Contents

# 1 Executive Summary

In this notebook I aim to create a data processing pipeline using Apache Spark with Dataproc on Google Cloud Platform. It is a common use case in data science to read data from one storage location (in our case Public Data available on Google cloud), perform transformations on it and write it into another storage location. I will explain how to connect pyspark to BigQuery, get the big data and transform it to a dataframe save it as a CSV file and perform a nice visualization using **"folium"** and **"geopy"** Libraries.

# 2 Introduction

I plan to build a heatmap for the number of identified Covid19 cases in Italy during '2020-02-24' ~'2020-09-25'. This data is publicly available in google cloud but we aim to have access to that through pyspark. I aim to visualze my findings a a heatmap on the map of Italy.

# 3 Create a Dataproc Cluster

First I create a Dataproc Cluster and I install and run a Jupyter notebook on it. I also enabled the required APIs such as BigQuery API, BigQuery Storage API, etc https://cloud.google.com/dataproc/docs/tutorials/jupyter-notebook (https://cloud.google.com/dataproc/docs/tutorials/jupyter-notebook) https://medium.com/google-cloud/apache-spark-and-jupyter-notebooks-made-easy-with-dataproc-component-gateway-fa91d48d6a5a (https://medium.com/google-cloud/apache-spark-and-jupyter-notebooks-made-easy-with-dataproc-component-gateway-fa91d48d6a5a)

# 4 Using Spark-bigquery-connector

Python 3 kernel within my Dataproc Cluster allow me to configure the SparkSession in my notebook . I only need to include the spark-bigquery-connector required to use the BigQuery Storage API.

# 5 Setup a Session

In [ ]:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
  .appName('Myapp')\
  .config('spark.jars', 'gs://spark-lib/bigquery/spark-bigquery-latest_2.12.jar') \
  .getOrCreate()
```

# 6 Get the Data as a Dataframe

In [ ]:

```
table = "bigquery-public-data.covid19_italy.data_by_province"
df = spark.read \
  .format("bigquery") \
  .option("table", table) \
  .load()
df.printSchema()
```

In [ ]:

```
# Creating two dataframes
df1=df \
  .select('province_code', 'latitude', 'longitude')
df_agg = df \
  .select('province_code', 'province_name', 'confirmed_cases', 'latitude', 'longitude', 'date') \
  .where("date > '2020-02-24'") \
  .groupBy('province_code') \
  .sum('confirmed_cases')
```

In [ ]:

```
# Removing the duplicate records out of the dataframes and joining them
df1 = df1.drop_duplicates(subset=['province_code'])
inner_join = df1.join(df_agg, df1.province_code == df_agg.province_code)
inner_join = inner_join.drop_duplicates(subset=['latitude', 'longitude'])
```

In [ ]:

```
#Saveing the result
inner_join.toPandas().to_csv(r'export_dataframe.csv', index = False, header=True)
```

At this point our dataframe is saved on the cloud and we can download it easily to our local system

# 7 Load Libraries

In [22]:

