IBM Training

Student Exercises

(Machine Learning and Optimization)

Lab-4: Forecast COVID-19 outbreaks – **find optimal ways to** transfer sick patients

By Alain Chabrier
Lee Angelelli

Legal Copyright: *© Copyright IBM Corp. 2020*
Course materials may not be reproduced in whole or in part without the prior written permission of IBM

Table of Contents

Introduction

This lab will introduce users to the use of IBM's predictive analytics and decision optimization technologies to solve COVID-19 problems. The coronavirus has infected millions of people leading to severe illness symptoms resulting in hundreds of thousands deaths. During the initial outbreak not all areas were affected the same. Hospitals located in COVID-19 epidemic outbreak locations were overwhelmed with sick and dying patients.

This lab will apply predictive analytics to analyze different factors among people to predict future COVID-19 infection rates in an area. Based on areas predicted to have high COVID-19 infections – this lab will apply optimization techniques to create optimized plans for transferring COVID-19 patients from hospitals located in epidemic areas to hospitals with less COVID-19 patients. Our intention is to educate people who are involved in the COVID-19 response decision-making process, in applying IBM's predictive and optimization technologies to help them improve planning and responding to the next wave of COVID-19 cases.

Objectives

The goal of this lab is to educate attendees on how to apply IBM predictive analytics and optimization tools to different applications of COVID-19 like (1) predicting future infections and (2) optimizing response for better decision making. Students will use Watson Studio to load and step through a notebook that applies Decision Optimization to optimize the transport of affected people between hospitals to avoid being over capacity. Working through this notebook, students will learn these skills.

1. Learn how to load data form different places (departments, current situation, etc.) to be used for analysis.

2. Learn how to represent the current situation on a map using folium. folium makes it easy to visualize data that's been manipulated in Python on an interactive leaflet map.

3. Learn how to use a LinearRegression model from sklearn to predict new cases for each department.

4. Learn how to use Decision Optimization to model and optimize patient transfers.

5. Learn how to use folium to display the optimized future patient transfers plan i.e. all the transfers from the solution.

COVID-19 Use Case - Planning the Patient Transfers

These days, one of the critical problem for our governments is to manage the heterogeneous propagation of the virus over the territories. Not all areas are infected at the same level. While confinement of the population and prohibition of travel is reducing the propagation of the virus, the transfer of sick people between different areas can reduce the stress of hospitals in the most

critical regions. In France, transfers are now generalized, using military planes (see here or here), helicopters or high-speed trains, but also locally with ambulances.

The problem of finding optimal ways to relocate sick people among areas can benefit from a combination of Machine Learning (ML) and Decision Optimization (DO).

Based on the recent evolution of the number of critical intensive care cases in each area, predictive models can be trained to forecast the evolution per area on the coming days. This data, in addition to the capacity of the hospitals for each of the areas and some description of the constraints applying on the possible transfers can then be used in a decision optimization model. This scheme where ML is used first to forecast future COVID-19 infections and DO is used to prescribe the best optimal actions is a very common use case.

You can find here an example notebook.

Import and display input data

We use data from the French government data.gouv.fr site: https://www.data.gouv.fr/fr/datasets/donnees-relatives-a-lepidemie-de-covid-19/#_ in addition to some imported data on the different French administrative "departments" (GPS coordinates of frontier and center).

Using Folium, we can easily represent this data on a map. The circles show the number of intensive care cases in each department. Red circles show above normal capacity.

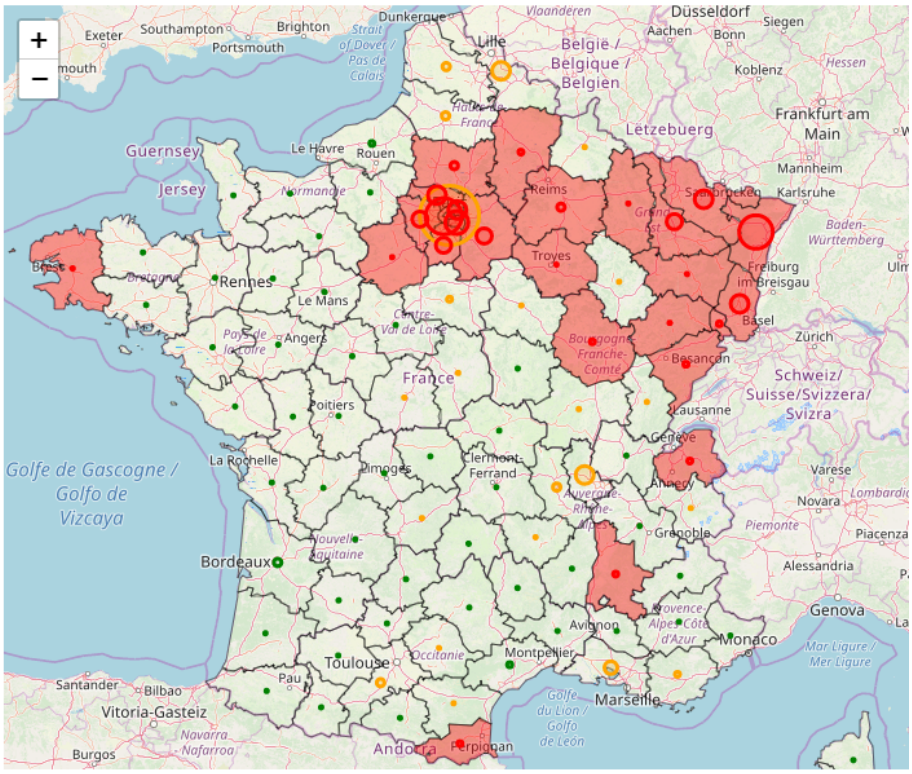


Figure 1 - Current situation

Simplistic predictive model

A very simplistic predictive model is trained for illustration. We use LinearRegression from sklearn, knowing obviously that the epidemy is not linear at all. But this gives an idea of how ML would work here. Excerpt of the LinearRegression model is shown below.

```
import numpy as np
from sklearn.linear_model import LinearRegressionNB_PERIODS = 3def predict_more(d, n):
    X = hosp_data[d].index.tolist()
    X.reverse()
    X=np.array(X).reshape(-1,1)
    y = hosp_data[d]['rea'].tolist()
    y.reverse()
    y=np.array(y).reshape(-1,1)regsr=LinearRegression()
    regsr.fit(X,y)to_predict_x = [i for i in range(len(X), len(X)+n)]
    to_predict_x= np.array(to_predict_x).reshape(-1,1)
    predicted_y= regsr.predict(to_predict_x)
    delta_y = [ int(round(max(0, predicted_y[i][0]-y[len(y)-1][0]))) for i in range(n)]
    return delta_ynew_rea ={ d:predict_more(d, NB_PERIODS) for d in deps}
print (new_rea)
```

The outcome is a prediction of the expected number of new intensive care cases to occur in the next few days we will plan (here `NB_PERIODS = 3`).

Sample Decision Optimization model

Our hypothesis for the optimization model is that two different types of transfers can be done:

- long distance transfers (planes, trains) with the number of transfers limited over the whole country, several people can be transferred at a time.
- short distance transfers (ambulances) with the number of transfers limited per area, and with just one person at a time.

