PROJECT
DYNAMIC COVID-19 SPREADING CONTROL VIA PERMISSIVE POPULATION MOBILITY POLICIES

MODULE
MOBILITY MODEL

PROJECT CONTRIBUTORS
Fernando Rios
Hassan Ouanir
Ian Kavuma
Adish

Purpose:
The purpose of this project is avoid spread of the COVID19 by analyze the mobility data. By identify group of people being affected with various criticality and controlling their transportability to gather with much affected location. The module mobility model purpose is generating a matrix that let us identify the trips from - to each area that were pre-divided in a grid of 25 rows and 100 columns.

To achieve this goal the module mobility model was divided in four parts to explain clearly:

Part I: **Data pre-processing and exploratory data analysis (EDA)**
- Importing libraries and loading the data
- Data validation, and
- Data describe

Part II: **Creating general matrix pickup vs dropoff**

Part III: Creating matrix pickup vs dropoff by hours
Part IV: Data visualization

Part I. Data pre-processing and Exploratory Data Analysis (EDA)

To run the notebook just modify the variable "path_csv" with the path of the train dataset in your google colab.

In this section we import libraries to perform the associated task, we used pandas to load data file and data manipulation, matplotlib and seaborn to plot heat map of distribution data and numpy to handle multi-dimensional arrays.

 # Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

 # Loading training dataset

In this section we have load csv data file into "df_train" dataset using pandas read_csv command. Dataset df_train contains total of 3414443 rows and 6 columns.

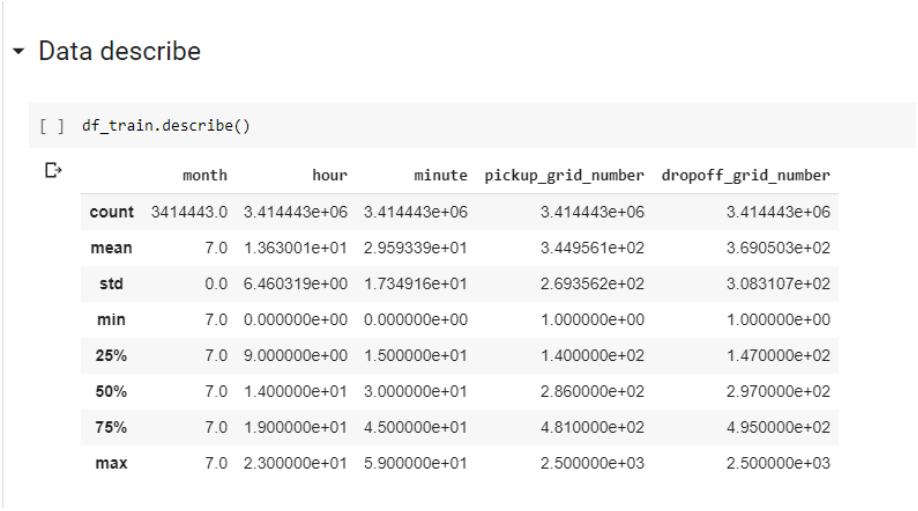
```
# Path of training data file in Google Drive (each group will modify the path of its traininig file)
path_csv = "/content/drive/My Drive/Notebooks/Mobility/training_data.csv"

# This function mount google colab and, read training file csv, returns a dataframe
def ds_from_googlecolab(path_csv):
  from google.colab import drive
  drive.mount('/content/drive')
  ds = pd.read_csv(path_csv)
  return ds

# Called to the function above mentioned to bring the training dataframe with the 5 first rows.
df_train = ds_from_googlecolab(path_csv)
df_train.head()

# Understanding the shape of the training dataset
 print("Rows:", df_train.shape[0], "Columns:", df_train.shape[1])
```

 # Data Validation

```
# Finding empty values over the columns used in the project.
def num_of_empty(ds):
  ds_empty= ds.loc[(ds['pickup_grid_number']=='') | (ds['dropoff_grid_number']=='') |
(ds['hour']=='')]
  return ds_empty.shape[0]

num_row_empty = num_of_empty(df_train)

print("Number of row with empty values: ", num_row_empty)
```

 # Data describe

df_train.describe()