```
import pandas as pd
import numpy as np
import folium
import requests
from geopy.geocoders import Nominatim
from pandas.io.json import json_normalize
```

Folium is a library for creating interactive maps.

# 8 Read the Data

In [79]:
```
df_train = pd.read_csv('Downloads/export_dataframe.csv')
df_train
```

Out[79]:

| | province_code | province_name | latitude | longitude | province_code.1 | sum(confirmed_cases) |
|---|---|---|---|---|---|---|
| **0** | 96 | Biella | 45.566511 | 8.054082 | 96 | 182372 |
| **1** | 23 | Verona | 45.438390 | 10.993527 | 23 | 912293 |
| **2** | 87 | Catania | 37.502878 | 15.087047 | 87 | 174827 |
| **3** | 74 | Brindisi | 40.638485 | 17.946016 | 74 | 113081 |
| **4** | 99 | Rimini | 44.060901 | 12.565630 | 99 | 400671 |
| **...** | ... | ... | ... | ... | ... | ... |
| **103** | 61 | Caserta | 41.074659 | 14.332405 | 61 | 114684 |
| **104** | 84 | Agrigento | 37.309711 | 13.584575 | 84 | 30160 |
| **105** | 28 | Padova | 45.406930 | 11.876087 | 28 | 756088 |
| **106** | 81 | Trapani | 38.018501 | 12.513657 | 81 | 30183 |
| **107** | 71 | Foggia | 41.462269 | 15.543051 | 71 | 211754 |

108 rows × 6 columns

In [80]:
```
df_train.isnull().sum()
```

Out[80]:
```
province_code           0
province_name           0
latitude                1
longitude               1
province_code.1         0
sum(confirmed_cases)    0
dtype: int64
```

In [81]:
```
df_train=df_train.dropna()
```

In [82]:
```
df_train
```

Out[82]:

| | province_code | province_name | latitude | longitude | province_code.1 | sum(confirmed_cases) |
|---|---|---|---|---|---|---|
| **0** | 96 | Biella | 45.566511 | 8.054082 | 96 | 182372 |
| **1** | 23 | Verona | 45.438390 | 10.993527 | 23 | 912293 |
| **2** | 87 | Catania | 37.502878 | 15.087047 | 87 | 174827 |
| **3** | 74 | Brindisi | 40.638485 | 17.946016 | 74 | 113081 |
| **4** | 99 | Rimini | 44.060901 | 12.565630 | 99 | 400671 |
| **...** | ... | ... | ... | ... | ... | ... |
| **103** | 61 | Caserta | 41.074659 | 14.332405 | 61 | 114684 |
| **104** | 84 | Agrigento | 37.309711 | 13.584575 | 84 | 30160 |
| **105** | 28 | Padova | 45.406930 | 11.876087 | 28 | 756088 |
| **106** | 81 | Trapani | 38.018501 | 12.513657 | 81 | 30183 |
| **107** | 71 | Foggia | 41.462269 | 15.543051 | 71 | 211754 |

107 rows × 6 columns

In [83]:
```
df_train= df_train.drop("province_code.1", axis= 1)
```

It seems through we have access to 107 province in Italy only.

## 9 EDA

In [86]:
```
# let us look at the top 20 in the list
df_train.sort_values('sum(confirmed_cases)', ascending=False).head(20)
