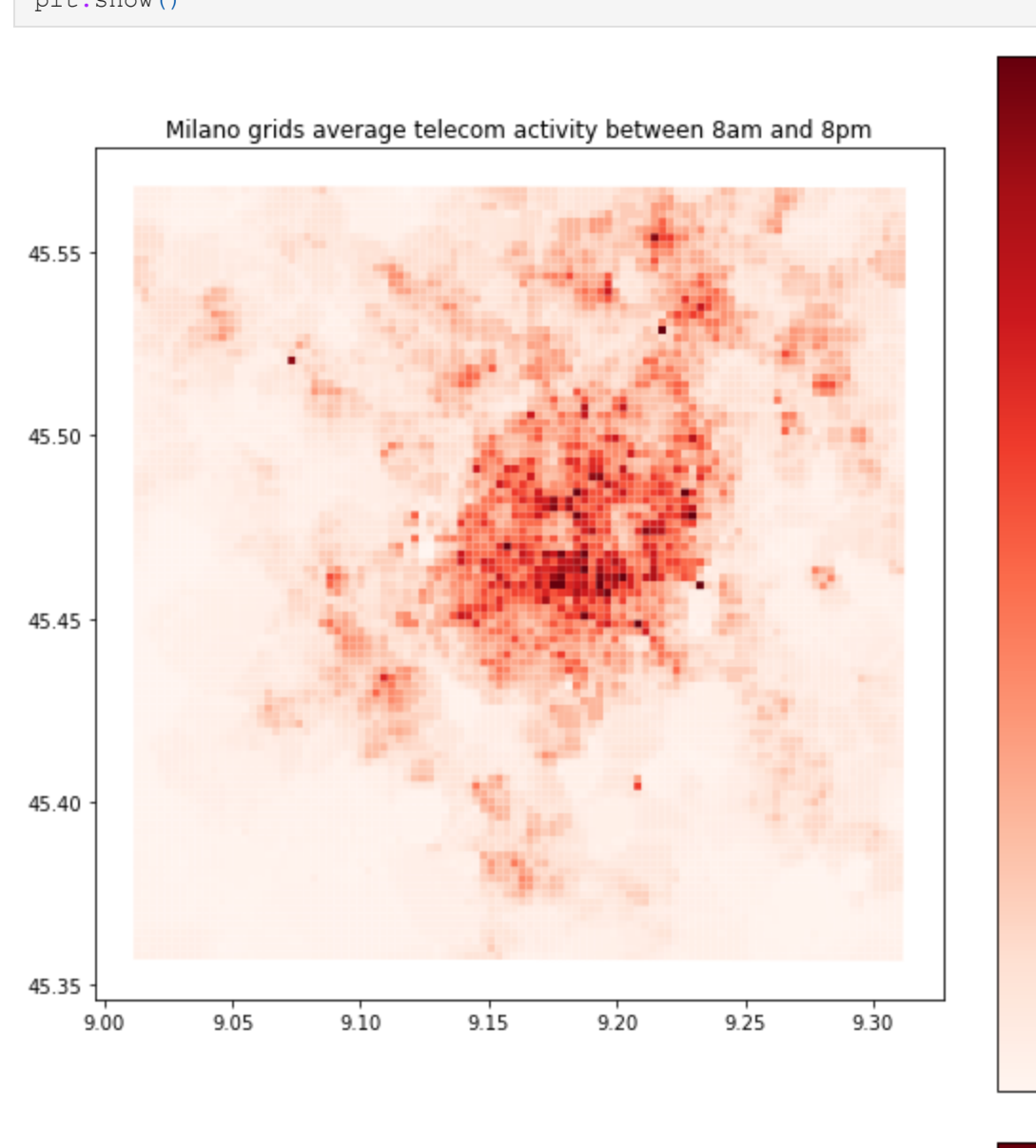
Turi create is not able to handle geoJSON file format, so we will use geopandas.