The parameters we use (completely invented) are:

```
MAX_NB_LONG_TRANSFERS_PER_PERIOD = 3
MAX_CASES_PER_LONG_TRANSFERS = 20
MAX_NB_SHORT_TRANSFERS_PER_DEPARTMENT = 3
LONG_DISTANCE = 200
```

We use our `docplex` API to model the problem:

```
from docplex.mp.model import Model
mdl = Model("Plan Transfers")
```

The decision variables represent the transfer links to be used in the best (optimal) transfer plan (an integer variable for the number of persons transferred and a binary variable indicating whether the link is used or not, with a constraint linking both). Another set of auxiliary variables represents the occupancy for each department and each period.

```
use_link_vars = mdl.binary_var_cube(deps, deps, transfer_periods, name="use_link")
link_vars = mdl.integer_var_cube(deps, deps, transfer_periods, lb=0,
ub=MAX_CASES_PER_LONG_TRANSFERS, name="link")
occupancy_vars = mdl.integer_var_matrix(deps, all_periods, lb=0, name="occupancy")
```

Then we formulate all the constraints:

```
# Initial state
mdl.add_constraints(occupancy_vars[d, 0] == initial[d] for d in deps)

# structural constraint between user_link and link
mdl.add_constraints(use_link_vars[d, d1, t] == (link_vars[d, d1, t] >= 1) for d in deps for d1 in deps
for t in transfer_periods)

# Short transfers bounds
mdl.add_constraints(link_vars[d1, d2, t] <= 1 for d1 in deps for d2 in deps if not is_long[d1][d2] for t
in transfer_periods)

# number of transfers from a department less than current number of cases
mdl.add_constraints(mdl.sum(link_vars[d, d1, t] for d1 in deps) <= occupancy_vars[d, t] for d in deps
for t in transfer_periods)

# maximum number of LONG transfers
mdl.add_constraints(mdl.sum(use_link_vars[d1, d2, t] for d1 in deps for d2 in deps if is_long[d1][d2])
<= MAX_NB_LONG_TRANSFERS_PER_PERIOD for t in transfer_periods)

# maximum number of SHORT transfers
mdl.add_constraints(mdl.sum(use_link_vars[d1, d2, t] for d1 in deps if not is_long[d1][d2] for t in
transfer_periods) <= MAX_NB_SHORT_TRANSFERS_PER_DEPARTMENT for d2 in deps )

# conservation constraints including new cases to come
mdl.add_constraints(occupancy_vars[d, t+1] == new_rea[d][t] + occupancy_vars[d, t] +
mdl.sum(link_vars[d1, d, t] for d1 in deps) - mdl.sum(link_vars[d, d1, t] for d1 in deps) for d in deps
for t in transfer_periods)
```

And the objectives. The main objective is to reduce the total overcapacity on the areas, but we should also limit unnecessary transfers.

```
final_overcapacity = mdl.sum(mdl.max(0,  occupancy_vars[d, NB_PERIODS] - capacity[d]) for d in deps) mdl.add_kpi(final_overcapacity)
nb_long_transfers = mdl.sum(use_link_vars[d1, d2, t] for d1 in deps for d2 in deps if is_long[d1][d2] for t in transfer_periods)
mdl.add_kpi(nb_long_transfers)
nb_short_transfers = mdl.sum(use_link_vars[d1, d2, t] for d1 in deps for d2 in deps if not is_long[d1][d2] for t in transfer_periods)
mdl.add_kpi(nb_short_transfers)
mdl.minimize(1000 * final_overcapacity + 10 * nb_long_transfers + nb_short_transfers)
```

We can then solve the problem (using the CPLEX engine) and get the prescribed decisions.

```
mdl.set_time_limit(20)
mdl.solve(log_output=True)
```

Display the solution

Using the same Folium package, we can represent the proposed transfers on a map. The long-distance transfers are represented in green and the short distance ones in black.

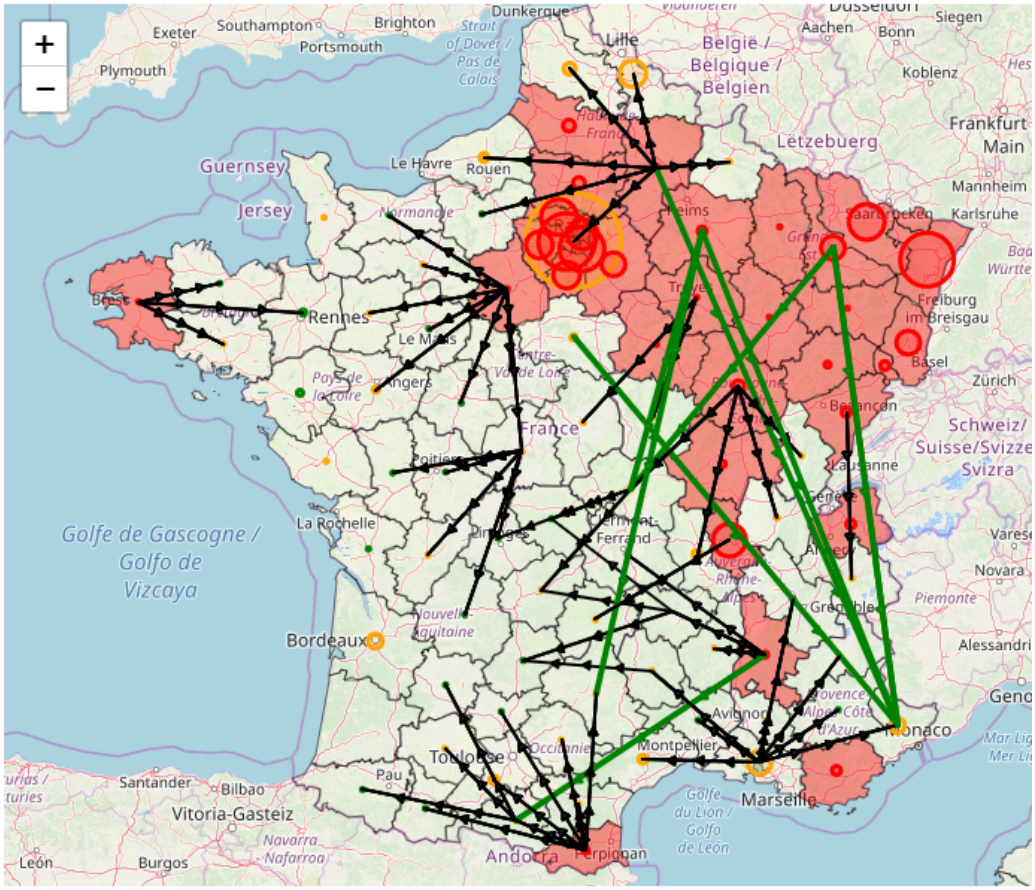


Figure 2 - Proposed optimal solution.

Conclusions

I want to insist again on the fact this is not a ready-to-use solution, but a demonstration of how ML and DO technology could be used to propose mathematically optimal transfer decisions. You can find here the complete notebook.

In practice, additional data, constraints and objectives should be taken into account. For example, we can easily imagine that the level of criticality of individual infected persons would be taken into account. Also, the characteristics of hospitals in the different areas, including the neighborhood of the airport or train station, should be taken into account. Note also my model only considered transfers within France, while some transfers already took place from France to Germany and Luxembourg.

