Week4 Assignment

May 7, 2020

[1]:
```
import requests # library to handle requests
import pandas as pd # library for data analsysis
import numpy as np # library to handle data in a vectorized manner
import random # library for random number generation
!conda install -c conda-forge lxml --yes
!conda install -c conda-forge geopy --yes
from geopy.geocoders import Nominatim # module to convert an address into␣
↪latitude and longitude values

# libraries for displaying images
from IPython.display import Image
from IPython.core.display import HTML

# tranforming json file into a pandas dataframe library
from pandas.io.json import json_normalize

!conda install -c conda-forge folium=0.5.0 --yes
import folium # plotting library
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/jupyterlab/conda/envs/python

  added / updated specs:
    - lxml


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    libxslt-1.1.33             |       h7d1a2b0_0         426 KB
    lxml-3.8.0                 |           py36_0         3.8 MB  conda-forge
    openssl-1.1.1g             |       h516909a_0         2.1 MB  conda-forge
    ------------------------------------------------------------
```

```
                                          Total:         6.4 MB

The following NEW packages will be INSTALLED:

  libxslt            pkgs/main/linux-64::libxslt-1.1.33-h7d1a2b0_0
  lxml               conda-forge/linux-64::lxml-3.8.0-py36_0

The following packages will be UPDATED:

  openssl                                 1.1.1f-h516909a_0 -->
1.1.1g-h516909a_0



Downloading and Extracting Packages
openssl-1.1.1g       | 2.1 MB    | ##################################### | 100%
lxml-3.8.0           | 3.8 MB    | ##################################### | 100%
libxslt-1.1.33       | 426 KB    | ##################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/jupyterlab/conda/envs/python

  added / updated specs:
    - geopy


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    geographiclib-1.50         |             py_0          34 KB  conda-forge
    geopy-1.21.0               |             py_0          58 KB  conda-forge
    ------------------------------------------------------------
                                          Total:          92 KB

The following NEW packages will be INSTALLED:

  geographiclib      conda-forge/noarch::geographiclib-1.50-py_0
  geopy              conda-forge/noarch::geopy-1.21.0-py_0
```

```
Downloading and Extracting Packages
geopy-1.21.0         | 58 KB     | ##################################### | 100%
geographiclib-1.50   | 34 KB     | ##################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible
solve.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/jupyterlab/conda/envs/python

  added / updated specs:
    - folium=0.5.0


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    altair-4.1.0               |             py_1         614 KB  conda-forge
    branca-0.4.1               |             py_0          26 KB  conda-forge
    brotlipy-0.7.0             |py36h8c4c3a4_1000         346 KB  conda-forge
    chardet-3.0.4              |py36h9f0ad1d_1006         188 KB  conda-forge
    cryptography-2.9.2         |   py36h45558ae_0         613 KB  conda-forge
    folium-0.5.0               |             py_0          45 KB  conda-forge
    pandas-1.0.3               |   py36h830a2c2_1        11.1 MB  conda-forge
    pysocks-1.7.1              |   py36h9f0ad1d_1          27 KB  conda-forge
    pytz-2020.1                |     pyh9f0ad1d_0         227 KB  conda-forge
    toolz-0.10.0               |             py_0          46 KB  conda-forge
    urllib3-1.25.9             |             py_0          92 KB  conda-forge
    vincent-0.4.4              |             py_1          28 KB  conda-forge
    ------------------------------------------------------------
                                           Total:        13.3 MB

The following NEW packages will be INSTALLED:

  altair             conda-forge/noarch::altair-4.1.0-py_1
  attrs              conda-forge/noarch::attrs-19.3.0-py_0
  branca             conda-forge/noarch::branca-0.4.1-py_0
  brotlipy           conda-forge/linux-64::brotlipy-0.7.0-py36h8c4c3a4_1000
  chardet            conda-forge/linux-64::chardet-3.0.4-py36h9f0ad1d_1006
  cryptography       conda-forge/linux-64::cryptography-2.9.2-py36h45558ae_0
  entrypoints        conda-forge/linux-64::entrypoints-0.3-py36h9f0ad1d_1001
```

```
  folium             conda-forge/noarch::folium-0.5.0-py_0
  idna               conda-forge/noarch::idna-2.9-py_1
  importlib-metadata conda-forge/linux-64::importlib-
metadata-1.6.0-py36h9f0ad1d_0
  importlib_metadata conda-forge/noarch::importlib_metadata-1.6.0-0
  jinja2             conda-forge/noarch::jinja2-2.11.2-pyh9f0ad1d_0
  jsonschema         conda-forge/linux-64::jsonschema-3.2.0-py36h9f0ad1d_1
  markupsafe         conda-forge/linux-64::markupsafe-1.1.1-py36h8c4c3a4_1
  pandas             conda-forge/linux-64::pandas-1.0.3-py36h830a2c2_1
  pyopenssl          conda-forge/noarch::pyopenssl-19.1.0-py_1
  pyrsistent         conda-forge/linux-64::pyrsistent-0.16.0-py36h8c4c3a4_0
  pysocks            conda-forge/linux-64::pysocks-1.7.1-py36h9f0ad1d_1
  pytz               conda-forge/noarch::pytz-2020.1-pyh9f0ad1d_0
  requests           conda-forge/noarch::requests-2.23.0-pyh8c360ce_2
  toolz              conda-forge/noarch::toolz-0.10.0-py_0
  urllib3            conda-forge/noarch::urllib3-1.25.9-py_0
  vincent            conda-forge/noarch::vincent-0.4.4-py_1
  zipp               conda-forge/noarch::zipp-3.1.0-py_0


Downloading and Extracting Packages
pysocks-1.7.1        | 27 KB     | ##################################### | 100%
toolz-0.10.0         | 46 KB     | ##################################### | 100%
pytz-2020.1          | 227 KB    | ##################################### | 100%
chardet-3.0.4        | 188 KB    | ##################################### | 100%
folium-0.5.0         | 45 KB     | ##################################### | 100%
urllib3-1.25.9       | 92 KB     | ##################################### | 100%
branca-0.4.1         | 26 KB     | ##################################### | 100%
cryptography-2.9.2   | 613 KB    | ##################################### | 100%
brotlipy-0.7.0       | 346 KB    | ##################################### | 100%
pandas-1.0.3         | 11.1 MB   | ##################################### | 100%
altair-4.1.0         | 614 KB    | ##################################### | 100%
vincent-0.4.4        | 28 KB     | ##################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

[2]:
```
CLIENT_ID = 'UFBYQ1N3XR3LBAEJNASRHROFDUCSQQYXBGXH1AWSGVSCFUPI' # your␣
 ↪Foursquare ID
CLIENT_SECRET = 'JHMO2LEJOORHRRCSJKJYWBLJFOOB2HGCIOJOOONOVLU5KNVP' # your␣
 ↪Foursquare Secret
VERSION = '20180604'
LIMIT = 30
search_query = 'Chinese'
radius = 15000
geolocator = Nominatim(user_agent="foursquare_agent")
```

[3]:
```
#Foursquare data, closest Chinese restaurant from San Siro Stadium Milan
address_milan = 'San Siro Stadium'
location_milan = geolocator.geocode(address_milan)
latitude_milan = location_milan.latitude
longitude_milan = location_milan.longitude
url_milan = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_milan, longitude_milan, VERSION,␣
 ↪search_query, radius, LIMIT)
results_milan = requests.get(url_milan).json()
venues_milan = results_milan['response']['venues']
df_milan = json_normalize(venues_milan)
df_milan['City'] = 'Milan'