▾ Data describe

```
[ ]  df_train.describe()
```

	month	hour	minute	pickup_grid_number	dropoff_grid_number
count	3414443.0	3.414443e+06	3.414443e+06	3.414443e+06	3.414443e+06
mean	7.0	1.363001e+01	2.959339e+01	3.449561e+02	3.690503e+02
std	0.0	6.460319e+00	1.734916e+01	2.693562e+02	3.083107e+02
min	7.0	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00
25%	7.0	9.000000e+00	1.500000e+01	1.400000e+02	1.470000e+02
50%	7.0	1.400000e+01	3.000000e+01	2.860000e+02	2.970000e+02
75%	7.0	1.900000e+01	4.500000e+01	4.810000e+02	4.950000e+02
max	7.0	2.300000e+01	5.900000e+01	2.500000e+03	2.500000e+03

We can observe in the table, the dataset has trips just of month 7. The feature hour looks well, the minimum is 0 and maximum is 23 hour. Minutes looks good as well, from 0 minutes to 59 minutes. Pickup has trips from zone 0 to zone 2500, clearly skewed right with a mean of zone 344. Dropoff has trips from zone 0 to 2500, clearly skewed right as well with a mean of zone 369.

We have also observed that data provided to us has no missing values. All dataset's format was aligned.

Part II. Creating general matrix pickup vs dropoff

 # General Matrix

This matrix generates the number of trips by zones

```
matrix = df_train.groupby(['pickup_grid_number','dropoff_grid_number']).size().unstack().fillna(0)
```

```
matrix
```

Part II. Creating general matrix pickup vs dropoff

General Matrix

This matrix generates the number of trips by zones

```
[ ] matrix = df_train.groupby(['pickup_grid_number','dropoff_grid_number']).size().unstack().fillna(0)
    matrix
```

dropoff_grid_number	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0	16.0	17.0	18.0	19.0	20.0	21.0	22.0	23.0	24.0	25.0	26.0	27.0	28.0	29.0	30.0	31
pickup_grid_number																															
1.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
2.0	0.0	17.0	5.0	1.0	1.0	3.0	4.0	10.0	2.0	4.0	0.0	3.0	1.0	5.0	3.0	1.0	0.0	8.0	0.0	2.0	3.0	2.0	1.0	3.0	2.0	4.0	5.0	5.0	0.0	2.0	
3.0	1.0	2.0	57.0	14.0	14.0	19.0	21.0	42.0	22.0	44.0	20.0	54.0	21.0	31.0	13.0	21.0	9.0	22.0	11.0	34.0	17.0	15.0	10.0	52.0	29.0	42.0	18.0	30.0	7.0	34.0	
4.0	1.0	3.0	14.0	59.0	10.0	10.0	17.0	36.0	28.0	58.0	18.0	55.0	36.0	33.0	15.0	18.0	5.0	20.0	6.0	31.0	17.0	23.0	20.0	53.0	43.0	65.0	14.0	37.0	9.0	40.0	
5.0	0.0	3.0	10.0	6.0	36.0	17.0	4.0	22.0	18.0	33.0	9.0	45.0	43.0	24.0	11.0	10.0	6.0	12.0	0.0	25.0	9.0	22.0	10.0	34.0	16.0	35.0	12.0	21.0	4.0	21.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2400.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2425.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2450.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2475.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2500.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

957 rows × 979 columns

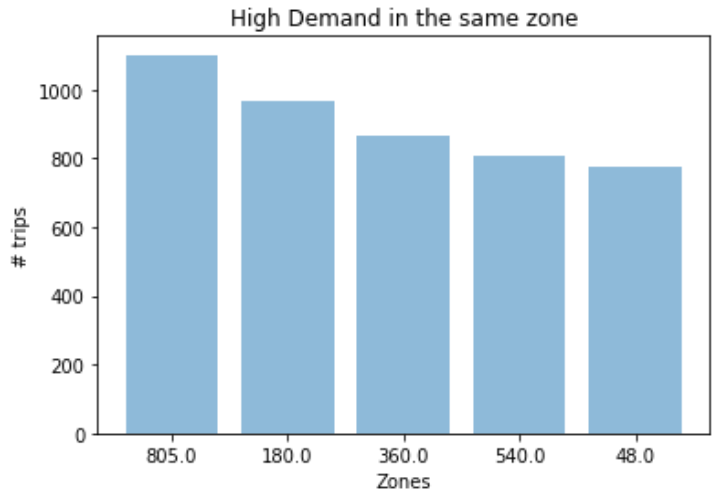
```
# Understanding shape of the matrix
matrix.shape
```