This notebook is therefore only provided as an example which can be used to discover or learn optimization. You can freely try this notebook on Watson Studio Cloud. If you are in a position to use such technology in practice for the Covid-19 situation, don't hesitate to contact us, we are willing to help.

IBM France
Alain Chabrier
STSM - Decision Optimization
IBM Cloud and Cognitive Software
Alain.chabrier@ibm.com

IBM USA
Lee Angelelli
Executive Client Architect
IBM Global Markets
langelel@us.ibm.com

Instructions

Plan transfers between areas

This notebook shows how we can use Decision Optimization to optimize the move of affected people between hospitals to avoid being overcapacity.

The problem is solved per department.

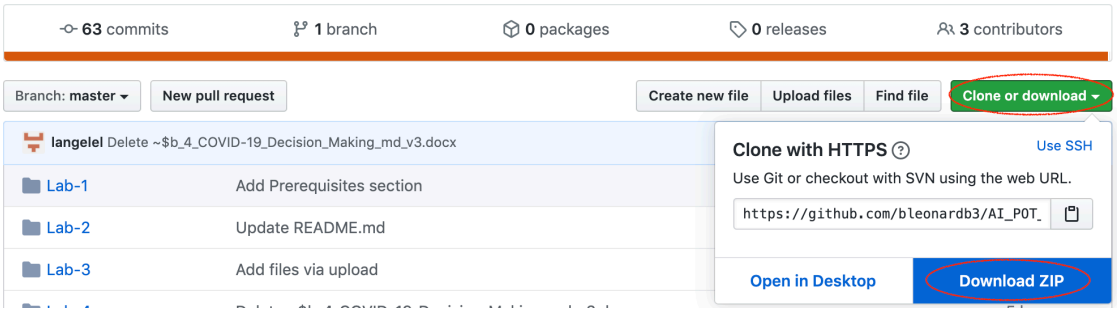
There are 5 parts in this notebooks:

1. Load the data form different places (departments, current situation, etc.)
2. represent the current situation on a map
3. predict new cases to come for each department
4. plan transfers
5. display all the transfers from the solution

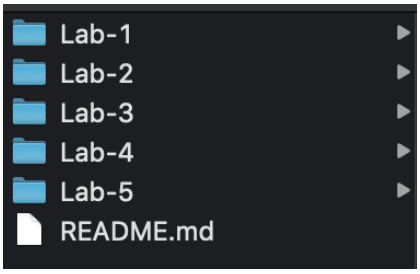
DISCLAIMER: this notebook is thought to demonstrate through an example how Machine Learning and Decision Optmization could be used, but it is partially based on fake data and knowledge of the real problem, and some significant work would be required to incorporate real data and real constraints and obejctives so that the outcome could be useful.

Download files from github

1. If you have not clone the "Applying AI, NLP, and Optimization to COVID-19 Project" from the github site – please enter this URL https://github.com/bleonardb3/AI_POT_11-12-2020 into a web browser to access the github site.
2. Click on the "Clone or download" button. Click the "Download Zip" button. Save and unzip the "AI_POT_11-12-2020-main.zip" file to your local file system.

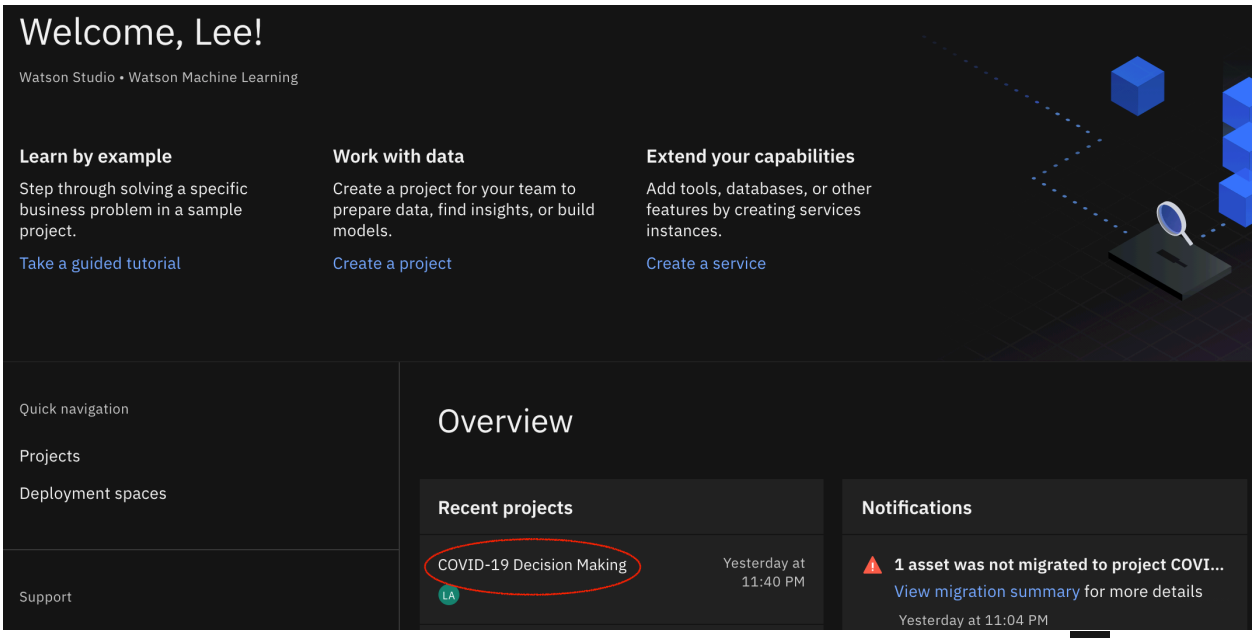


3. You should see these files in your file system.

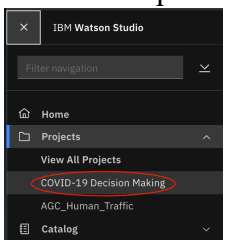


Access Watson Studio COVID-19 Project

1. Enter this URL https://dataplatform.cloud.ibm.com/ into your web browser. Login to Watson Studio. Click on your COVID-19 project.
2. In your "Recently updated projects" click on your "COVID-19 Project" to open it.



3. If you don't see your project once you are logged in – click the hamburger icon ☰ in the top left of your screen. Click your project in the dropdown menu.

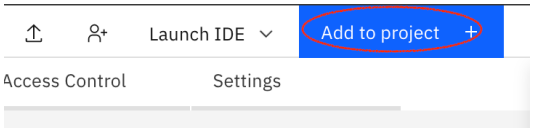


4. You should see a project view of your COVID-19 project.

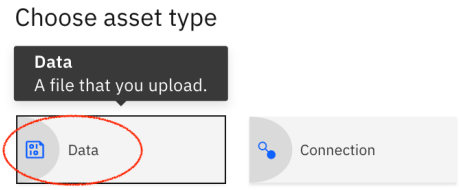
Upload dataset

This exercise you will add a .csv file to your project. You will use this data file to read in the hospitals and their locations into a Pandas series.