#Foursquare data, closest Chinese restaurant from Allianz Stadium, Turin
address_turin = 'Allianz Stadium'
location_turin = geolocator.geocode(address_turin)
latitude_turin = location_turin.latitude
longitude_turin = location_turin.longitude
url_turin = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_turin, longitude_turin, VERSION,␣
 ↪search_query, radius, LIMIT)
results_turin = requests.get(url_turin).json()
venues_turin = results_turin['response']['venues']
df_turin = json_normalize(venues_turin)
df_turin['City'] = 'Turin'

#Foursquare data from Stadion Maksimir, Zagreb
address_zagreb = 'Stadion Maksimir'
location_zagreb = geolocator.geocode(address_zagreb)
latitude_zagreb = location_zagreb.latitude
longitude_zagreb = location_zagreb.longitude
url_zagreb = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_zagreb, longitude_zagreb, VERSION,␣
 ↪search_query, radius, LIMIT)
results_zagreb = requests.get(url_zagreb).json()
venues_zagreb = results_zagreb['response']['venues']
df_zagreb = json_normalize(venues_zagreb)
df_zagreb['City'] = 'Zagreb'

#Foursquare data from Stadio Olimpico, Rome
address_rome = 'Stadio Olimpico'
location_rome = geolocator.geocode(address_rome)
latitude_rome = location_rome.latitude
longitude_rome = location_rome.longitude
```

```
url_rome = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_rome, longitude_rome, VERSION,␣
 ↪search_query, radius, LIMIT)
results_rome = requests.get(url_rome).json()
venues_rome = results_rome['response']['venues']
df_rome = json_normalize(venues_rome)
df_rome['City'] = 'Rome'

#Foursquare data from Stadio San Paolo, Naples
address_naples = 'Stadio San Paolo'
location_naples = geolocator.geocode(address_naples)
latitude_naples = location_naples.latitude
longitude_naples = location_naples.longitude
url_naples = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_naples, longitude_naples, VERSION,␣
 ↪search_query, radius, LIMIT)
results_naples = requests.get(url_naples).json()
venues_naples = results_naples['response']['venues']
df_naples = json_normalize(venues_naples)
df_naples['City'] = 'Naples'

#Foursquare data from Allianz Arena, Munich
address_munich = 'Allianz arena'
location_munich = geolocator.geocode(address_munich)
latitude_munich = location_munich.latitude
longitude_munich = location_munich.longitude
url_munich = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_munich, longitude_munich, VERSION,␣
 ↪search_query, radius, LIMIT)
results_munich = requests.get(url_munich).json()
venues_munich = results_munich['response']['venues']
df_munich = json_normalize(venues_munich)
df_munich['City'] = 'Munich'

#Foursquare data from Red Bull Arena, Leipzig
address_leipzig = 'Red Bull Arena'
location_leipzig = geolocator.geocode(address_leipzig)
latitude_leipzig = location_leipzig.latitude
longitude_leipzig = location_leipzig.longitude
url_leipzig = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_leipzig, longitude_leipzig,␣
 ↪VERSION, search_query, radius, LIMIT)
results_leipzig = requests.get(url_leipzig).json()
```

```
venues_leipzig = results_leipzig['response']['venues']
df_leipzig = json_normalize(venues_leipzig)
df_leipzig['City'] = 'Leipzig'

#Foursquare data from Volksparkstadion, Hamburg
address_hamburg = 'Volksparkstadion'
location_hamburg = geolocator.geocode(address_hamburg)
latitude_hamburg = location_hamburg.latitude
longitude_hamburg = location_hamburg.longitude
url_hamburg = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_hamburg, longitude_hamburg,␣
 ↪VERSION, search_query, radius, LIMIT)
results_hamburg = requests.get(url_hamburg).json()
venues_hamburg = results_hamburg['response']['venues']
df_hamburg = json_normalize(venues_hamburg)
df_hamburg['City'] = 'Hamburg'

#Foursquare data from Olympiastadion Berlin, Berlin
address_berlin = 'Olympiastadion Berlin'
location_berlin = geolocator.geocode(address_berlin)
latitude_berlin = location_berlin.latitude
longitude_berlin = location_berlin.longitude
url_berlin = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_berlin, longitude_berlin, VERSION,␣
 ↪search_query, radius, LIMIT)
results_berlin = requests.get(url_berlin).json()
venues_berlin = results_berlin['response']['venues']
df_berlin = json_normalize(venues_berlin)
df_berlin['City'] = 'Berlin'

#Foursquare data from Le Parc des Princes, Paris
address_paris = 'Le Parc des Princes'
location_paris = geolocator.geocode(address_paris)
latitude_paris = location_paris.latitude
longitude_paris = location_paris.longitude
url_paris = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_paris, longitude_paris, VERSION,␣
 ↪search_query, radius, LIMIT)
results_paris = requests.get(url_paris).json()
venues_paris = results_paris['response']['venues']
df_paris = json_normalize(venues_paris)
df_paris['City'] = 'Paris'

#Foursquare data from Stadio Artemio Franchi, Florence
```

```
address_florence = 'Stadio Artemio Franchi'
location_florence = geolocator.geocode(address_florence)
latitude_florence = location_florence.latitude
longitude_florence = location_florence.longitude
url_florence = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_florence, longitude_florence,␣
 ↪VERSION, search_query, 15000, LIMIT)
results_florence = requests.get(url_florence).json()
venues_florence = results_florence['response']['venues']
df_florence = json_normalize(venues_florence)
df_florence['City'] = 'Florence'