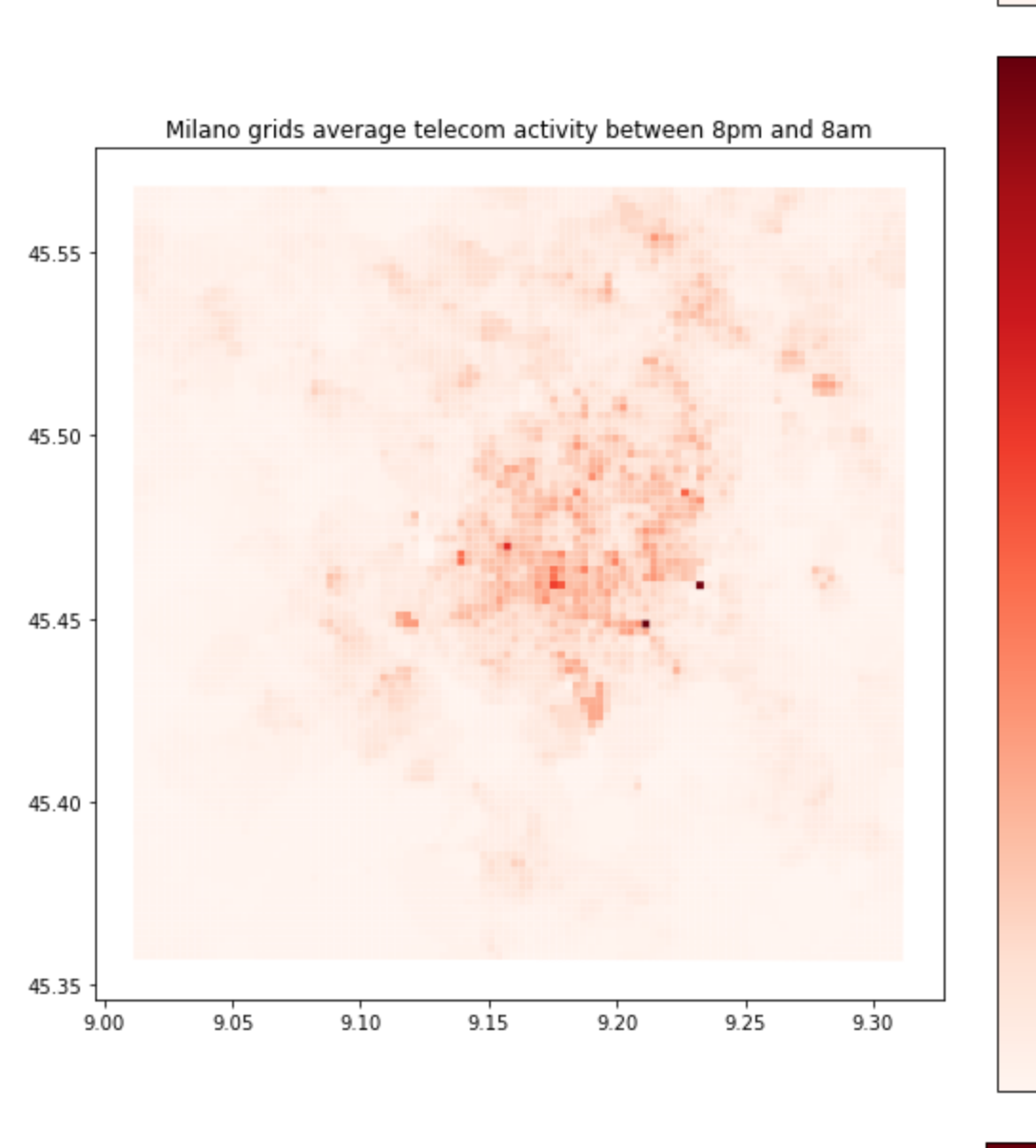
```
# Read grid files
milano_grid = gpd.read_file("data/milano-grid.geojson")
trentino_grid = gpd.read_file("data/trentino-grid.geojson")

# Rename grid data column cellId to SquareID
milano_grid.rename(columns={"cellId" : 'SquareID'}, inplace=True)
trentino_grid.rename(columns={"cellId" : 'SquareID'}, inplace=True)

# Convert SFrames to DataFrames
milano_tele = pd.DataFrame(milano_tele_am_pm)
trentino_tele = pd.DataFrame(trentino_tele_am_pm)

# Join grid data to telecomm data
milano = milano_grid.merge(milano_tele, on='SquareID')
trentino = trentino_grid.merge(trentino_tele, on='SquareID')

# Print first 5 rows
milano.head(5)
trentino.head(5)
```

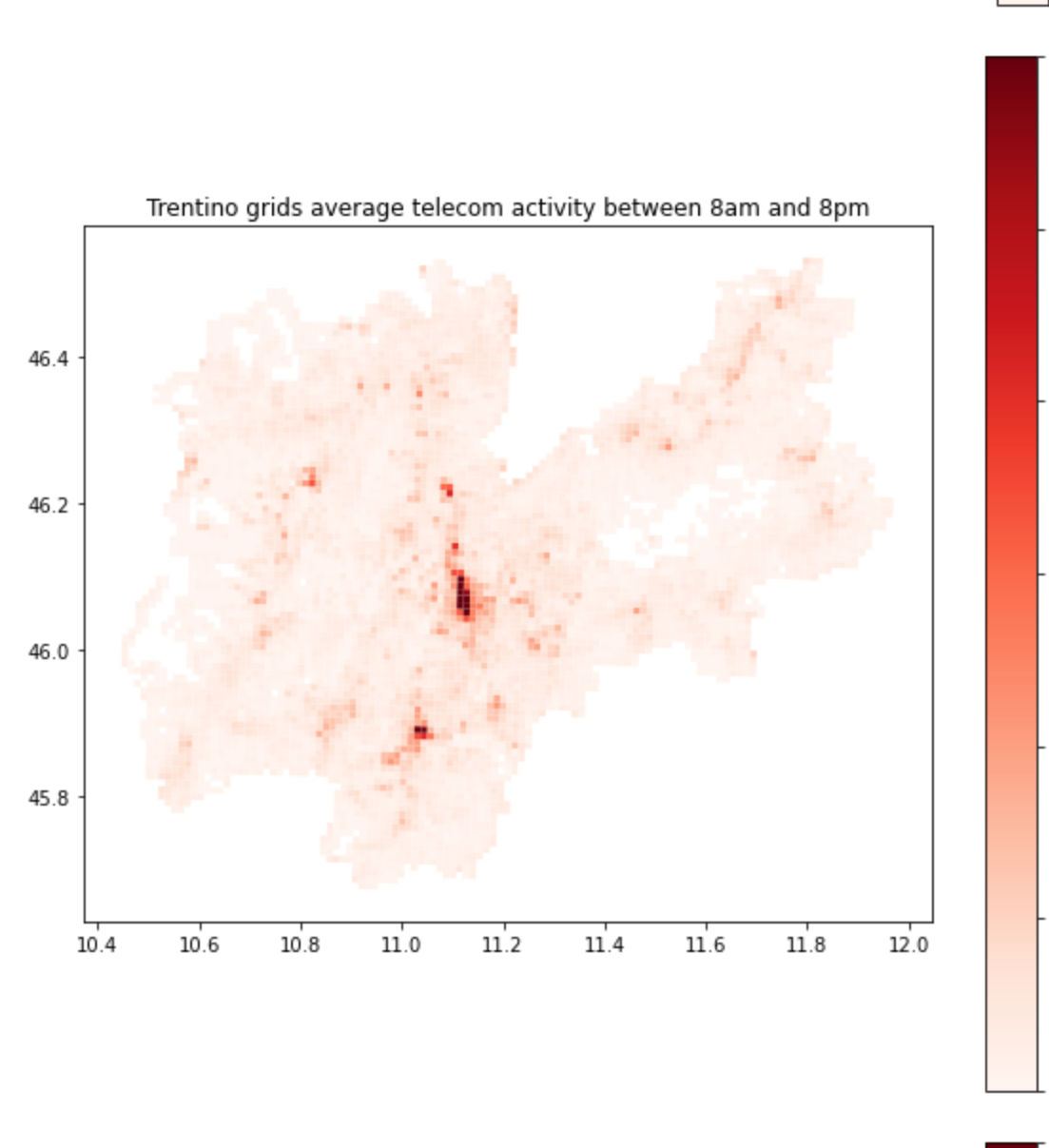
	SquareID	geometry	am_pm	avg_total
0	38	POLYGON ((10.91462 45.68201, 10.92746 45.68179...	am	0.764647
1	38	POLYGON ((10.91462 45.68201, 10.92746 45.68179...	pm	0.276707
2	39	POLYGON ((10.92746 45.68179, 10.94029 45.68157...	am	0.549747
3	39	POLYGON ((10.92746 45.68179, 10.94029 45.68157...	pm	0.170299
4	40	POLYGON ((10.94029 45.68157, 10.95312 45.68136...	am	0.524520