We find 957 pickup zones by 979 dropoff zones used in the all trips

```
#Calculations about trips in the same area
# This function returns a tuple with the trips from a zone to the same zone.
def same_zone(matriz):
  res = []
  v_cols= matriz.columns
  v_rows= matriz.index
  for i in v_cols:
    for j in v_rows:
      if (i==j):
        res.append((i,matriz[i][j]))
  return (res)

# Calling the function same_zone given the matrix.
resul = same_zone(matrix)
resul

# This function searches in the tuple the area with the most trips.
def Max_SameZone(tuple):
        return max(tuple,key=lambda zone:zone[1])
```

```
# Calling to the function Max_SameZone
print("Zone:", Max_SameZone(resul)[0], "Number of trips:", Max_SameZone(resul)[1])

 # This function searches in the tuple the area with the fewest trips.
def Min_SameZone(tuple):
        return min(tuple,key=lambda zone:zone[1])

# Calling to the function Min_SameZone
print("Zone:", Min_SameZone(resul)[0], "Number of trips:", Min_SameZone(resul)[1])

 # This function sort the tuple ascendant or descendent
def Sort_Tuple(tup,asc):
  if type(asc)==bool:
    if asc==True:
      return(sorted(tup, key = lambda x: x[1]))
    else:
      return(sorted(tup, key = lambda x: -x[1]))
  else:
    print("Parameter asc should be boolean")

  # Printing the tuple ordered
 print(Sort_Tuple(resul,True)) # False is descendent , True is ascendent



# Getting 5 high demand zones, picking and dropping in the same zone.
FiveTuple=Sort_Tuple(resul,False)[:5]
FiveTuple

# This functions gets only the zones
def Get_Zones(tuple):
  return [tuple_[0] for tuple_ in FiveTuple]

# This functions gets only the number of trips
def Get_NumTrips(tuple):
  return [tuple_[1] for tuple_ in FiveTuple]
```

```
# Retriving Zones and values of the 5 high demand zones, picking and dropping in the same zone.
Zones=Get_Zones(FiveTuple)
Values=Get_NumTrips(FiveTuple)
Zones, Values

# Function that create a barchart with the high demand in the same zone
def barchart(tuple):
  objects = Get_Zones(FiveTuple)
  y_pos = np.arange(len(objects))
  performance = Get_NumTrips(FiveTuple)
 plt.bar(y_pos, performance, align='center', alpha=0.5)
  plt.xticks(y_pos, objects)
  plt.xlabel('Zones')
  plt.ylabel('# trips')
  plt.title('High Demand in the same zone')
  plt.show()

 # Tuple ordered to show it in the barchart
FiveGreatest=Sort_Tuple(resul,False)[:5]
FiveGreatest

 # Calling to the function barchart to plot the 5 zones with high demand in trips, from / to same zone.
barchart(FiveGreatest)
```



The above graphic show us that the zone 805. People use this zone to pick up and dropoff.

Total de trips given the area

```
# All pickup trips given area
def pickup_by_zone(matriz,zone):
  sum_zone =matriz[matriz.index==zone].sum(axis=1)
  return sum_zone.values[0]

# Example: How many pickup trips were in the zone 805
pickup_by_zone(matrix,805)
```

```
1318.0
```

This makes sense, because in the barchart showed lines above, we saw that zone 805 is the most used from / to same zone, this means that the difference between 1318 and 1102 are 216 trips, that were picked up from the zone 805 but the droppoff was in a different zone.

```
# All dropoff trips given zone
def droppoff_by_zone(matriz,zone):
  return matriz[zone].sum()

# Example: How many dropoff trips were in the zone 2500
droppoff_by_zone(matrix,2500)

# All trips from / to zone
def total_zone(matriz,zonepickup, zonedropoff):
  total = pickup_by_zone(matriz,zonepickup)+droppoff_by_zone(matriz,zonedropoff)
  return total

# Calculating all the trips that were picked up in the zone 1 and all the trips droppoff in the zone 1
total_zone(matrix,1,1)
```