#Foursquare data from Stadium de Toulouse, Toulouse
address_toulouse = 'Stadium de Toulouse'
location_toulouse = geolocator.geocode(address_toulouse)
latitude_toulouse = location_toulouse.latitude
longitude_toulouse = location_toulouse.longitude
url_toulouse = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_toulouse, longitude_toulouse,␣
 ↪VERSION, search_query, radius, LIMIT)
results_toulouse = requests.get(url_toulouse).json()
venues_toulouse = results_toulouse['response']['venues']
df_toulouse = json_normalize(venues_toulouse)
df_toulouse['City'] = 'Toulouse'

#Foursquare data from Emirates Stadium, London
address_london = 'Emirates Stadium'
location_london = geolocator.geocode(address_london)
latitude_london = location_london.latitude
longitude_london = location_london.longitude
url_london = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_london, longitude_london, VERSION,␣
 ↪search_query, radius, LIMIT)
results_london = requests.get(url_london).json()
venues_london = results_london['response']['venues']
df_london = json_normalize(venues_london)
df_london['City'] = 'London'

#Foursquare data from Anfield Stadium, Liverpool
address_liverpool = 'Anfield Stadium'
location_liverpool = geolocator.geocode(address_liverpool)
latitude_liverpool = location_liverpool.latitude
longitude_liverpool = location_liverpool.longitude
```

```
url_liverpool = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_liverpool, longitude_liverpool,␣
 ↪VERSION, search_query, radius, LIMIT)
results_liverpool = requests.get(url_liverpool).json()
venues_liverpool = results_liverpool['response']['venues']
df_liverpool = json_normalize(venues_liverpool)
df_liverpool['City'] = 'Liverpool'

#Foursquare data from Old Trafford, Manchester
address_manchester = 'Old Trafford'
location_manchester = geolocator.geocode(address_manchester)
latitude_manchester = location_manchester.latitude
longitude_manchester = location_manchester.longitude
url_manchester = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_manchester, longitude_manchester,␣
 ↪VERSION, search_query, radius, LIMIT)
results_manchester = requests.get(url_manchester).json()
venues_manchester = results_manchester['response']['venues']
df_manchester = json_normalize(venues_manchester)
df_manchester['City'] = 'Manchester'

#Foursquare data from Krestovsky Stadium, Saint Petersburg
address_saint_petersburg = 'Krestovsky Stadium'
location_saint_petersburg = geolocator.geocode(address_saint_petersburg)
latitude_saint_petersburg = location_saint_petersburg.latitude
longitude_saint_petersburg = location_saint_petersburg.longitude
url_saint_petersburg = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_saint_petersburg,␣
 ↪longitude_saint_petersburg, VERSION, search_query, radius, LIMIT)
results_saint_petersburg = requests.get(url_saint_petersburg).json()
venues_saint_petersburg = results_saint_petersburg['response']['venues']
df_saint_petersburg = json_normalize(venues_saint_petersburg)
df_saint_petersburg['City'] = 'Saint Petersburg'

#Foursquare data from Camp Nou, Barcelona
address_barcelona = 'Camp Nou'
location_barcelona = geolocator.geocode(address_barcelona)
latitude_barcelona = location_barcelona.latitude
longitude_barcelona = location_barcelona.longitude
url_barcelona = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_barcelona, longitude_barcelona,␣
 ↪VERSION, search_query, radius, LIMIT)
results_barcelona = requests.get(url_barcelona).json()
```

```
venues_barcelona = results_barcelona['response']['venues']
df_barcelona = json_normalize(venues_barcelona)
df_barcelona['City'] = 'Barcelona'

#Foursquare data from Santiago Bernabeu Stadium, Madrid
address_madrid = 'Santiago Bernabeu Stadium'
location_madrid = geolocator.geocode(address_madrid)
latitude_madrid = location_madrid.latitude
longitude_madrid = location_madrid.longitude
url_madrid = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_madrid, longitude_madrid, VERSION,␣
 ↪search_query, radius, LIMIT)
results_madrid = requests.get(url_madrid).json()
venues_madrid = results_madrid['response']['venues']
df_madrid = json_normalize(venues_madrid)
df_madrid['City'] = 'Madrid'

#Foursquare data from Ramon Sanchez-Pizjuan Stadium, Sevilla
address_sevilla = 'Ramon Sanchez-Pizjuan Stadium'
location_sevilla = geolocator.geocode(address_sevilla)
latitude_sevilla = location_sevilla.latitude
longitude_sevilla = location_sevilla.longitude
url_sevilla = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_sevilla, longitude_sevilla,␣
 ↪VERSION, search_query, radius, LIMIT)
results_sevilla = requests.get(url_sevilla).json()
venues_sevilla = results_sevilla['response']['venues']
df_sevilla = json_normalize(venues_sevilla)
df_sevilla['City'] = 'Sevilla'

#Foursquare data from Johan Cruijff Arena, Amsterdam
address_amsterdam = 'Johan Cruijff Arena'
location_amsterdam = geolocator.geocode(address_amsterdam)
latitude_amsterdam = location_amsterdam.latitude
longitude_amsterdam = location_amsterdam.longitude
url_amsterdam = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_amsterdam, longitude_amsterdam,␣
 ↪VERSION, search_query, radius, LIMIT)
results_amsterdam = requests.get(url_amsterdam).json()
venues_amsterdam = results_amsterdam['response']['venues']
df_amsterdam = json_normalize(venues_amsterdam)
df_amsterdam['City'] = 'Amsterdam'

#Foursquare data from Estadio do Dragao, Porto
```

```
address_porto = 'Estadio do Dragao'
location_porto = geolocator.geocode(address_porto)
latitude_porto = location_porto.latitude
longitude_porto = location_porto.longitude
url_porto = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_porto, longitude_porto, VERSION,␣
 ↪search_query, radius, LIMIT)
results_porto = requests.get(url_porto).json()
venues_porto = results_porto['response']['venues']
df_porto = json_normalize(venues_porto)
df_porto['City'] = 'Porto'

#Foursquare data from Estádio da Luz, Lisbon
address_lisbon = 'Estádio da Luz'
location_lisbon = geolocator.geocode(address_lisbon)
latitude_lisbon = location_lisbon.latitude
longitude_lisbon = location_lisbon.longitude
url_lisbon = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_lisbon, longitude_lisbon, VERSION,␣
 ↪search_query, radius, LIMIT)
results_lisbon = requests.get(url_lisbon).json()
venues_lisbon = results_lisbon['response']['venues']
df_lisbon = json_normalize(venues_lisbon)
df_lisbon['City'] = 'Lisbon'