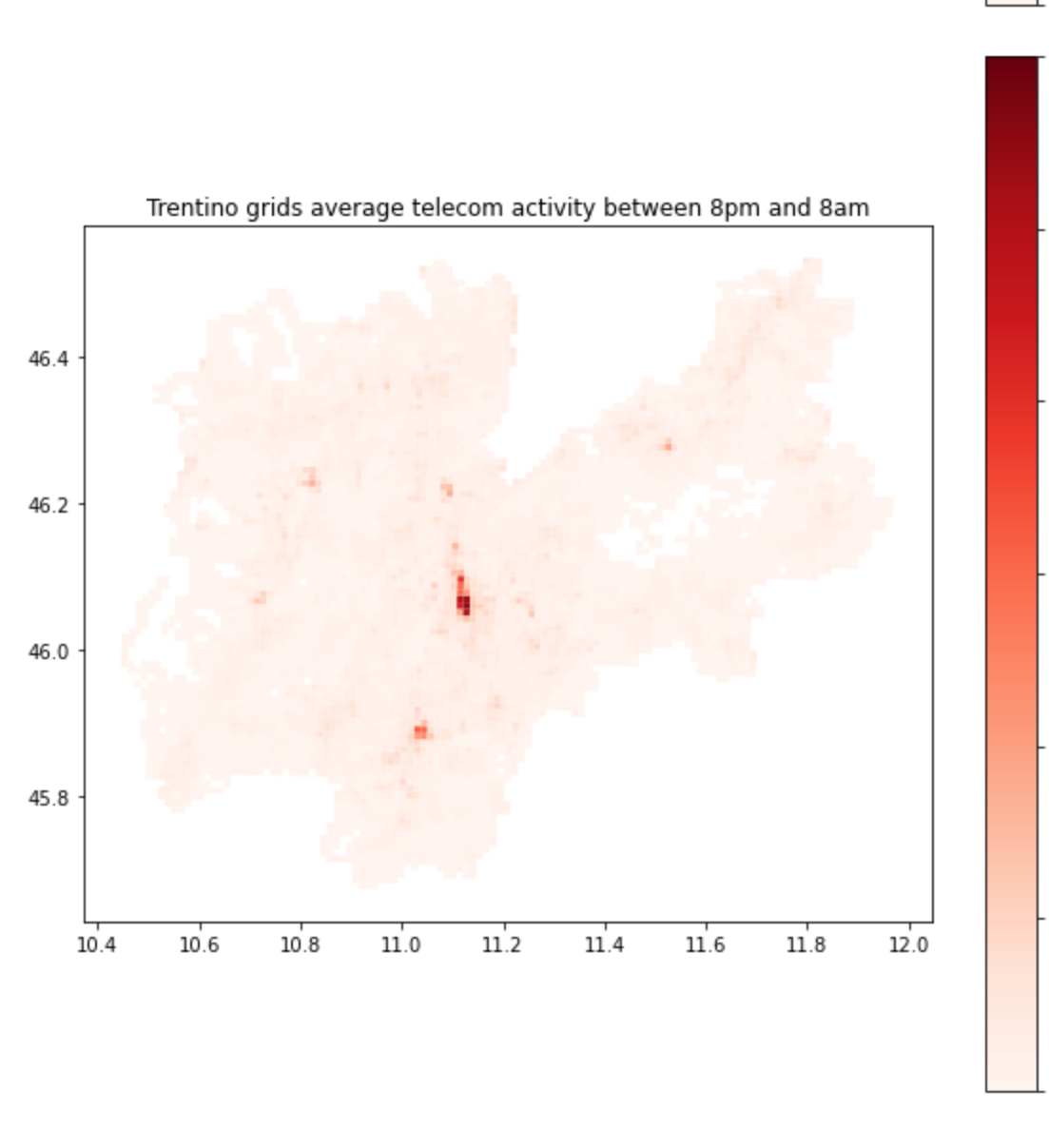
Plot the heat maps

```
# Split Milano and Trentino data to AM and PM datasets
milano_am = milano[milano["am_pm"] == "am"]
milano_pm = milano[milano["am_pm"] == "pm"]
trentino_am = trentino[trentino["am_pm"] == "am"]
trentino_pm = trentino[trentino["am_pm"] == "pm"]

# Plot Milano AM
milano_am.plot(column='avg_total', legend=True, cmap="Reds", vmin=0, vmax=30)
plt.title("Milano grids average telecom activity between 8am and 8pm")
plt.show()

# Plot Milano PM
milano_pm.plot(column='avg_total', legend=True, cmap="Reds", vmin=0, vmax=30)
plt.title("Milano grids average telecom activity between 8pm and 8am")
plt.show()

# Plot Trentino AM
trentino_am.plot(column='avg_total', legend=True, cmap="Reds", vmin=0, vmax=30)
plt.title("Trentino grids average telecom activity between 8am and 8pm")
plt.show()

# Plot Trentino PM
trentino_pm.plot(column='avg_total', legend=True, cmap="Reds", vmin=0, vmax=30)
plt.title("Trentino grids average telecom activity between 8pm and 8am")
plt.show()
```









Answer to research question 6:

As we previously mentioned, we had already found the answer to this question in part 4. The time of the day has a significant impact on telecommunication activity, and activity is higher during the daytime.