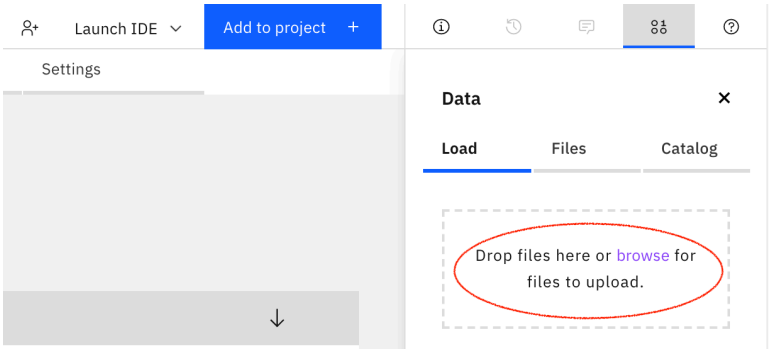
1. Click on the "Assets" tab in the upper left. Then click "Add to project" upper right.



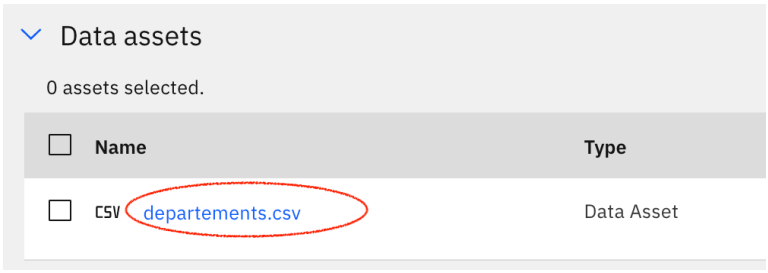
2. Click "Data"



3. In the "Data" pane on the righthand side. Navigate to the "Lab-4" folder > grab and drop the "departements.csv file into the "Load" drop files box.



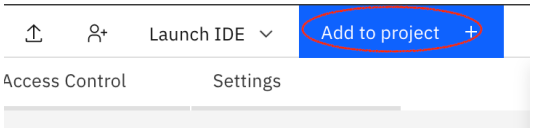
4. You should see the "departements.csv" file displayed in your project > Data assets section.



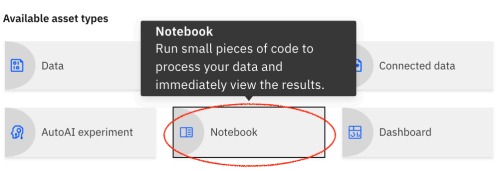
Create a Notebook

This exercise you will add a notebook to your project

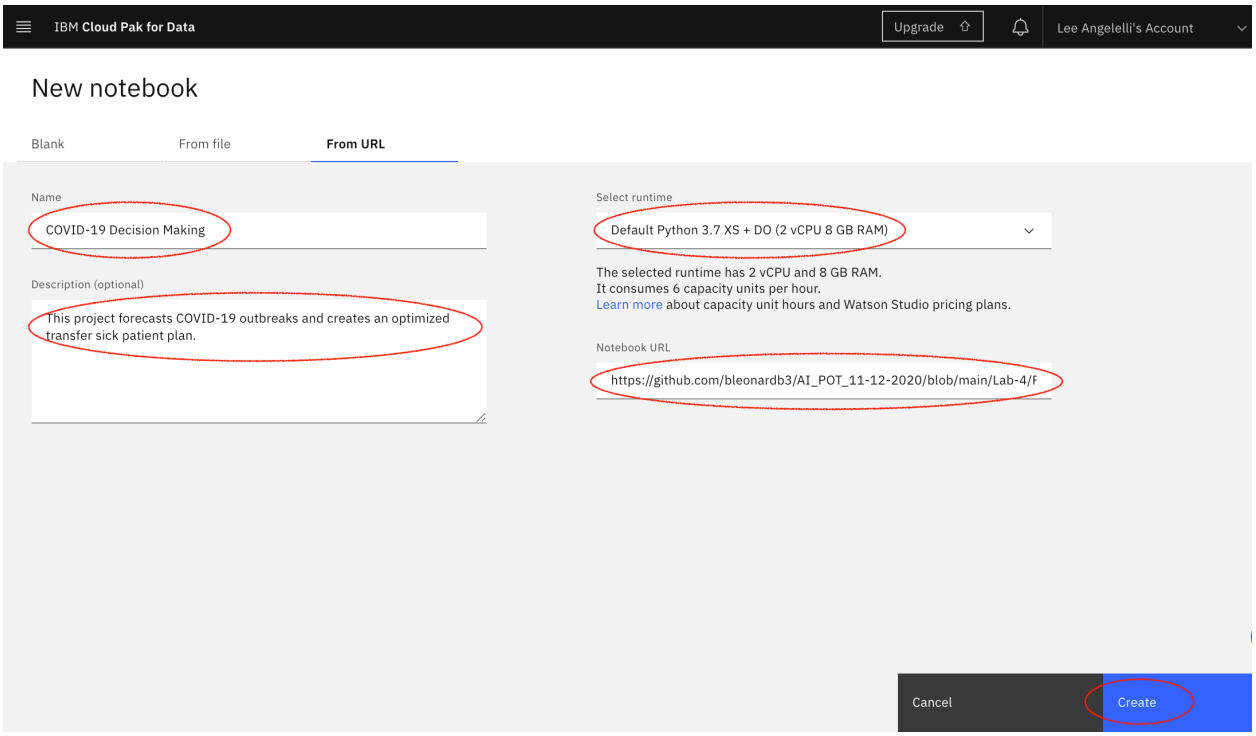
1. Click "Add to project"



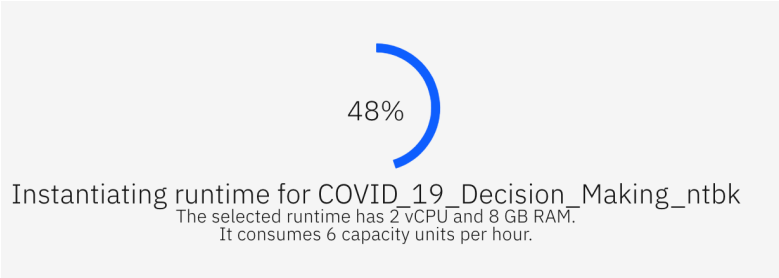
2. Click "Notebook"



3. **Click the "From URL" tab** > Enter a notebook name "COVID-19 Decision Making" > enter a description "This project forecasts COVID-19 outbreaks and creates an optimized transfer sick patient plan." > **make sure you select "Default Python 3.7 XS + DO (2 vCPU 8 GB RAM)" in the "Select runtime" selection list** > copy this URL https://github.com/bleonardb3/AI_POT_11-12-2020/blob/main/Lab-4/FranceCovid19.ipynb into the "Notebbook URL" textbox > click "Create" button.



4. You will see a progress circle showing how much of the notebook is loaded.



Instructions to run a Jupyter Notebook.

5. A Jupyter notebook consists of a series of cells. These cells are of two types (1) documentation cells containing markdown, and (2) code cells (denoted by a bracket on the left of the cell) where you write Python code, R, or Scala code depending on the type of notebook. Code cells can be run by putting the cursor in the code cell and pressing **<Shift><Enter>** on the keyboard. Alternatively, you can execute the cells by clicking on **Run icon** on the menu bar that will run the current cell (where the cursor is located) and then select the cell below. In this way, repeatedly clicking on **Run** executes all the cells in the notebook. When a code cell is executed the brackets on the left change to an asterisk '*' to indicate the code cell is executing. When completed, a sequence number appears. For the rest of this exercise run each cell in the Jupyter notebook.