#Foursquare data from Olimpiyskiy National Sports Complex, Kyiv
address_kyiv = 'Olimpiyskiy National Sports Complex'
location_kyiv = geolocator.geocode(address_kyiv)
latitude_kyiv = location_kyiv.latitude
longitude_kyiv = location_kyiv.longitude
url_kyiv = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_kyiv, longitude_kyiv, VERSION,␣
 ↪search_query, radius, LIMIT)
results_kyiv = requests.get(url_kyiv).json()
venues_kyiv = results_kyiv['response']['venues']
df_kyiv = json_normalize(venues_kyiv)
df_kyiv['City'] = 'Kyiv'

#Foursquare data from Otkritie Arena, Moscow
address_moscow = 'Otkritie Arena'
location_moscow = geolocator.geocode(address_moscow)
latitude_moscow = location_moscow.latitude
longitude_moscow = location_moscow.longitude
```

```
url_moscow = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_moscow, longitude_moscow, VERSION,␣
 ↪search_query, radius, LIMIT)
results_moscow = requests.get(url_moscow).json()
venues_moscow = results_moscow['response']['venues']
df_moscow = json_normalize(venues_moscow)
df_moscow['City'] = 'Moscow'

#Foursquare data from King Baudouin Stadium, Brussels
address_brussels = 'King Baudouin Stadium'
location_brussels = geolocator.geocode(address_brussels)
latitude_brussels = location_brussels.latitude
longitude_brussels = location_brussels.longitude
url_brussels = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_brussels, longitude_brussels,␣
 ↪VERSION, search_query, radius, LIMIT)
results_brussels = requests.get(url_brussels).json()
venues_brussels = results_brussels['response']['venues']
df_brussels = json_normalize(venues_brussels)
df_brussels['City'] = 'Brussels'

#Foursquare data from Stadion Letna, Prague
address_prague = 'Stadion Letna'
location_prague = geolocator.geocode(address_prague)
latitude_prague = location_prague.latitude
longitude_prague = location_prague.longitude
url_prague = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_prague, longitude_prague, VERSION,␣
 ↪search_query, radius, LIMIT)
results_prague = requests.get(url_prague).json()
venues_prague = results_prague['response']['venues']
df_prague = json_normalize(venues_prague)
df_prague['City'] = 'Prague'

#Foursquare data from National Stadium Warsaw, Warsaw
address_warsaw = 'National Stadium Warsaw'
location_warsaw = geolocator.geocode(address_warsaw)
latitude_warsaw = location_warsaw.latitude
longitude_warsaw = location_warsaw.longitude
url_warsaw = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_warsaw, longitude_warsaw, VERSION,␣
 ↪search_query, radius, LIMIT)
results_warsaw = requests.get(url_warsaw).json()
```

```
venues_warsaw = results_warsaw['response']['venues']
df_warsaw = json_normalize(venues_warsaw)
df_warsaw['City'] = 'Warsaw'

#Foursquare data from Stadion Letzigrund, Zurich
address_zurich = 'Stadion Letzigrund'
location_zurich = geolocator.geocode(address_zurich)
latitude_zurich = location_zurich.latitude
longitude_zurich = location_zurich.longitude
url_zurich = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_zurich, longitude_zurich, VERSION,␣
 ↪search_query, radius, LIMIT)
results_zurich = requests.get(url_zurich).json()
venues_zurich = results_zurich['response']['venues']
df_zurich = json_normalize(venues_zurich)
df_zurich['City'] = 'Zurich'

#Foursquare data from Ernst Happel Stadium, Vienna
address_vienna = 'Ernst Happel Stadium'
location_vienna = geolocator.geocode(address_vienna)
latitude_vienna = location_vienna.latitude
longitude_vienna = location_vienna.longitude
url_vienna = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_vienna, longitude_vienna, VERSION,␣
 ↪search_query, radius, LIMIT)
results_vienna = requests.get(url_vienna).json()
venues_vienna = results_vienna['response']['venues']
df_vienna = json_normalize(venues_vienna)
df_vienna['City'] = 'Vienna'

#Foursquare data from Athens Olympic Stadium, Athens
address_athens = 'Athens Olympic Stadium'
location_athens = geolocator.geocode(address_athens)
latitude_athens = location_athens.latitude
longitude_athens = location_athens.longitude
url_athens = 'https://api.foursquare.com/v2/venues/search?
 ↪client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.
 ↪format(CLIENT_ID, CLIENT_SECRET, latitude_athens, longitude_athens, VERSION,␣
 ↪search_query, radius, LIMIT)
results_athens = requests.get(url_athens).json()
venues_athens = results_athens['response']['venues']
df_athens = json_normalize(venues_athens)
df_athens['City'] = 'Athens'
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
```

```
packages/ipykernel_launcher.py:9: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
  if __name__ == '__main__':
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:20: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:31: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:42: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:53: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:64: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:75: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:86: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:97: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:108: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:119: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:130: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:141: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:152: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:163: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:174: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:185: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:196: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:207: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:218: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:229: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:240: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:251: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:262: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:273: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:284: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:295: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:306: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:317: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:328: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
```

[4]:
```
#List of all dataframe
```

```
city_list = [df_milan, df_turin, df_rome, df_naples, df_munich, df_leipzig,␣
 ↪df_hamburg, df_berlin, df_paris, df_florence, df_toulouse, df_london,␣
 ↪df_liverpool, df_manchester, df_saint_petersburg, df_barcelona, df_madrid,␣
 ↪df_sevilla, df_amsterdam, df_porto, df_lisbon, df_kyiv, df_moscow,␣
 ↪df_brussels, df_prague, df_warsaw, df_zurich, df_vienna, df_athens,␣
 ↪df_zagreb]
#Extract restaurant of the shortest distance from stadium in each city
for i in range(0, len(city_list)):
    city_list[i] = city_list[i][city_list[i]['location.distance'].
 ↪eq(city_list[i]['location.distance'].min())]
```

[5]:
```
#Append all dataframe from all cities
df_foursquare = pd.DataFrame([])
for i in range(0, len(city_list)):
    df_foursquare = df_foursquare.append(city_list[i])
```

[6]:
```
#Import population by European cities
r1 = requests.get('https://worldpopulationreview.com/continents/
 ↪cities-in-europe/')
df_population = pd.read_html(r1.text)
df_population = df_population[0]
#Filter to 30 cities of interest
city_name =␣
 ↪['Milan','Turin','Rome','Naples','Munich','Leipzig','Hamburg','Berlin','Paris','Toulouse','
 ↪Petersburg','Barcelona','Madrid','Sevilla','Amsterdam','Porto','Lisbon','Kyiv','Moscow','Br
df_population = df_population.loc[df_population['Name'].isin(city_name)]
df_population = df_population.rename(columns = {'Name':'City'})
```

[7]:
```
#Merge df_foursquare and df_population
df = df_foursquare.merge(df_population, on='City', how = 'inner')
```

[8]:
```
#Import property price by European cities
r2 = requests.get('https://www.numbeo.com/property-investment/
 ↪region_rankings_current.jsp?region=150')
df_propertyprice = pd.read_html(r2.text)
df_propertyprice = df_propertyprice[2]
```

[9]:
```
#Import crime index by European cities
r3 = requests.get('https://www.numbeo.com/crime/region_rankings_current.jsp?
 ↪region=150')
df_crimeindex = pd.read_html(r3.text)
df_crimeindex = df_crimeindex[2]
```

[10]:
```
#Import average monthly salary by European cities
r4 = requests.get('https://www.numbeo.com/cost-of-living/region_prices_by_city?
 ↪itemId=105&region=150')
```