We found the same results from the heatmaps. It seems like the activity is highest during the day time in all grid locations. We can also spot the city centers' from the heatmaps, and it looks like there is some activity even during the night, but still significantly less than during the daytime.

7. Investigate and plot the correlation between air quality and weather (temperature, sunshine, precipitation, etc.).

This exercise will use Milano's data because we do not have pollution data for Trentino. We will study if there is a correlation between precipitation intensity and air quality (pollution µg/m).

In the study, we will use 10minutes intervals from both measurements to make our results as precise as possible. The dataset has measurements for 9 different pollution types and because their properties and amount differ significantly, we will study all of them separately.

To answer this question, we will use data from three different datasets. From the first dataset, we get the amount of pollution, from the second the precipitation information, and the third type of pollution.

Milano pollution dataset has columns:

1. Location in format: id
2. Date in format: YYYY/MM/DD
3. Amount of pollution in format: µg/m

Milano weather dataset columns:

1. Timestamp: timestamp value with the following format: YYYYMMDDHHmm;
2. Square id: id of a given square of Milano GRID;
3. Intensity: intensity value of the precipitation. It is a value between 0 and 3;
4. Coverage: percentage value of the quadrant covered by the precipitation;
5. Type: type of the precipitation. It is a value between 0 and 2. Classes are: Absent=0, Rain=1 and Snow=2.

Milano intensity classes are defined as follow:

1. Absent: precipitation quantity equal to 0 mm/h. Defined as type 0.
2. Slight: precipitation quantity equal in [0,2] mm/h. Defined as type 1.
3. Moderate: precipitation quantity equal in [2,10] mm/h. Defined as type 2.
4. Heavy: precipitation quantity equal in [10,100] mm/h. Defined as type 3.

Pollution information has following format:

1. Station id.
2. Station name

3. Station latitude
4. Station longitude
5. Pollution_type
6. Pollution_measure
7. Date format

Define function to read pollution data

Define function to read pollution data. Time is in format: 2013/12/31 04:00, but if using Turi read_csv to read multiple files, it drops the hours and minutes. This is because, some of the files have only date and no time, but we want to use both to increase accuracy.

Also change timestamp format to match format used in weather data: 2013/12/31 04:00 -> 201312310400.

```
# Define function to read multiple telecom files and append them to one SFrame
def create_pollution_SFrame(paths):
    # Initialize results of
    sf = SFrame({'id':[1],'timestamp':[1], 'pollution':[1]})
    sf["id"] = sf["id"].astype(int)
    sf["timestamp"] = sf["timestamp"].astype(int)
    sf["pollution"] = sf["pollution"].astype(float)

    # Read the files and append dataframes to result_df
    for path in paths:
        sf_tmp = SFrame.read_csv(path, sep=',', header=None, column_type_hints=str)
        sf_tmp = sf_tmp.rename({'X1': 'id', 'X2':'timestamp', 'X3':"pollution"})
        if len(sf_tmp["timestamp"][0]) > 12:
            sf_tmp["id"] = sf_tmp["id"].astype(int)
            sf_tmp["timestamp"] = sf_tmp["timestamp"].astype(str)
            sf_tmp["pollution"] = sf_tmp["pollution"].astype(float)
            sf = sf.append(sf_tmp)

    # Change timestamp format
    sf["timestamp"] = sf["timestamp"].apply(lambda time: time.replace(" ", ""))
    sf["timestamp"] = sf["timestamp"].apply(lambda time: time.replace(":", ""))
    sf["timestamp"] = sf["timestamp"].apply(lambda time: time.replace("/", ""))
    sf["timestamp"] = sf["timestamp"].astype(int)

    # Return results
    return sf
```

Read pollution data, weather data and pollution information data.

Pollution information data has coordinates of the stations and pollution types.