Part III. Creating matrix pickup vs dropoff by hours

General Matrix by hour

```
matrizbyhours = df_train.groupby(['hour','pickup_grid_number','dropoff_grid_number']).size().unstack().fillna(0)
matrizbyhours
```

Part III. Creating matrix pickup vs dropoff by hours

General Matrix by hour

```
[ ] matrizbyhours = df_train.groupby(['hour','pickup_grid_number','dropoff_grid_number']).size().unstack().fillna(0)
    matrizbyhours
```

	dropoff_grid_number	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0	16.0	17.0	18.0	19.0	20.0	21.0	22.0	23.0	24.0	25.0	26.0	27.0	28.0	29.0	30.0
hour	pickup_grid_number																														
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	2.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
	3.0	0.0	0.0	4.0	0.0	1.0	0.0	0.0	1.0	0.0	3.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2.0	2.0	0.0	1.0	0.0	1.0
	4.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	2.0	3.0	2.0	0.0	1.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	3.0	4.0	3.0	0.0	1.0	0.0	0.0
	5.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	2.0	1.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23	2232.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	2256.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	2275.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	2350.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	2450.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

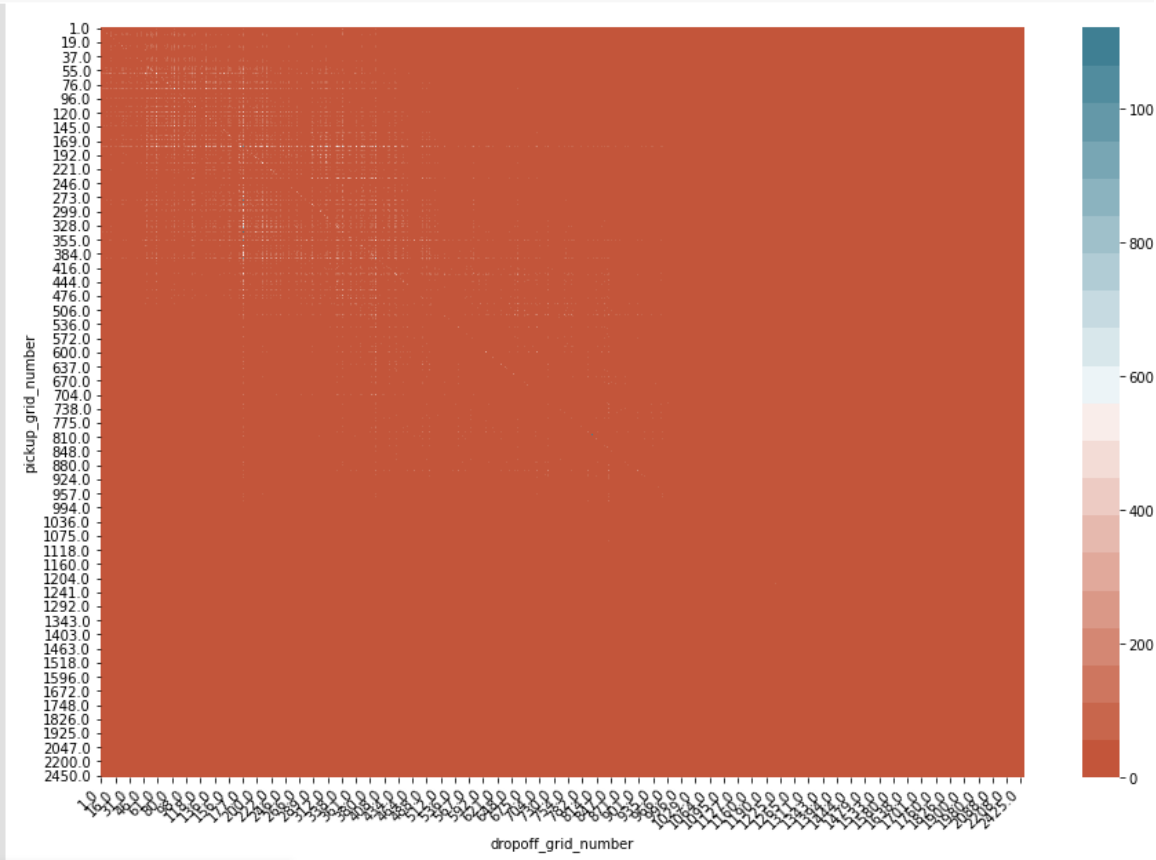
16738 rows × 979 columns

Part IV. Data visualization

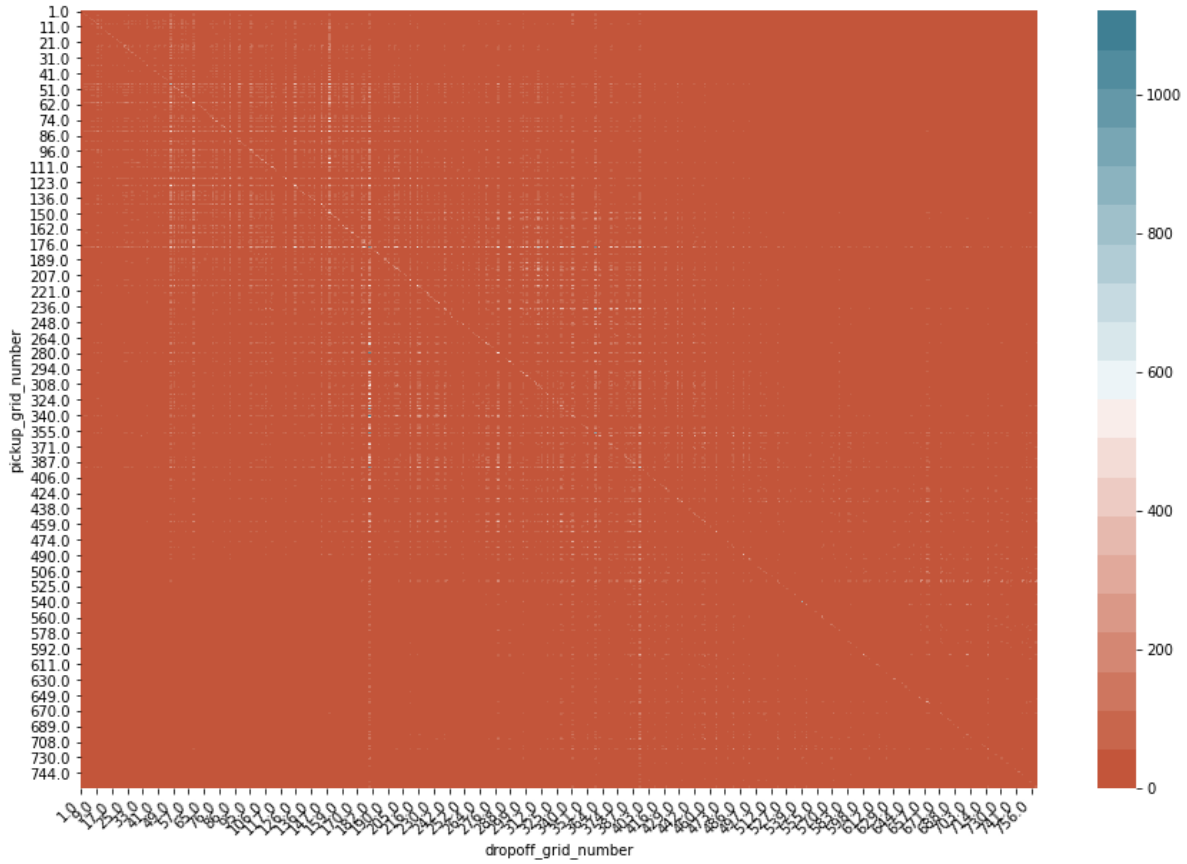
 # Heatmap to visualize the zones of pickup and dropoff aggregated

```
# This function creates a heatmap to visualize the zones of pickup and dropoff, we give the matrix, width and height of the heatmap.
def plot(matriz,width,height):
  fig, ax = plt.subplots(figsize=(width,height))
  ax = sns.heatmap(matriz,
      vmin=0, vmax=max(matriz.max()),
      cmap=sns.diverging_palette(20, 220, n=20),
      square=False,
  )
  ax.set_xticklabels(
      ax.get_xticklabels(),
      rotation=45,
      horizontalalignment='right'
  );
plot(matrix,15,10)
```

```
plot(matrix,15,10)
```