6. Run pip install brunel.

```
In [ ]: pip install brunel
```

You should see the message "Successfully built brunel". Note: you will NOT need to restart your kernel to use the updated brunel package.

```
Building wheels for collected packages: brunel
  Building wheel for brunel (setup.py) ... done
  Created wheel for brunel: filename=brunel-2.6.2-py3-none-any.whl size=2161246 
00604eea64be76a
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/5b/76/52/7910d0d24e582cc63e
Successfully built brunel
Installing collected packages: Py4J, brunel
Successfully installed Py4J-0.10.9.1 brunel-2.6.2
Note: you may need to restart the kernel to use updated packages.
```

7. Import the necessary files.

```
In [1]: import brunel
        import pandas as pd
        import math
        import csv
        import collections
```

8. This lab uses data from the French government data.gouv.fr site: https://www.data.gouv.fr/fr/datasets/donnees-relatives-a-lepidemie-de-covid-19/# . First we will import data about hospitals' COVID-19 patients.

```
In [2]: url ="https://www.data.gouv.fr/fr/datasets/r/63352e38-d353-4b54-bfd1-f1b3ee1cabd7"
        df=pd.read_csv(url, sep=";")
        df.head()
```

Out[2]:

	dep	sexe	jour	hosp	rea	rad	dc
0	01	0	2020-03-18	2	0	1	0
1	01	1	2020-03-18	1	0	1	0
2	01	2	2020-03-18	1	0	0	0
3	02	0	2020-03-18	41	10	18	11
4	02	1	2020-03-18	19	4	11	6

9. Let's remove some departments which are not concerned by possible moves.

```
In [3]: deps = df.dep.unique()
        deps = deps[deps != '971']
        deps = deps[deps != '972']
        deps = deps[deps != '973']
        deps = deps[deps != '974']
        deps = deps[deps != '976']
        deps = deps[~pd.isna(deps)]
```

10. Let's create a dictionary of aggregated data per department.

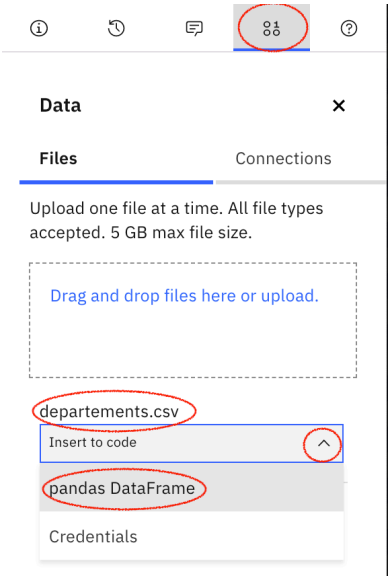
```
In [4]: hosp_data = {}
        for d in deps:
            data = df[df.dep==d]
            data = data.groupby('jour').sum().reset_index().drop(columns=['sexe'])
            data['jour']=pd.to_datetime(data['jour'])
            data = data.sort_values(by=['jour'], ascending=False)
            hosp_data[d] = data
```

11. We will use some GPS data on departments from a loaded data asset. Follow these instructions to import the "departements.csv" that you uploaded in your "Data Assets" above. The departements.csv file contains different French administrative “departments” and their GPS coordinates of frontier and center.

    11.1. Place your cursor in the blank cell.

```
In [ ]:
```

    11.2. Click on the 0/1 icon located in the upper right. In the Data expand menu locate the file "departements.csv". Click on the "Insert to code" dropdown menu select "pandas DataFrame"



    11.3. Change these lines of code:

```
df_data_1 = pd.read_csv(body)
df_data_1.head()
```

    to

```
deps_data = pd.read_csv(body)
deps_data.head()
```

    11.4. Your code cell should look similar to this image.

In [5]:

```
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials
# You might want to remove those credentials before you share the notebook.
client_40296b37f7e14ef5bd0156dcbbe23a9b = ibm_boto3.client(service_name='xxxxxx',
    ibm_api_key_id='xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    ibm_auth_endpoint="https://xxxxxxxxxx",
    config=Config(signature_version='xxxxxxxxx'),
    endpoint_url='https://xxxxxxxxxxx.com')

body = client_40296b37f7e14ef5bd0156dcbbe23a9b.get_object(Bucket='covid19decisionmaking-donotdelete
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

deps_data = pd.read_csv(body)
deps_data.head()
```

11.5. Execute this code cell to import the departements.csv file into a pandas dataframe.

Out[5]:

	Nd	DEPARTEMENT	CAPACITY	LONGITUDE	LATITUDE
0	1	AIN	26	5d20'56" E	46d05'58"
1	2	AISNE	43	3d33'30" E	49d33'34"
2	3	ALLIER	33	3d11'18" E	46d23'37"
3	4	ALPES-DE-HAUTE-PROVENCE	13	6d14'38" E	44d06'22"
4	5	HAUTES-ALPES	24	6d15'47" E	44d39'49"

12. Execute the "strip_zero" function. This is used to fix incompatibilities is using the "deps" list data with the deps_data series data structure.

In [6]:

```
def strip_zero(val):
    if val[0]=='0':
        return (str(int(val)))
    return val
```

13. Convert the departments LONGITUDE and LATITUDE from strings to real numbers.

In [7]:
```
def coord(val):
    neg = (val[-1] == 'O')
    #print (pos)
    c1 = val.split('d')[0]
    #print (c1)
    c2 = val.split('d')[1].split("'")[0]
    #print (c2)
    c3 = val.split('d')[1].split("'")[1].split('"')[0]
    #print (c3)
    res = float(c1) + float(c2)/60 + float(c3)/3600
    if neg:
        res = -res
    return res


deps_data['LONGITUDE'] = deps_data['LONGITUDE'].apply(lambda x: coord(x))
deps_data['LATITUDE'] = deps_data['LATITUDE'].apply(lambda x: coord(x))

deps_data = deps_data.set_index(['Nd'])
deps_data.head()
```

Out[7]:

Nd	DEPARTEMENT	CAPACITY	LONGITUDE	LATITUDE
1	AIN	26	5.348889	46.099444
2	AISNE	43	3.558333	49.559444
3	ALLIER	33	3.188333	46.393611
4	ALPES-DE-HAUTE-PROVENCE	13	6.243889	44.106111
5	HAUTES-ALPES	24	6.263056	44.663611

14. Calculate the magnitude circle based on the number of COVID-19 patients for a given area.

In [8]:
```
def distance(lt1, lg1, lt2, lg2):

    R = 6373.0

    lat1 = math.radians(lt1)
    lon1 = math.radians(lg1)
    lat2 = math.radians(lt2)
    lon2 = math.radians(lg2)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = math.sin(dlat / 2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon / 2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

    distance = R * c
    return distance
```

2. Display the current situation on a map

We will use folium to represent the data on maps.