Grid data has information of weather data locations.

```
# Define path to folder of Milano pollution data files
pollution_paths = glob2.glob('data/milano_pollution_data/*')

# Run milano_pollution function and get all Milano pollution data
pollution = create_pollution_SFrame(pollution_paths)

# Read weather files and rename columns
weather = SFrame.read_csv('data/MeteoMilano_01-11-13_01-01-14.csv', sep=",", header=No...
weather = weather.rename({'X1':'timestamp', 'X2':'quadID', 'X3':'intensity', 'X4':"cov...

# Read pollution information file
pollution_info = SFrame.read_csv('data/milano_airQuality/milano_pollution_legend2.csv...
pollution_info = pollution_info.rename({'X1':'id', 'X2':'name', 'X3':'lat', 'X4':'lon'...
                  'X5':'pollution_type', 'X6':"pollution_measure"...
                  'X7':"date_format"})

# Print first rows
pollution.print_rows(num_rows=5)
pollution_info.print_rows(num_rows=5)
weather.print_rows(num_rows=5)
```

```
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_6956.csv
Parsing completed. Parsed 54 lines in 0.004946 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_5722.csv
Parsing completed. Parsed 1353 lines in 0.005888 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_6372.csv
Parsing completed. Parsed 1460 lines in 0.006122 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_6366.csv
Parsing completed. Parsed 1332 lines in 0.006145 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_5531.csv
Parsing completed. Parsed 1312 lines in 0.006096 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_5725.csv
Parsing completed. Parsed 1312 lines in 0.007266 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_10278.csv
Parsing completed. Parsed 1459 lines in 0.006753 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_10279.csv
Parsing completed. Parsed 1460 lines in 0.008044 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_5542.csv
Parsing completed. Parsed 1461 lines in 0.006125 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_6328.csv
Parsing completed. Parsed 1462 lines in 0.006164 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_6062.csv
Parsing completed. Parsed 1462 lines in 0.006669 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_20020.csv
Parsing completed. Parsed 1460 lines in 0.006328 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_10280.csv
Parsing completed. Parsed 1461 lines in 0.006164 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_5552.csv
Parsing completed. Parsed 743 lines in 0.006568 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_5550.csv
Parsing completed. Parsed 1462 lines in 0.006142 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_10282.csv
Parsing completed. Parsed 1352 lines in 0.00611 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_10283.csv
Parsing completed. Parsed 1462 lines in 0.005863 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_5551.csv
Parsing completed. Parsed 1062 lines in 0.00758 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_10320.csv
Parsing completed. Parsed 1461 lines in 0.00566 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_17126.csv
Parsing completed. Parsed 1462 lines in 0.006261 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_17127.csv
Parsing completed. Parsed 61 lines in 0.006678 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_20004.csv
Parsing completed. Parsed 1462 lines in 0.005799 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_6320.csv
Parsing completed. Parsed 59 lines in 0.006378 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_10273.csv
Parsing completed. Parsed 1461 lines in 0.006054 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_5827.csv
Parsing completed. Parsed 1461 lines in 0.006297 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_20005.csv
Parsing completed. Parsed 1484 lines in 0.006091 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_6057.csv
Parsing completed. Parsed 692 lines in 0.006229 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_5823.csv
Parsing completed. Parsed 1462 lines in 0.006112 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_5834.csv
Parsing completed. Parsed 61 lines in 0.006105 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_17122.csv
Parsing completed. Parsed 1461 lines in 0.005916 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_6354.csv
Parsing completed. Parsed 88 lines in 0.006828 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_6340.csv
Parsing completed. Parsed 1462 lines in 0.005991 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_5841.csv
Parsing completed. Parsed 1462 lines in 0.006239 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_5504.csv
Parsing completed. Parsed 1462 lines in 0.006116 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_5506.csv
Parsing completed. Parsed 59 lines in 0.006011 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_pollution_data/mi_pollution_6344.csv
Parsing completed. Parsed 1462 lines in 0.007431 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/MeteoMilano_01-11-13_01-01-14.csv
Parsing completed. Parsed 100 lines in 0.008735 secs.
------------------------------------------------------
Inferred types from first 100 line(s) of file as
column_type_hints=[int,int,int,int]
If parsing fails due to incorrect types, you can correct
the inferred type list above and pass it to read_csv in
the column_type_hints argument
------------------------------------------------------
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/MeteoMilano_01-11-13_01-01-14.csv
Parsing completed. Parsed 33316 lines in 0.012019 secs.
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_airQuality/milano_pollution_legend2.csv
Parsing completed. Parsed 36 lines in 0.005387 secs.
------------------------------------------------------
Inferred types from first 100 line(s) of file as
column_type_hints=[int,str,float,float,str,str,str]
If parsing fails due to incorrect types, you can correct
the inferred type list above and pass it to read_csv in
the column_type_hints argument
------------------------------------------------------
Finished parsing file /Users/saukk/Desktop/Distributed data infrastructure/GraphLab project/data/milano_airQuality/milano_pollution_legend2.csv
Parsing completed. Parsed 36 lines in 0.006151 secs.
```

id	pollution	timestamp
5722	10.0	201312310100
5722	9.0	201312310200
5722	9.0	201312310300
5722	10.0	201312310400
5722	10.0	201312310500

[40994 rows x 3 columns]

id	name	lat	lon	pollution_type
20020	Milano - via Carlo Pascal	45.478452	9.235016	Ammonia
17127	Milano - viale Marche	45.496067	9.193023	Benzene
17126	Milano -via Carlo Pascal	45.478452	9.235016	Benzene
6057	Milano - via Senato	45.47078	9.19718	Benzene
6062	Milano - P.zza Zavattari	45.476089	9.143509	Benzene

pollution_measure	date_format
µg/m	YYYY/MM/DD
µg/m	YYYY/MM/DD HH24:MI
µg/m	YYYY/MM/DD HH24:MI
µg/m	YYYY/MM/DD HH24:MI
µg/m	YYYY/MM/DD HH24:MI

[36 rows x 7 columns]

timestamp	quadID	intensity	coverage	prec_type
201311010000	1	0	0	0
201311010000	2	0	0	0
201311010000	3	0	0	0
201311010000	4	0	0	0
201311010000	5	0	0	0

[33316 rows x 5 columns]

Join pollution and pollution_info to pollution, calculate weather quadrants average and join pollution and weather by timestamps.