# Zoom to visualize better, we used 500 records in accordance with 75% (third quartile)
plot(matrix.iloc[0:500,0:500],15,10)

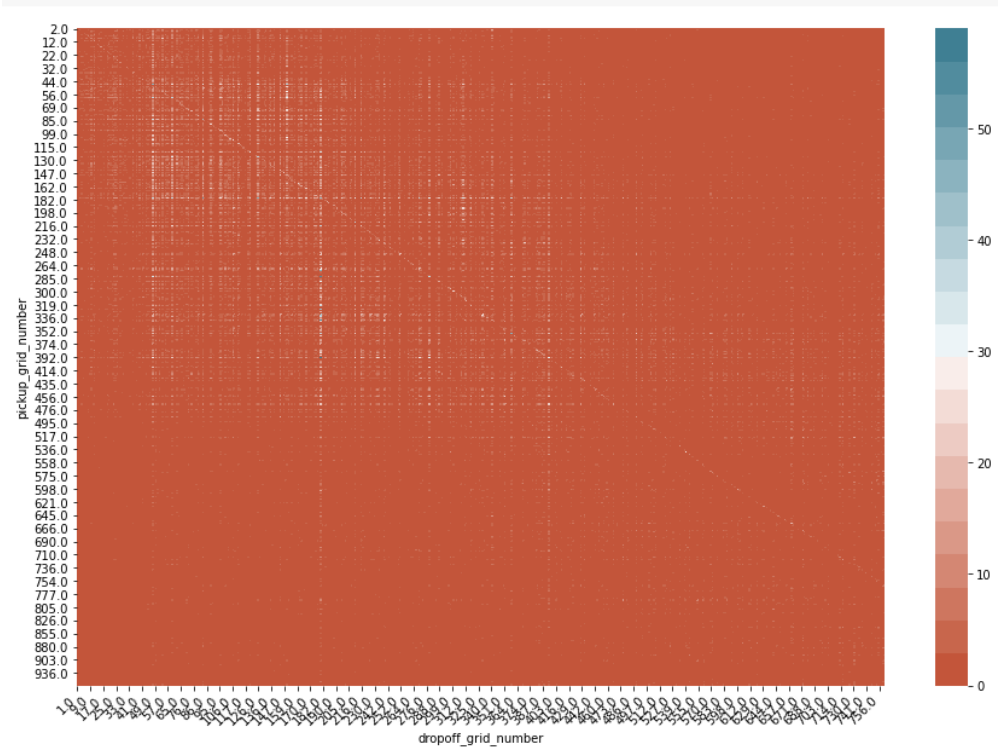


This heatmap show us many things interesting like: there are strong lines on the Y axis, one of them is on the pickup number 180, this makes sense because 180 is a zone with many trips according to the graph showed lines above. Clearly, we can see a diagonal line, showing that many trips were picked up and dropped off in the same zone.

#Heatmap to visualize the zones of pick up and dropoff by hours

```
# Function filter matrix given the hour
def trips_by_hour(matriz,hour): # hours [0 to 23]
  if ((hour<0) | (hour>23)): # returns true if all digits otherwise false
    return print("Range of months is [0,23]")
  else:
    mf = matriz.loc[hour]
    return mf

 # Plotting matrix of the hour 20, 8PM
trips_in_hour=trips_by_hour(matrizbyhours,20)
```