```
df_salary = pd.read_html(r4.text)
df_salary = df_salary[2]
```

[11]:
```
#Import cost of living by European cities
r5 = requests.get('https://www.numbeo.com/cost-of-living/
 ↪region_rankings_current.jsp?region=150')
df_costofliving = pd.read_html(r5.text)
df_costofliving = df_costofliving[2]
```

[12]:
```
#Merge Numbeo dataframes
df_numbeo = df_propertyprice.merge(df_crimeindex, on='City', how = 'inner').
 ↪merge(df_salary, on='City', how = 'inner').merge(df_costofliving, on='City',␣
 ↪how = 'inner')
#Split column into city and country
df_numbeo[['City','Country']] = df_numbeo['City'].str.split(', ',expand=True)
```

[13]:
```
#Replace city name to match other dataframe
df_numbeo.City = df_numbeo.City.replace({"Seville (Sevilla)": "Sevilla", "Kiev␣
 ↪(Kyiv)": "Kyiv"})
```

[14]:
```
#Filter to 30 cities of interest
df_numbeo = df_numbeo.loc[df_numbeo['City'].isin(city_name)]
```

[15]:
```
#Merge Foursquare, population and Numbeo data
df_merge = df.merge(df_numbeo, on='City', how = 'inner')
df_merge.shape
```

[15]:
```
(30, 43)
```

[16]:
```
#Reduce columns and take reciprocal of salary, rent index, cost of living,␣
 ↪grocery index, restaurant price index
df_final = df_merge[['City', 'Country_y', 'location.lat', 'location.lng', '2020␣
 ↪Population', 'Safety Index', 'location.distance', 'Local Purchasing Power␣
 ↪Index', 'Gross Rental Yield City Centre', 'Gross Rental Yield Outside of␣
 ↪Centre', 'Average Monthly Net Salary (After Tax)', 'Rent Index', 'Cost of␣
 ↪Living Index', 'Groceries Index', 'Restaurant Price Index', 'Crime Index']]
df_final['1/Salary'] = 1/df_merge['Average Monthly Net Salary (After Tax)']
df_final['1/Rent Index'] = 1/df_merge['Rent Index']
df_final['1/Cost of Living'] = 1/df_merge['Cost of Living Index']
df_final['1/Groceries Index'] = 1/df_merge['Groceries Index']
df_final['1/Restaurant Price Index'] = 1/df_merge['Restaurant Price Index']
df_final['1/Crime Index'] = 1/df_merge['Crime Index']
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports
until
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[17]:
```
from sklearn.cluster import KMeans
from sklearn import preprocessing
df_reduced = df_final.drop(['City', 'Country_y', 'location.lat', 'location.
 ↪lng'], axis=1)
X = np.asarray(df_reduced)
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
kclusters = 5
k_means = KMeans(init = "k-means++", n_clusters = kclusters, n_init = 24)
k_means.fit(X)
k_means_labels = k_means.labels_
df_reduced.columns
```

[17]:
```
Index(['2020 Population', 'Safety Index', 'location.distance',
       'Local Purchasing Power Index', 'Gross Rental Yield City Centre',
       'Gross Rental Yield Outside of Centre',
       'Average Monthly Net Salary (After Tax)', 'Rent Index',
       'Cost of Living Index', 'Groceries Index', 'Restaurant Price Index',
       'Crime Index', '1/Salary', '1/Rent Index', '1/Cost of Living',
       '1/Groceries Index', '1/Restaurant Price Index', '1/Crime Index'],
      dtype='object')
```

[18]:
```
#Insert decision class to dataframe
df_final.insert(0, 'Decision Class', k_means.labels_)
#Average value for each attribute by decision class
df_avg_by_class = df_final.groupby(['Decision Class']).mean()
print(df_final[['Decision Class','City']])
```

```
    Decision Class             City
0                2           Milan
1                0           Turin
2                0            Rome
3                0          Naples
4                2          Munich
5                0         Leipzig
6                2         Hamburg
7                2          Berlin
8                2           Paris
9                2        Florence
10               0        Toulouse
11               2          London
12               0       Liverpool
13               0      Manchester
14               1  Saint Petersburg
15               0       Barcelona
16               2          Madrid
17               3         Sevilla
18               2       Amsterdam
```

```
19              3             Porto
20              3            Lisbon
21              1              Kyiv
22              1            Moscow
23              0          Brussels
24              3            Prague
25              3            Warsaw
26              4            Zurich
27              2            Vienna
28              0            Athens
29              3            Zagreb
```

[19]:
```
import matplotlib.cm as cm
import matplotlib.colors as colors
from folium.features import DivIcon
# Create map
map_clusters = folium.Map(location=[df['location.lat'].mean(),␣
 ↪df_final['location.lng'].mean()], zoom_start=3.5)
# Set color scheme for the clusters
x = np.arange(kclusters)
ys = [i + x + (i*x)**2 for i in range(kclusters)]
colors_array = cm.rainbow(np.linspace(0, 1, len(ys)))
rainbow = [colors.rgb2hex(i) for i in colors_array]

for lat, lon, Class, City in zip(df_final['location.lat'], df_final['location.
 ↪lng'], df_final['Decision Class'], df_final['City']):
    label = '{}, {} {}'.format(City, 'Class', Class)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lon],
        radius=5,
        popup=label,
        color=rainbow[Class],
        fill=True,
        fill_color=rainbow[Class],
        fill_opacity=0.7).add_to(map_clusters)
map_clusters
```