15. Install folium.

In [9]:
```
!pip install folium
```

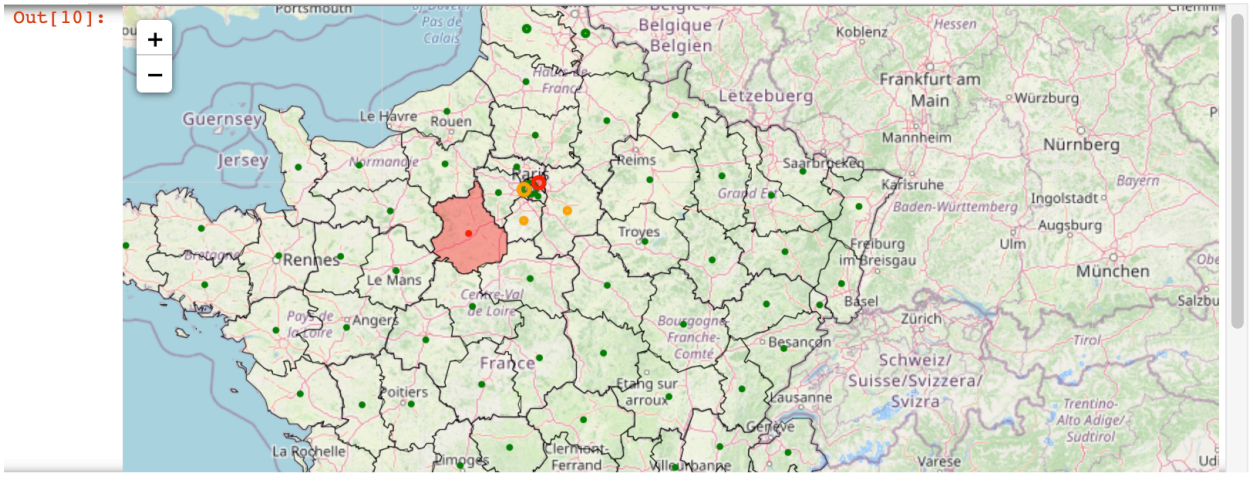
```
Collecting folium
  Downloading https://files.pythonhosted.org/packages/a4/f0/44e69d50519880287cc41e7c8a6acc58daa
9a9acf5f6afc52bcc70f69a6d/folium-0.11.0-py2.py3-none-any.whl (93kB)
     |████████████████████████████████| 102kB 7.7MB/s ta 0:00:011
Requirement already satisfied: numpy in /opt/conda/envs/Python36/lib/python3.6/site-packages (f
rom folium) (1.15.4)
Requirement already satisfied: requests in /opt/conda/envs/Python36/lib/python3.6/site-packages
(from folium) (2.21.0)
Requirement already satisfied: jinja2>=2.9 in /opt/conda/envs/Python36/lib/python3.6/site-packa
ges (from folium) (2.10)
Collecting branca>=0.3.0 (from folium)
  Downloading https://files.pythonhosted.org/packages/13/fb/9eacc24ba3216510c6b59a4ea1cd53d87f2
5ba76237d7f4393abeaf4c94e/branca-0.4.1-py3-none-any.whl
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/conda/envs/Python36/lib/python3.6/
site-packages (from requests->folium) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/envs/Python36/lib/python3.6/sit
e-packages (from requests->folium) (2020.4.5.1)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /opt/conda/envs/Python36/lib/python3.6/
site-packages (from requests->folium) (1.24.1)
Requirement already satisfied: idna<2.9,>=2.5 in /opt/conda/envs/Python36/lib/python3.6/site-pa
ckages (from requests->folium) (2.8)
Requirement already satisfied: MarkupSafe>=0.23 in /opt/conda/envs/Python36/lib/python3.6/site-
packages (from jinja2>=2.9->folium) (1.1.0)
Installing collected packages: branca, folium
Successfully installed branca-0.4.1 folium-0.11.0
```

16. Let's now represent the current situation for each department

In [10]:
```
import folium

m = folium.Map(
    location=[45.5236, 5.6750],
    zoom_start=6
)

initial = {}
capacity = {}

for d, dep in deps_data.iterrows():
    if (d in ['1','2','3','4','5','6','7','8','9']):
        d = '0' + str(d)

    sz = int(hosp_data[d]['rea'].iat[0])
    initial[d] = sz
    capacity[d] = int(1.2*dep['CAPACITY'])

url = 'https://france-geojson.gregoiredavid.fr/repo/departements.geojson'

def style_function(feature):
    d = feature['properties']['code']
    return {
        'fillOpacity': 0.4 if initial[d]>capacity[d] else 0,
        'weight': 0.5,
        'color': 'black',
        'fillColor': 'red' if initial[d]>capacity[d] else 'white'
    }

folium.GeoJson(
    url,
    name='geojson',
    style_function=style_function
).add_to(m)

for d, dep in deps_data.iterrows():
    lt = dep['LATITUDE']
    lg = dep['LONGITUDE']
    nm = dep['DEPARTEMENT']

    if (d in ['1','2','3','4','5','6','7','8','9']):
        d = '0' + str(d)
```

```
    sz = int(hosp_data[d]['rea'].iat[0])
    initial[d] = sz
    #folium.Marker([lt, lg]).add_to(here_map)
    color = 'green'
    if initial[d]>0.5*capacity[d]:
        color = 'orange'
    if initial[d]>capacity[d]:
        color='red'
    folium.Circle(
        radius=sz*50,
        location=[lt, lg],
        popup=folium.Popup(nm + '(' + str(d)+')<br>'+str(sz)+' reas.<br>Capacity: '+str(capacity[d]
        color=color,
        fill=True,
        fill_color=color
    ).add_to(m)


m
```

Out[10]:



17. Create dataframe with status before transfers and calculate the total over capacity

In [16]:
```
before = pd.DataFrame(columns=['Nd', 'capacity', 'value', 'overcapacity'], data=[[d, capacity[d], i
print (before)

total_overall_before = before['overcapacity'].sum()
print(total_overall_before)
```

	Nd	capacity	value	overcapacity
0	01	31	0	0
1	02	51	16	0
2	03	39	2	0
3	04	15	0	0
4	05	28	0	0
5	06	160	2	0
6	07	22	2	0
7	08	25	6	0
8	09	25	2	0
9	10	19	6	0
10	11	37	2	0
11	12	34	0	0
12	13	296	77	0
13	14	110	2	0
14	15	20	0	0
15	16	27	2	0