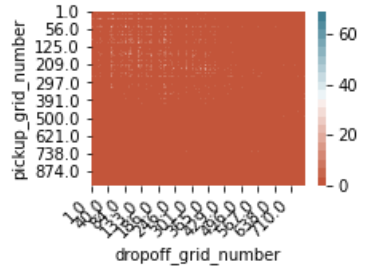
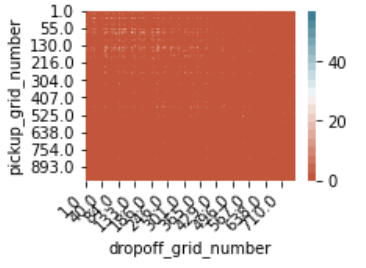
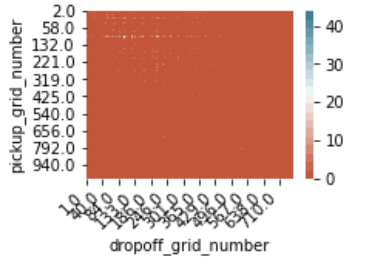
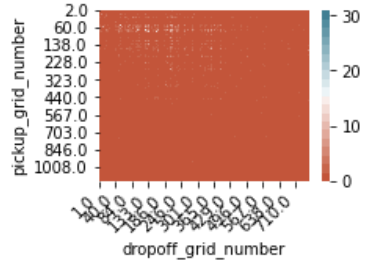
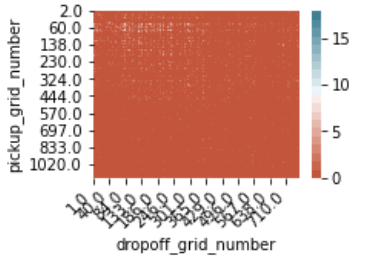
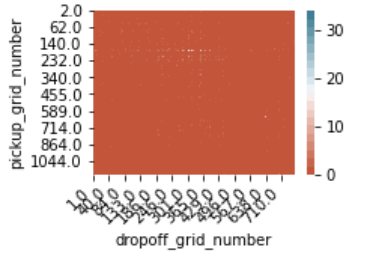
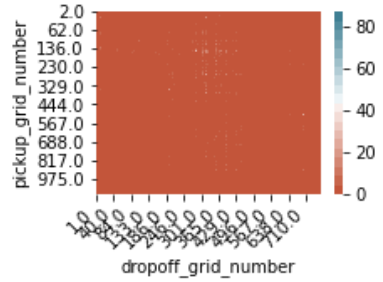
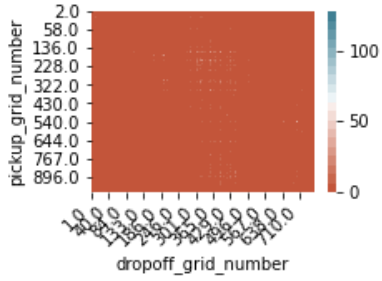
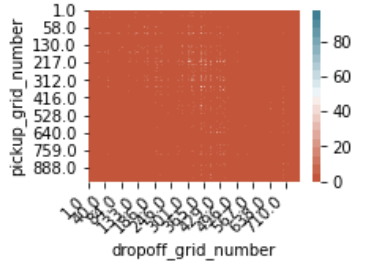
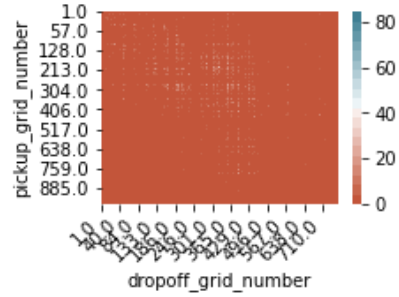
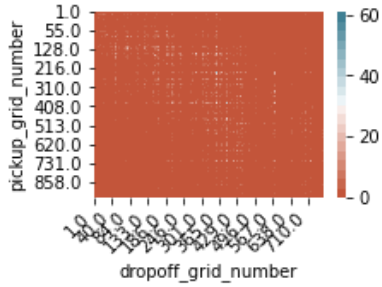
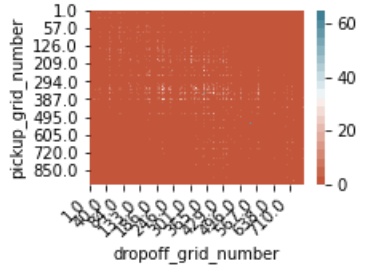
```
 # zoom plotting 500 first rows
plot(trips_in_hour.iloc[:500,:500],15,10)
```