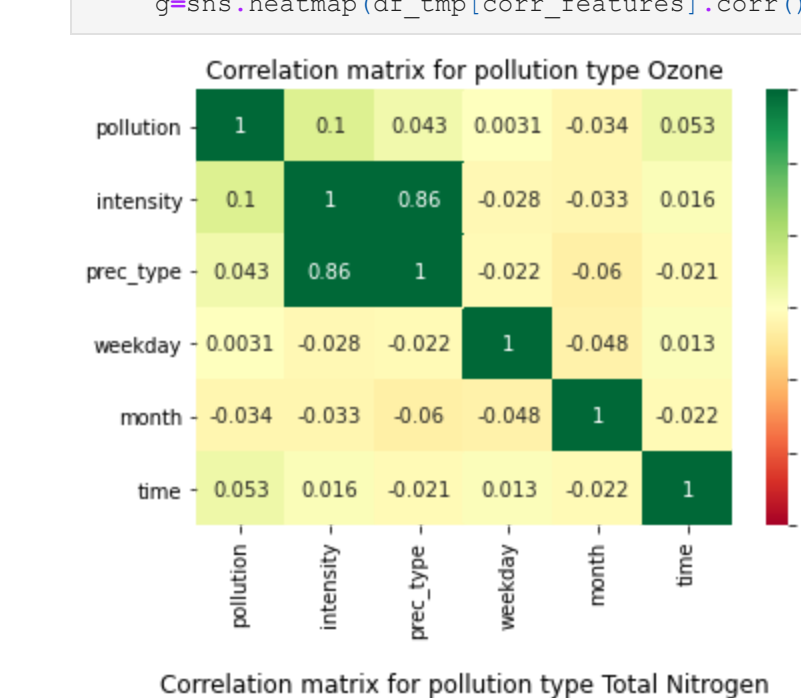
```
# Join pollution and pollution_info to pollution2
pollution2 = pollution.join(pollution_info, on="id", how="inner")

# Drop columns from pollution2, that we do not need.
pollution2 = pollution2.remove_column("name")
pollution2 = pollution2.remove_column("pollution_measure")
pollution2 = pollution2.remove_column("date_format")
pollution2 = pollution2.remove_column("lat")
pollution2 = pollution2.remove_column("lon")

# Group weather by timestamp and take the average of intensity and max of type.
# Weather data was in 4 quadrants, but because we do not have their
# coordinates, we will use avg intensity and max type to whole Milano.
weather2 = weather.groupby(["timestamp"], {"intensity":agg.AVG("intensity"),
                           "prec_type": agg.MAX("prec_type")})

# Join weather and pollution data
milano = pollution2.join(weather2, on="timestamp", how="inner")

# Add columns weekday, time and month, Monday=1 and sunday=7.
milano["timestamp"] = milano["timestamp"].astype(str)
milano["weekday"] = milano["timestamp"].apply(lambda time: (datetime.date(int(time[0:...
milano["month"] = milano["timestamp"].apply(lambda time: time[4:6])
milano["month"] = milano["month"].astype(int)
milano["time"] = milano["timestamp"].apply(lambda time: time[-4:])
milano["time"] = milano["time"].astype(int)

# Drop id and timestamp
milano = milano.remove_column('id')
milano = milano.remove_column('timestamp')

# Print first 5 rows of both
milano.print_rows(num_rows=5)
```

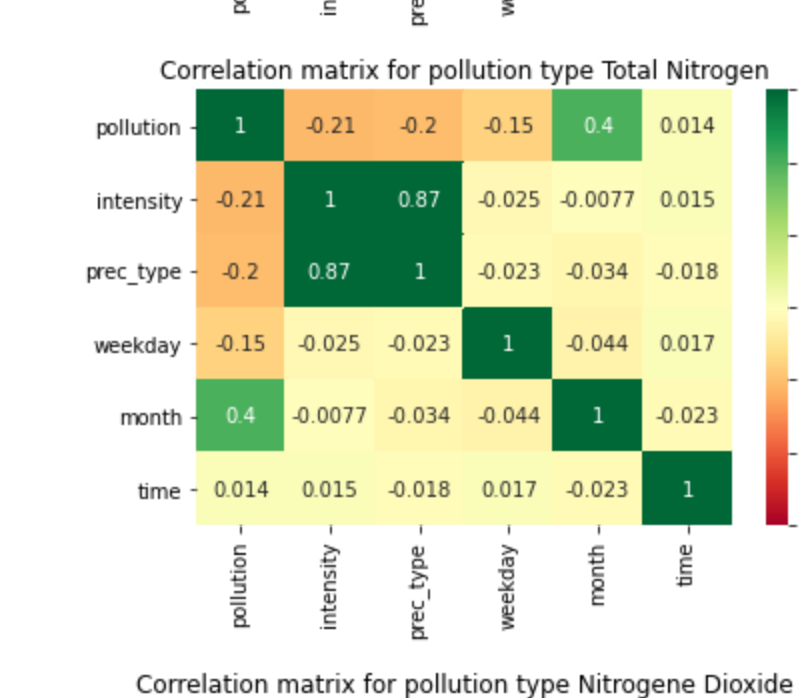
pollution	pollution_type	intensity	prec_type	weekday	month	time
10.0	Ozono	0.0	0	2	12	100
9.0	Ozono	0.0	0	2	12	200
9.0	Ozono	0.0	0	2	12	300
10.0	Ozono	0.0	0	2	12	400
10.0	Ozono	0.0	0	2	12	500

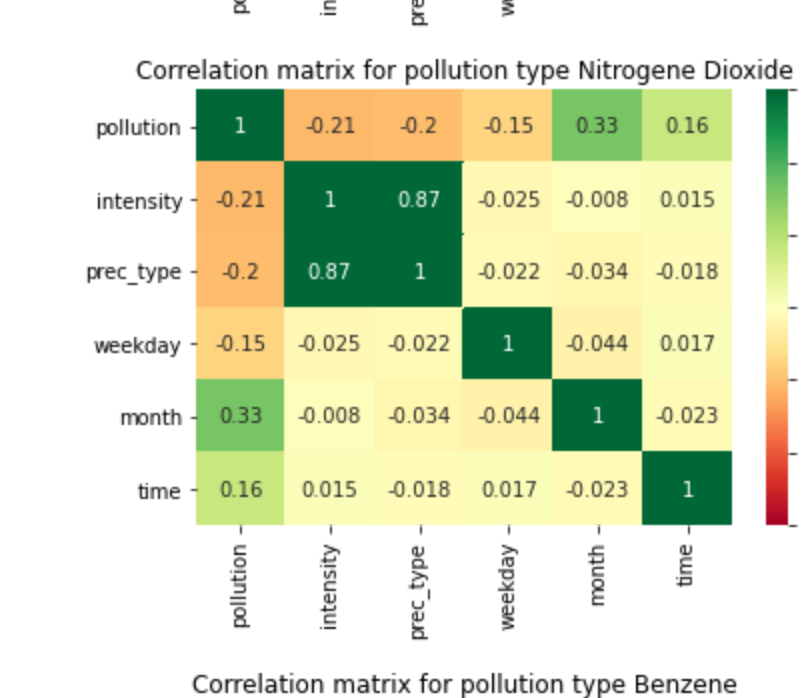
[37827 rows x 7 columns]