3. Predict the new cases

18. We use a very dummy linear regression to predict each department's new COVID-19 cases

In [17]:
```
import numpy as np
from sklearn.linear_model import LinearRegression


NB_PERIODS = 3

def predict_more(d, n):
    X = hosp_data[d].index.tolist()
    X.reverse()
    X=np.array(X).reshape(-1,1)
    y = hosp_data[d]['rea'].tolist()
    y.reverse()
    y=np.array(y).reshape(-1,1)

    regsr=LinearRegression()
    regsr.fit(X,y)

    to_predict_x = [i for i in range(len(X), len(X)+n)]
    to_predict_x= np.array(to_predict_x).reshape(-1,1)
    predicted_y= regsr.predict(to_predict_x)
    delta_y = [ int(round(max(0, predicted_y[i][0]-y[len(y)-1][0]))) for i in range(n)]
    return delta_y

new_rea ={ d:predict_more(d, NB_PERIODS) for d in deps}
print (new_rea)
```

```
{'01': [0, 0, 0], '02': [5, 4, 3], '03': [0, 0, 0], '04': [0, 0, 0], '05': [0, 0, 0], '06': [3,
2, 1], '07': [9, 8, 8], '08': [0, 0, 0], '09': [0, 0, 0], '10': [0, 0, 0], '11': [0, 0, 0], '1
2': [0, 0, 0], '13': [28, 25, 22], '14': [0, 0, 0], '15': [1, 0, 0], '16': [0, 0, 0], '17': [0,
0, 0], '18': [0, 0, 0], '19': [5, 4, 4], '21': [0, 0, 0], '22': [3, 3, 3], '23': [7, 7, 7], '2
4': [0, 0, 0], '25': [0, 0, 0], '26': [0, 0, 0], '27': [1, 1, 1], '28': [1, 1, 0], '29': [0, 0,
0], '2A': [0, 0, 0], '2B': [2, 2, 2], '30': [9, 8, 8], '31': [1, 0, 0], '32': [0, 0, 0], '33':
[0, 0, 0], '34': [0, 0, 0], '35': [0, 0, 0], '36': [1, 1, 0], '37': [6, 5, 5], '38': [3, 2, 1],
'39': [0, 0, 0], '40': [0, 0, 0], '41': [5, 5, 4], '42': [16, 14, 13], '43': [0, 0, 0], '44':
[0, 0, 0], '45': [10, 9, 8], '46': [0, 0, 0], '47': [1, 1, 1], '48': [0, 0, 0], '49': [4, 3,
2], '50': [0, 0, 0], '51': [0, 0, 0], '52': [5, 4, 4], '53': [1, 1, 1], '54': [0, 0, 0], '55':
[0, 0, 0], '56': [3, 2, 2], '57': [0, 0, 0], '58': [0, 0, 0], '59': [12, 8, 4], '60': [0, 0,
0], '61': [1, 1, 1], '62': [9, 8, 7], '63': [0, 0, 0], '64': [0, 0, 0], '65': [1, 1, 0], '66':
[0, 0, 0], '67': [0, 0, 0], '68': [0, 0, 0], '69': [23, 19, 15], '70': [0, 0, 0], '71': [1, 1,
0], '72': [0, 0, 0], '73': [0, 0, 0], '74': [0, 0, 0], '75': [30, 18, 6], '76': [0, 0, 0], '7
7': [25, 22, 20], '78': [0, 0, 0], '79': [0, 0, 0], '80': [0, 0, 0], '81': [0, 0, 0], '82': [3,
3, 3], '83': [0, 0, 0], '84': [0, 0, 0], '85': [4, 4, 4], '86': [0, 0, 0], '87': [0, 0, 0], '8
8': [0, 0, 0], '89': [13, 12, 12], '90': [0, 0, 0], '91': [22, 19, 17], '92': [5, 0, 0], '93':
[34, 32, 29], '94': [47, 42, 37], '95': [0, 0, 0]}
```

4. Optimize the transfers using Decision Optimization

19. Now let's create an optimization model.

Not knowing the real capacities and the real constraints and objectives, we have imagined that some limited short and long transfers could be organized in order to minimize the total over capacity.

In [12]:
```
from docplex.mp.environment import Environment
env = Environment()
env.print_information()
```

```
   * system is: Linux 64bit
   * Python version 3.6.9, located at: /opt/conda/envs/Python36/bin/python
   * docplex is present, version is 2.14.186
   * CPLEX library is present, version is 12.10.0.0, located at: /opt/conda/envs/Python36/lib/pyth
   on3.6/site-packages
   * pandas is present, version is 0.24.1
```

20. Parameters are the number of possible moves, and the maximum number of cases per move.

In [14]:
```
from docplex.mp.model import Model
mdl = Model("Plan Transfers")

MAX_NB_LONG_TRANSFERS_PER_PERIOD = 3
MAX_CASES_PER_LONG_TRANSFERS = 20

MAX_NB_SHORT_TRANSFERS_PER_DEPARTMENT = 3

LONG_DISTANCE = 200
```

21. For each "Department" determine if the distance between itself and all other departments is either a "SHORT" or "LONG" transportation route.

In [15]:
```
is_long = {d1: {d2: distance(deps_data['LATITUDE'][strip_zero(d1)], deps_data['LONGITUDE'][strip_zero(d1)], deps_data['LATITUDE'
```

22. Create a "Plan Transfers" short transfer bounds.

In [16]:
```
transfer_periods = list(range(NB_PERIODS))
all_periods = list(range(NB_PERIODS+1))

# Binary vars representing the "assigned" libraries for each coffee shop
use_link_vars = mdl.binary_var_cube(deps, deps, transfer_periods, name="use_link")
link_vars = mdl.integer_var_cube(deps, deps, transfer_periods, lb=0, ub=MAX_CASES_PER_LONG_TRANSFERS, name="link")
occupancy_vars = mdl.integer_var_matrix(deps, all_periods, lb=0, name="occupancy")

# Short transfers bounds
mdl.add_constraints(link_vars[d1, d2, t] <= 1 for d1 in deps for d2 in deps if not is_long[d1][d2] for t in transfer_periods)

mdl.print_information()
```

```
Model: Plan Transfers
 - number of variables: 55680
   - binary=27648, integer=28032, continuous=0
 - number of constraints: 4890
   - linear=4890
 - parameters: defaults
 - objective: none
 - problem type is: MILP
```

23. Add the constraints

In [17]:
```
# Initial state
mdl.add_constraints(occupancy_vars[d, 0] == initial[d] for d in deps)

# structural constraint between user_link and link
mdl.add_constraints(use_link_vars[d, d1, t] == (link_vars[d, d1, t] >= 1) for d in deps for d1 in deps for t in transfer_periods

# number of transfers from a department less than current number of cases
mdl.add_constraints(mdl.sum(link_vars[d, d1, t] for d1 in deps) <= occupancy_vars[d, t] for d in deps for t in transfer_periods)

# maximum number of LONG transfers
mdl.add_constraints(mdl.sum(use_link_vars[d1, d2, t] for d1 in deps for d2 in deps if is_long[d1][d2]) <= MAX_NB_LONG_TRANSFERS_


# maximum number of SHORT transfers
mdl.add_constraints(mdl.sum(use_link_vars[d1, d2, t] for d1 in deps if not is_long[d1][d2] for t in transfer_periods) <= MAX_NB_

# conservation constraints including new cases to come
mdl.add_constraints(occupancy_vars[d, t+1] == new_rea[d][t] + occupancy_vars[d, t] + mdl.sum(link_vars[d1, d, t] for d1 in deps)

mdl.print_information()
```

```
Model: Plan Transfers
 - number of variables: 83328
   - binary=55296, integer=28032, continuous=0
 - number of constraints: 60957
   - linear=33309, equiv=27648
 - parameters: defaults
 - objective: none
 - problem type is: MILP
```

24. Objective is to minimize the total over capacity at the end.

In [18]:
```
final_overcapacity = mdl.sum(mdl.max(0,  occupancy_vars[d, NB_PERIODS] - capacity[d]) for d in deps)
mdl.add_kpi(final_overcapacity)

nb_long_transfers = mdl.sum(use_link_vars[d1, d2, t] for d1 in deps for d2 in deps if is_long[d1][d2] for t in transfer_periods)
mdl.add_kpi(nb_long_transfers)

nb_short_transfers = mdl.sum(use_link_vars[d1, d2, t] for d1 in deps for d2 in deps if not is_long[d1][d2] for t in transfer_per
mdl.add_kpi(nb_short_transfers)

mdl.minimize(1000 * final_overcapacity + 10 * nb_long_transfers + nb_short_transfers)
```