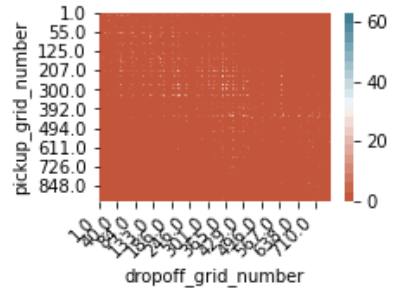
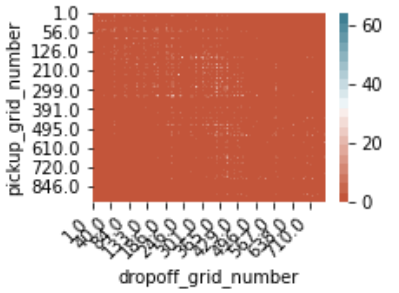
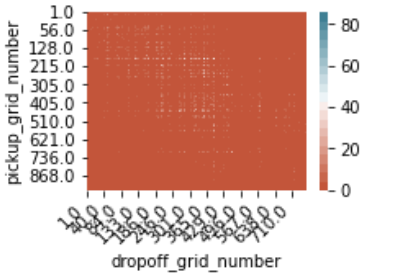
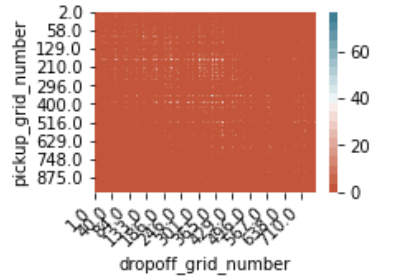
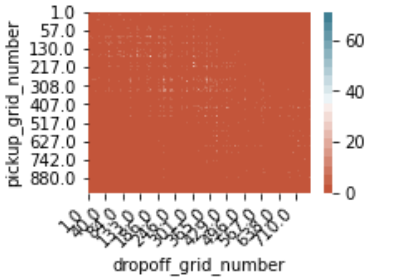
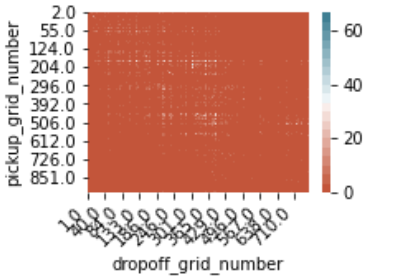
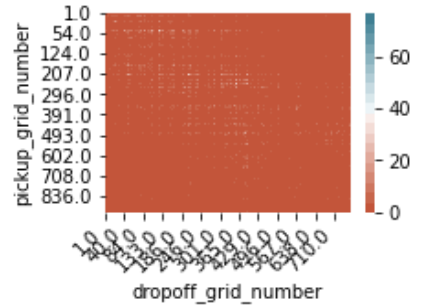
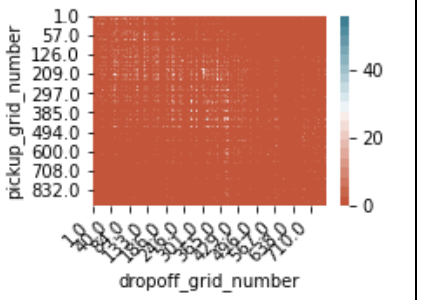
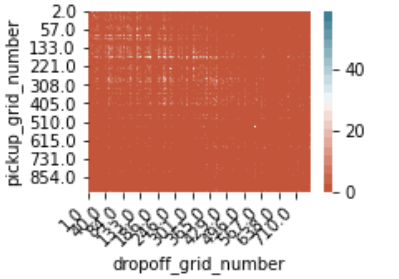


 # Heatmap for each hour

Plotting matrix by each hour from 1 hour to 23 hour

```
for i in range(24):
    trips_in_hour=trips_by_hour(matrizbyhours,i)
    plot(trips_in_hour.iloc[:500,:500],3,2)
```

Hour 0	Hour 1	Hour 2
		
Hour 3	Hour 4	Hour 5
		
Hour 6	Hour 7	Hour 8
		
Hour 9	Hour 10	Hour 11
		

Hour 12	Hour 13	Hour 14
		
Hour 15	**Hour 16**	**Hour 17**
		
Hour 18	**Hour 19**	**Hour 20**
		
Hour 21	**Hour 22**	**Hour 23**