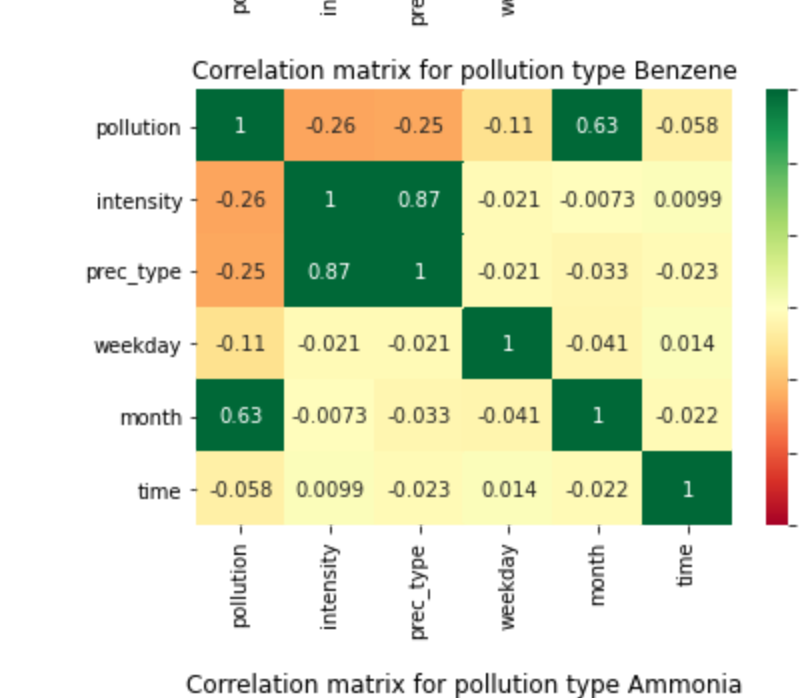
Make correlation matrix for each pollution type and find if precipitation has correlation to pollution

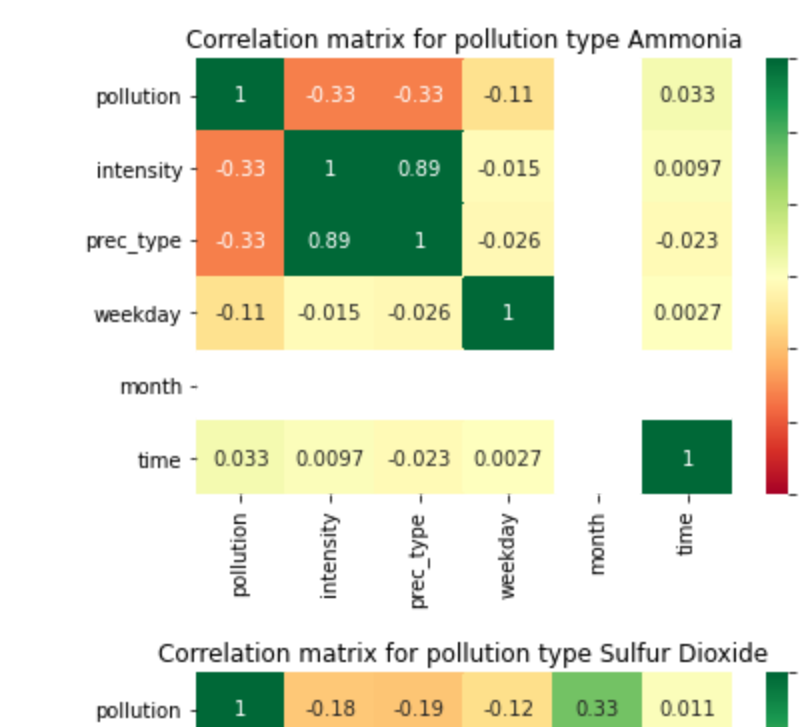
```
# Convert to dataframe and drop coordinates
df_milano = pd.DataFrame(milano)

# Ozone is in the data with names Ozone and Ozono. Rename Ozono to be Ozone.
df_milano = df_milano.replace({'pollution_type': "Ozono"}, "Ozone")

# Make list of pollution types
pol_types = list(df_milano["pollution_type"].unique())

# Loop throught pollution types and make correlation matrix for each type
for pol_type in pol_types:
    # Make tmp dataframe with given pollution type
    df_tmp = df_milano[df_milano["pollution_type"] == pol_type]
    df_tmp = df_tmp.drop(columns=["pollution_type"])
    # Make correlation matrix
    corrmat = df_tmp.corr()
    corr_features = corrmat.index
    plt.figure(figsize=(6,4))
    plt.title("Correlation matrix for pollution type {}". format(pol_type))
    g=sns.heatmap(df_tmp[corr_features].corr(), annot=True, cmap="RdYlGn", vmin=-0.6,
```