```

Out[86]:

| | province_code | province_name | latitude | longitude | sum(confirmed_cases) |
|---|---|---|---|---|---|
| **94** | 15 | Milano | 45.466794 | 9.190347 | 4169004 |
| **75** | 17 | Brescia | 45.539931 | 10.219103 | 2742099 |
| **89** | 1 | Torino | 45.073274 | 7.680687 | 2657644 |
| **41** | 16 | Bergamo | 45.694414 | 9.668425 | 2553422 |
| **72** | 19 | Cremona | 45.133367 | 10.024209 | 1193419 |
| **77** | 58 | Roma | 41.892770 | 12.483667 | 1126933 |
| **52** | 108 | Monza e della Brianza | 45.584390 | 9.273582 | 994770 |
| **79** | 10 | Genova | 44.411493 | 8.932699 | 951271 |
| **51** | 18 | Pavia | 45.185093 | 9.160157 | 942463 |
| **68** | 37 | Bologna | 44.494367 | 11.341721 | 928093 |
| **1** | 23 | Verona | 45.438390 | 10.993527 | 912293 |
| **73** | 35 | Reggio nell'Emilia | 44.697353 | 10.630080 | 890031 |
| **95** | 22 | Trento | 46.068935 | 11.121231 | 828280 |
| **28** | 33 | Piacenza | 45.051935 | 9.692633 | 807185 |
| **105** | 28 | Padova | 45.406930 | 11.876087 | 756088 |
| **8** | 36 | Modena | 44.646000 | 10.926155 | 719917 |
| **39** | 6 | Alessandria | 44.912974 | 8.615401 | 686924 |
| **14** | 13 | Como | 45.809991 | 9.085160 | 679186 |
| **27** | 34 | Parma | 44.801074 | 10.328350 | 653653 |
| **61** | 98 | Lodi | 45.314407 | 9.503721 | 647071 |

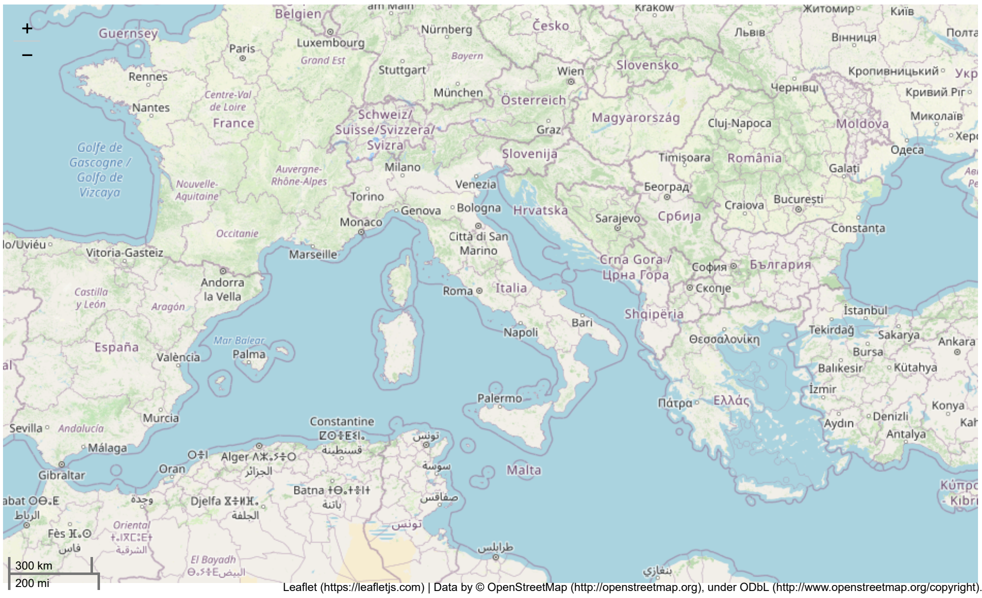
## 10 Create a map of Italy

In [75]:
```
def generateBaseMap(default_location=[41.90322, 12.49564], default_zoom_start=5):
    base_map = folium.Map(location=default_location, control_scale=True, zoom_start=default_zoom_start)
    return base_map
```

Contents

In [76]:
```
baseMap = generateBaseMap()
baseMap
```

Out[76]:



Leaflet (https://leafletjs.com) | Data by © OpenStreetMap (http://openstreetmap.org), under ODbL (http://www.openstreetmap.org/copyright).

## 11 Create a heatmap

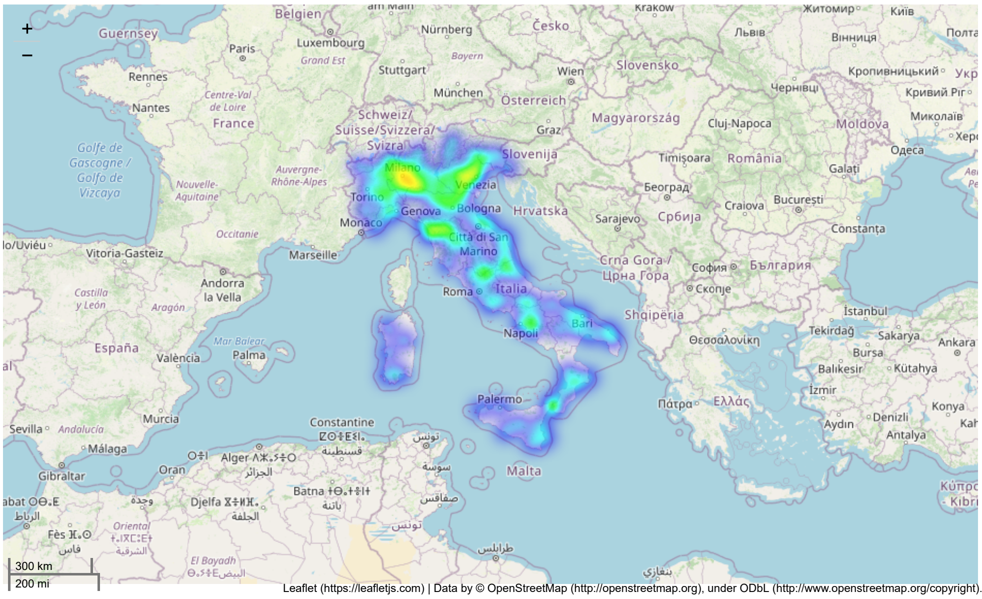
In [69]:
```
from folium.plugins import HeatMap
```

In [77]:
```
base_map = generateBaseMap()
HeatMap(data=df_train[["latitude","longitude","sum(confirmed_cases)"]].values.tolist(),
        radius=8, max_zoom=5).add_to(base_map)
```

Out[77]:
```
<folium.plugins.heat_map.HeatMap at 0x1a44df3cb88>
```

In [78]:
```
base_map
```

Out[78]:



Leaflet (https://leafletjs.com) | Data by © OpenStreetMap (http://openstreetmap.org), under ODbL (http://www.openstreetmap.org/copyright).

## 12 Conclusion

We did not have access to the region information, but our visualization for sure helps us to see the number of positive cases are outnumbered in the north part of Italy.