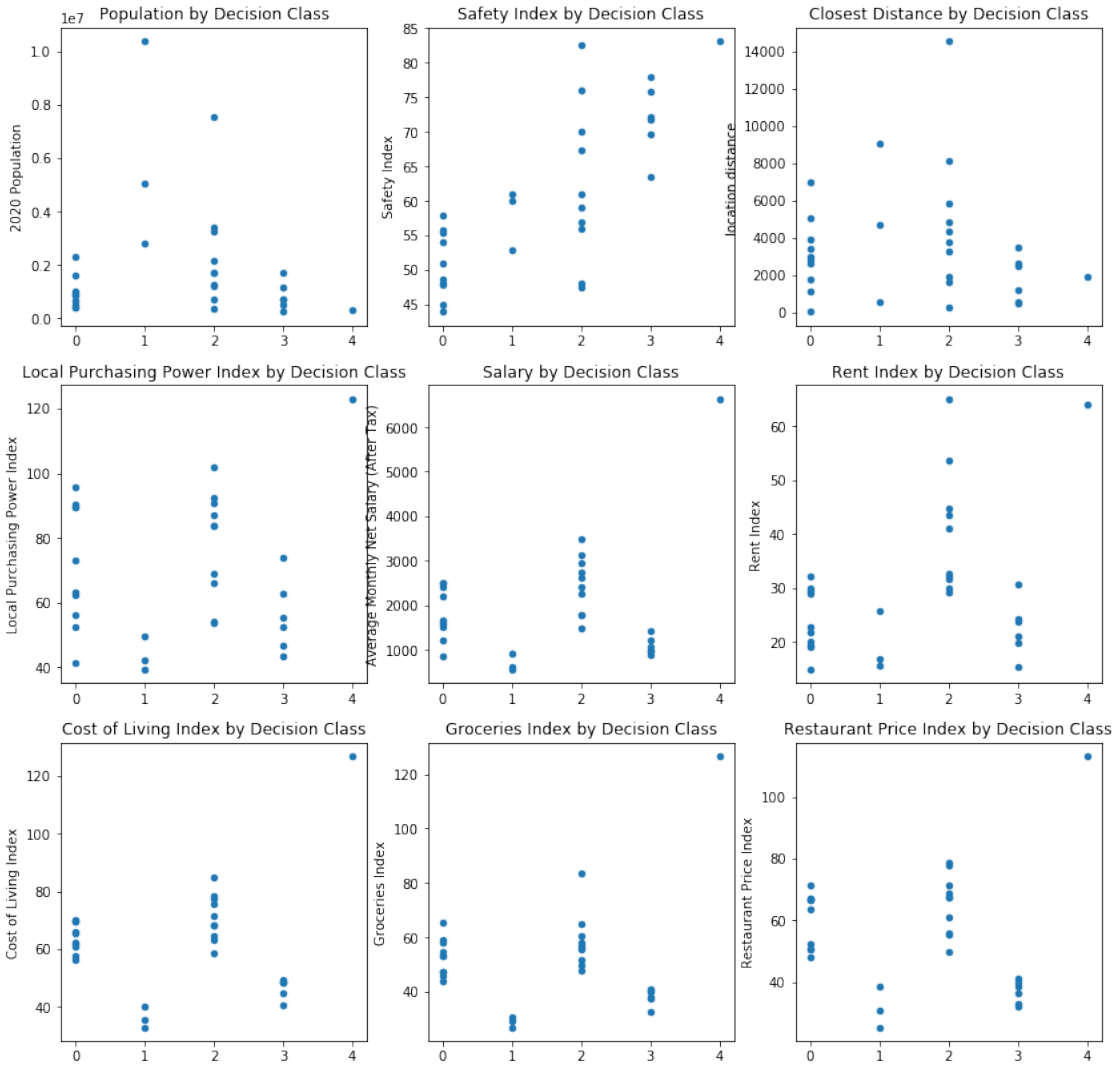
[19]:
```
<folium.folium.Map at 0x7f2e34a48550>
```

[20]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
fig = plt.figure()
ax0 = fig.add_subplot(3, 3, 1) # add subplot 1 (3 row, 2 columns, first plot)
ax1 = fig.add_subplot(3, 3, 2) # add subplot 2 (3 row, 2 columns, second plot)
ax2 = fig.add_subplot(3, 3, 3) # add subplot 2 (3 row, 2 columns, third plot)
ax3 = fig.add_subplot(3, 3, 4) # add subplot 1 (3 row, 2 columns, fourth plot)
```

```
ax4 = fig.add_subplot(3, 3, 5) # add subplot 2 (3 row, 2 columns, fifth plot)
ax5 = fig.add_subplot(3, 3, 6) # add subplot 2 (3 row, 2 columns, sixth plot)
ax6 = fig.add_subplot(3, 3, 7) # add subplot 1 (3 row, 2 columns, seventh plot)
ax7 = fig.add_subplot(3, 3, 8) # add subplot 2 (3 row, 2 columns, eighth plot)
ax8 = fig.add_subplot(3, 3, 9) # add subplot 2 (3 row, 2 columns, ninth plot)

df_final.plot(kind='scatter', x = 'Decision Class', y = '2020 Population',␣
 ↪figsize=(14, 14), ax=ax0)
ax0.set_title('Population by Decision Class')
ax0.set_xlabel('')

df_final.plot(kind='scatter', x = 'Decision Class', y = 'Safety Index',␣
 ↪figsize=(14, 14), ax=ax1)
ax1.set_title('Safety Index by Decision Class')
ax1.set_xlabel('')

df_final.plot(kind='scatter', x = 'Decision Class', y = 'location.distance',␣
 ↪figsize=(14, 14), ax=ax2)
ax2.set_title('Closest Distance by Decision Class')
ax2.set_xlabel('')

df_final.plot(kind='scatter', x = 'Decision Class', y = 'Local Purchasing Power␣
 ↪Index', figsize=(14, 14), ax=ax3)
ax3.set_title('Local Purchasing Power Index by Decision Class')
ax3.set_xlabel('')

df_final.plot(kind='scatter', x = 'Decision Class', y = 'Average Monthly Net␣
 ↪Salary (After Tax)', figsize=(14, 14), ax=ax4)
ax4.set_title('Salary by Decision Class')
ax4.set_xlabel('')

df_final.plot(kind='scatter', x = 'Decision Class', y = 'Rent Index',␣
 ↪figsize=(14, 14), ax=ax5)
ax5.set_title('Rent Index by Decision Class')
ax5.set_xlabel('')

df_final.plot(kind='scatter', x = 'Decision Class', y = 'Cost of Living Index',␣
 ↪figsize=(14, 14), ax=ax6)
ax6.set_title('Cost of Living Index by Decision Class')
ax6.set_xlabel('')

df_final.plot(kind='scatter', x = 'Decision Class', y = 'Groceries Index',␣
 ↪figsize=(14, 14), ax=ax7)
ax7.set_title('Groceries Index by Decision Class')
ax7.set_xlabel('')
```