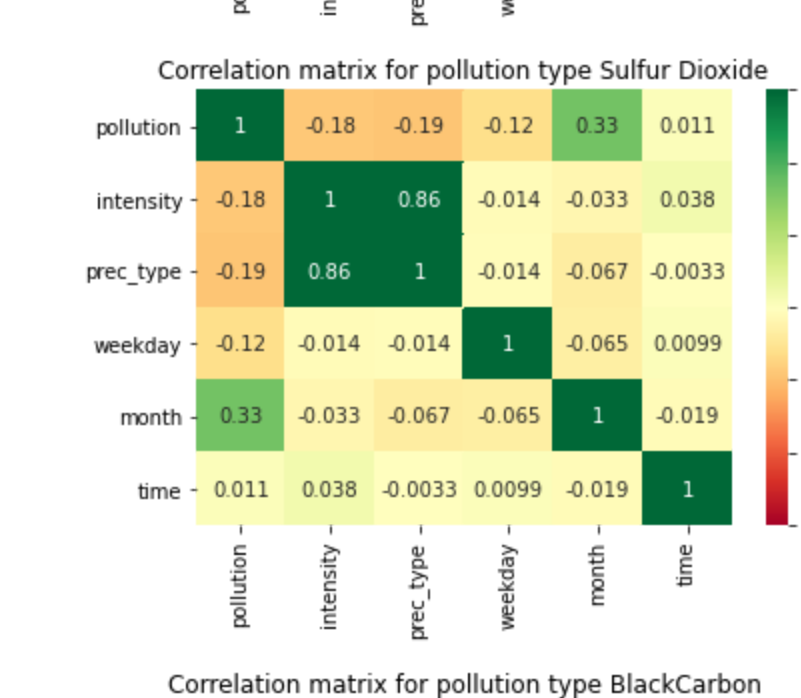


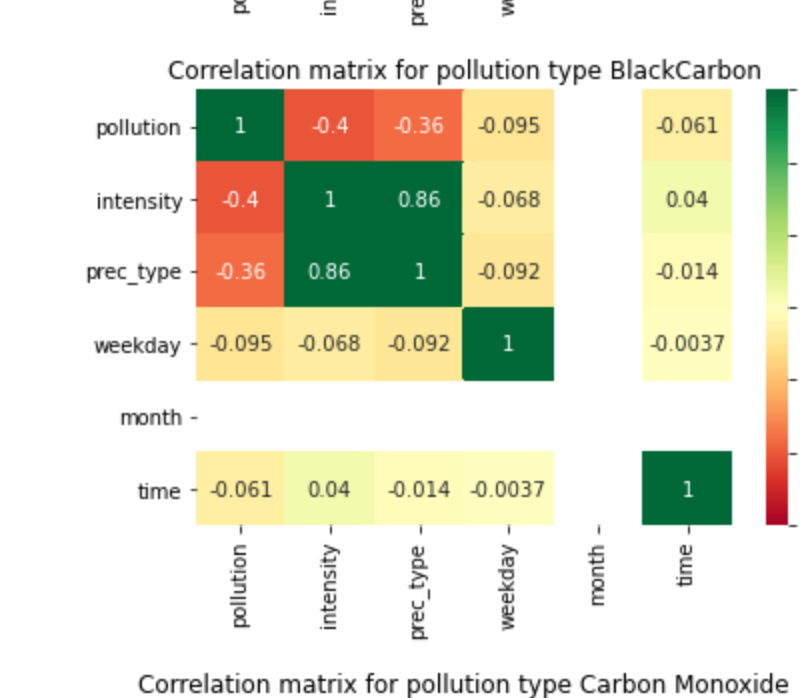


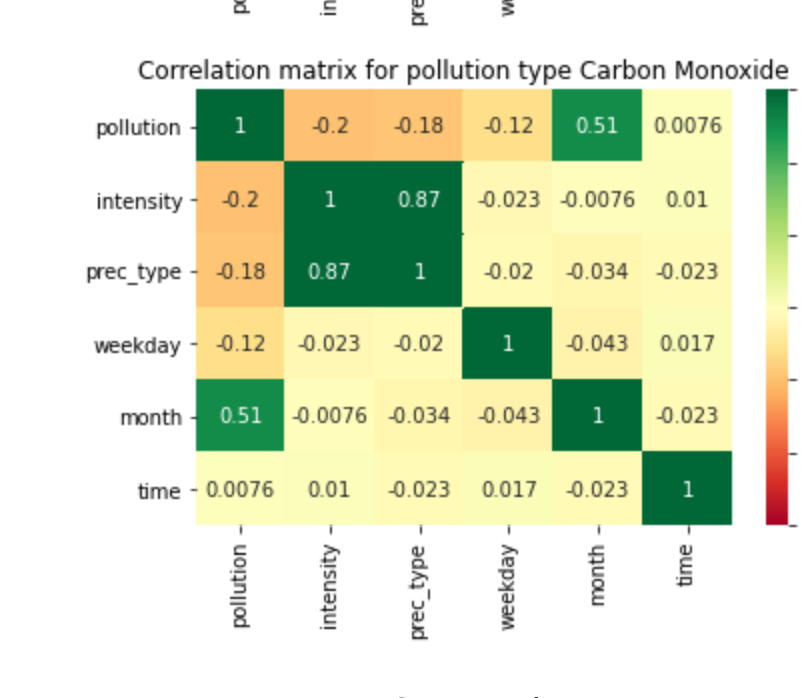












Answer to research question 7:

Our main question was if precipitation intensity or precipitation type affects the amount of pollution in the air. It does look like there is a correlation between precipitation and corruption, but the strength of the correlation depends on the pollution type. In every case, the precipitation intensity had a higher correlation to pollution than the precipitation type, but in most cases, both had a significant correlation to the amount of pollution.

The strongest correlation was found from the black carbon (corr = -0.4). The correlation was negative, which means that when rain intensity rises, the amount of black carbon in the air reduces.

For most of the pollution types, the correlation between pollution and precipitation intensity varied between -0.18 and -0.4. Ozone was the only exception, and it had a positive correlation of 0.1 to the precipitation intensity. Correlation can not be considered to be strong, but after I did some research on this topic, it confirmed this to be common phenomenon.

All the numbers are presented in the heatmap correlation matrices above.