25. Have the "docplex" solve the optimization model.

In [19]:
```
mdl.set_time_limit(20)
mdl.solve(log_output=True)
```

```
Performing restart 1

Repeating presolve.
Tried aggregator 2 times.
MIP Presolve eliminated 2 rows and 192 columns.
MIP Presolve modified 11084 coefficients.
Aggregator did 96 substitutions.
Reduced MIP has 41747 rows, 45539 columns, and 179834 nonzeros.
Reduced MIP has 24700 binaries, 20839 generals, 0 SOSs, and 17081 indicators.
Presolve time = 0.44 sec. (263.28 ticks)
Tried aggregator 1 time.
Reduced MIP has 41747 rows, 45539 columns, and 179834 nonzeros.
Reduced MIP has 24700 binaries, 20839 generals, 0 SOSs, and 17081 indicators.
Presolve time = 0.27 sec. (143.52 ticks)
Represolve time = 1.30 sec. (587.15 ticks)
*    821+    0                          104.0000       90.0000           13.46%
     821     0      100.0000     9      104.0000      Cuts: 24     5147    3.85%
*    821     0      integral     0      100.0000      Cuts: 11     5332    0.00%
     821     0        cutoff             100.0000      100.0000     5332    0.00%

Flow cuts applied:  2
Mixed integer rounding cuts applied:  16
Zero-half cuts applied:  2
Lift and project cuts applied:  1
Gomory fractional cuts applied:  8

Root node processing (before b&c):
  Real time             =    5.83 sec. (6896.49 ticks)
Parallel b&c, 2 threads:
  Real time             =   10.44 sec. (8356.22 ticks)
  Sync time (average)   =    0.75 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =   16.26 sec. (15252.71 ticks)
```

Out[19]: docplex.mp.solution.SolveSolution(obj=100,values={use_link_07_13_1:1,use..

26. Extract and print the moves

In [20]:
```
edges = [(t, d1, d2, int(link_vars[d1, d2, t]), is_long[d1][d2]) for t in transfer_periods for d1 in deps for d2 in deps if int(
print (edges)
```

```
[(0, '28', '18', 1, False), (0, '28', '27', 1, False), (0, '91', '18', 1, False), (0, '91', '27', 1, False), (0, '91', '51',
1, False), (0, '91', '76', 1, False), (0, '91', '78', 1, False), (0, '93', '01', 20, True), (0, '93', '05', 20, True), (0, '9
3', '38', 20, True), (1, '07', '13', 1, False), (1, '89', '70', 1, False), (1, '91', '10', 1, False), (1, '91', '27', 1, Fals
e), (1, '93', '56', 20, True), (1, '93', '79', 20, True), (1, '94', '10', 1, False), (1, '94', '18', 1, False), (1, '94', '3
5', 20, True), (1, '94', '76', 1, False), (2, '28', '53', 1, False), (2, '28', '78', 1, False), (2, '91', '50', 20, True),
(2, '91', '77', 1, False), (2, '94', '02', 1, False), (2, '94', '10', 1, False), (2, '94', '30', 20, True), (2, '94', '80',
1, False)]
```

27. Recreate structure for final situation and calculate final over capacity.

In [21]:
```
final = { d: int(occupancy_vars[d, NB_PERIODS].solution_value) for d in deps }

after = pd.DataFrame(columns=['Nd', 'capacity', 'value', 'overcapacity'], data=[[d, capacity[d], final[d], max(0, final[d]-capac
print(after)

total_overall_after = after['overcapacity'].sum()
print(total_overall_after)
```

```
    Nd  capacity  value  overcapacity
0   01        31     20             0
1   02        51     28             0
2   03        39      2             0
3   04        15      0             0
4   05        28     20             0
5   06       160      3             0
6   07        22     22             0
7   08        25      5             0
8   09        25      2             0
9   10        19      9             0
10  11        37      2             0

94  94        88     88             0
95  95       117     16             0

[96 rows x 4 columns]
0
```

5. Display the solution

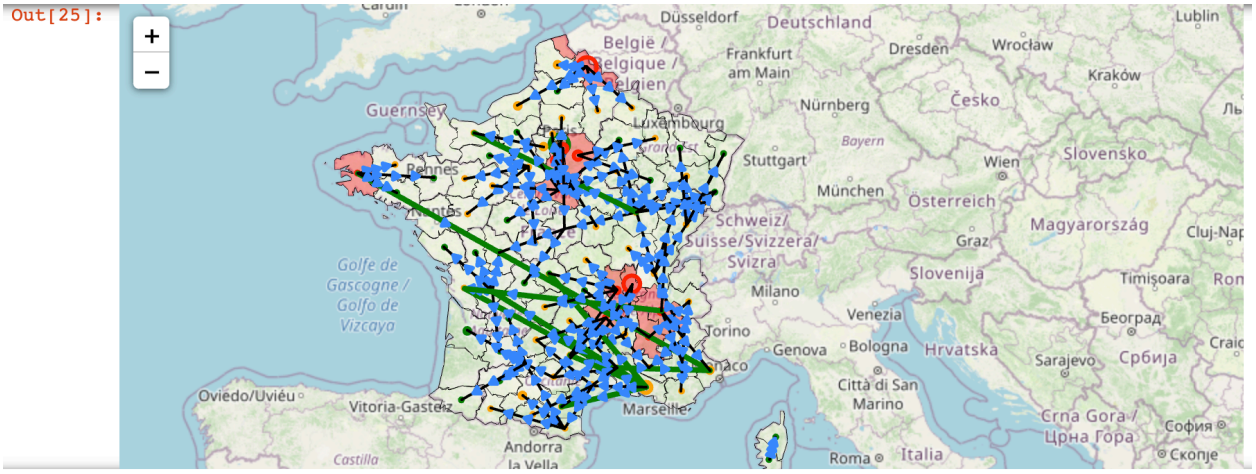
28. And represent the moves and the changes for impacted departments.

- Black links are the local transfers.
- Green links are the long distance transfers

In [47]:

```
def style_function(feature):
    d = feature['properties']['code']
    return {
        'fillOpacity': 0.4 if final[d]>capacity[d] else 0,
        'weight': 0.5,
        'color': 'black',
        'fillColor': 'red' if final[d]>capacity[d] else 'white'
    }


def get_angle(p1, p2):

    '''
    This function Returns angle value in degree from the location p1 to location p2

    Parameters it accepts :
    p1 : namedtuple with lat lon
    p2 : namedtuple with lat lon

    This function Return the vlaue of degree in the data type float

    Pleae also refers to for better understanding : https://gist.github.com/jeromer/2005586
    '''

    longitude_diff = np.radians(p2.lon - p1.lon)

    latitude1 = np.radians(p1.lat)
    latitude2 = np.radians(p2.lat)

    x_vector = np.sin(longitude_diff) * np.cos(latitude2)
    y_vector = (np.cos(latitude1) * np.sin(latitude2)
        - (np.sin(latitude1) * np.cos(latitude2)
        * np.cos(longitude_diff)))
    angle = np.degrees(np.arctan2(x_vector, y_vector))

    # Checking and adjustring angle value on the scale of 360
    if angle < 0:
        return angle + 360
    return angle


def get_arrows(locations, color, size, n_arrows):

    '''
    Get a list of correctly placed and rotated
    arrows/markers to be plotted
```

Out[25]:



Great job! Now you are ready to help communities and businesses develop optimization plans to respond to predicted COVID-19 infections.