```
df_final.plot(kind='scatter', x = 'Decision Class', y = 'Restaurant Price␣
↪Index', figsize=(14, 14), ax=ax8)
ax8.set_title('Restaurant Price Index by Decision Class')
ax8.set_xlabel('')
```

[20]: Text(0.5, 0, '')

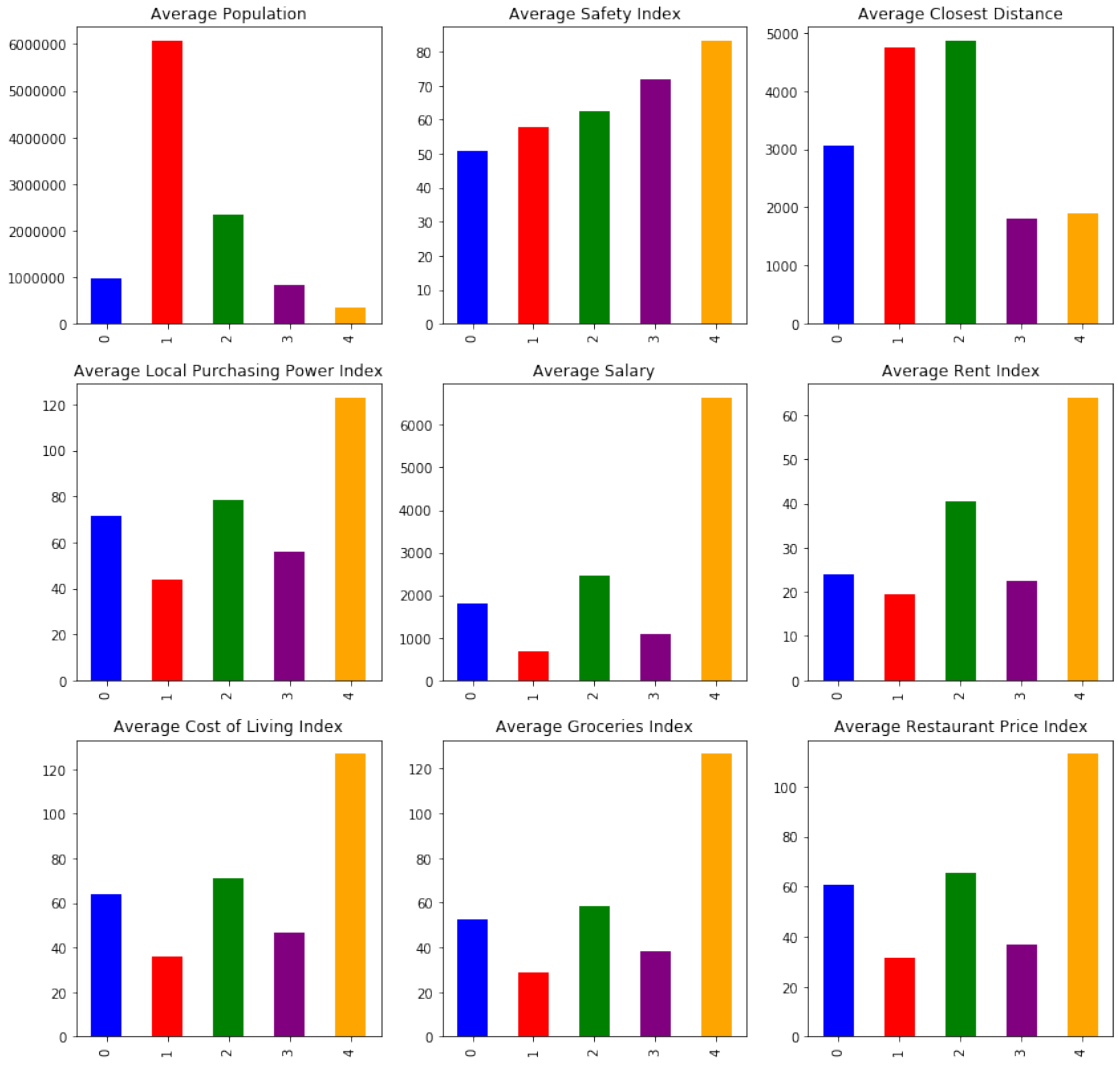



```
[21]: fig = plt.figure()
ax0 = fig.add_subplot(3, 3, 1)
ax1 = fig.add_subplot(3, 3, 2)
ax2 = fig.add_subplot(3, 3, 3)
ax3 = fig.add_subplot(3, 3, 4)
ax4 = fig.add_subplot(3, 3, 5)
ax5 = fig.add_subplot(3, 3, 6)
```

```
ax6 = fig.add_subplot(3, 3, 7)
ax7 = fig.add_subplot(3, 3, 8)
ax8 = fig.add_subplot(3, 3, 9)

df_avg_by_class['2020 Population'].plot(kind='bar', color = ['blue', 'red',␣
↪'green', 'purple', 'orange'], figsize=(14, 14), ax=ax0)
ax0.set_title('Average Population')
ax0.set_xlabel('')

df_avg_by_class['Safety Index'].plot(kind='bar', color = ['blue', 'red',␣
↪'green', 'purple', 'orange'], figsize=(14, 14), ax=ax1)
ax1.set_title('Average Safety Index')
ax1.set_xlabel('')

df_avg_by_class['location.distance'].plot(kind='bar', color = ['blue', 'red',␣
↪'green', 'purple', 'orange', 'cyan'], figsize=(14, 14), ax=ax2)
ax2.set_title('Average Closest Distance')
ax2.set_xlabel('')

df_avg_by_class['Local Purchasing Power Index'].plot(kind='bar', color =␣
↪['blue', 'red', 'green', 'purple', 'orange', 'cyan'], figsize=(14, 14),␣
↪ax=ax3)
ax3.set_title('Average Local Purchasing Power Index')
ax3.set_xlabel('')

df_avg_by_class['Average Monthly Net Salary (After Tax)'].plot(kind='bar',␣
↪color = ['blue', 'red', 'green', 'purple', 'orange', 'cyan'], figsize=(14,␣
↪14), ax=ax4)
ax4.set_title('Average Salary')
ax4.set_xlabel('')

df_avg_by_class['Rent Index'].plot(kind='bar', color = ['blue', 'red', 'green',␣
↪'purple', 'orange', 'cyan'], figsize=(14, 14), ax=ax5)
ax5.set_title('Average Rent Index')
ax5.set_xlabel('')

df_avg_by_class['Cost of Living Index'].plot(kind='bar', color = ['blue',␣
↪'red', 'green', 'purple', 'orange', 'cyan'], figsize=(14, 14), ax=ax6)
ax6.set_title('Average Cost of Living Index')
ax6.set_xlabel('')

df_avg_by_class['Groceries Index'].plot(kind='bar', color = ['blue', 'red',␣
↪'green', 'purple', 'orange', 'cyan'], figsize=(14, 14), ax=ax7)
ax7.set_title('Average Groceries Index')
ax7.set_xlabel('')
```

```
df_avg_by_class['Restaurant Price Index'].plot(kind='bar', color = ['blue',␣
 ↪'red', 'green', 'purple', 'orange', 'cyan'], figsize=(14, 14), ax=ax8)
ax8.set_title('Average Restaurant Price Index')
ax8.set_xlabel('')
```

[21]: Text(0.5, 0, '')



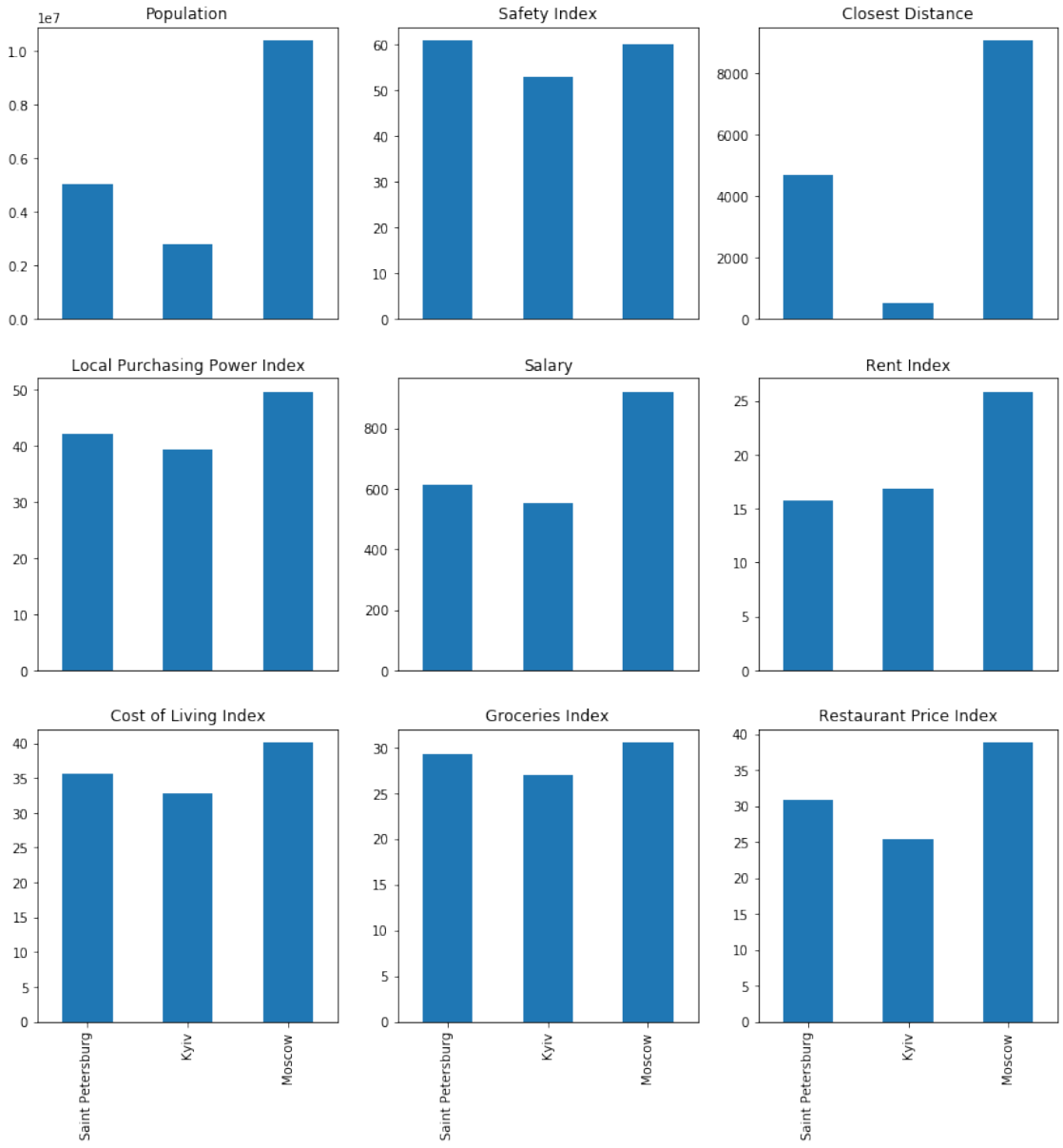
[25]:
```
df_eastern_europe = df_final[(df_final['Decision Class'] ==1)]
```

[26]:
```
fig = plt.figure()
ax0 = fig.add_subplot(3, 3, 1)
ax1 = fig.add_subplot(3, 3, 2)
ax2 = fig.add_subplot(3, 3, 3)
ax3 = fig.add_subplot(3, 3, 4)
```

```
ax4 = fig.add_subplot(3, 3, 5)
ax5 = fig.add_subplot(3, 3, 6)
ax6 = fig.add_subplot(3, 3, 7)
ax7 = fig.add_subplot(3, 3, 8)
ax8 = fig.add_subplot(3, 3, 9)

df_eastern_europe.plot(x = 'City', y = '2020 Population', kind='bar',␣
↪figsize=(14, 14), legend = False, ax=ax0)
ax0.set_title('Population')
ax0.axes.get_xaxis().set_visible(False)

df_eastern_europe.plot(x = 'City', y = 'Safety Index', kind='bar', figsize=(14,␣
↪14), legend = False, ax=ax1)
ax1.set_title('Safety Index')
ax1.axes.get_xaxis().set_visible(False)

df_eastern_europe.plot(x = 'City', y = 'location.distance', kind='bar',␣
↪figsize=(14, 14), legend = False, ax=ax2)
ax2.set_title('Closest Distance')
ax2.axes.get_xaxis().set_visible(False)

df_eastern_europe.plot(x = 'City', y = 'Local Purchasing Power Index',␣
↪kind='bar', figsize=(14, 14), legend = False, ax=ax3)
ax3.set_title('Local Purchasing Power Index')
ax3.axes.get_xaxis().set_visible(False)

df_eastern_europe.plot(x = 'City', y = 'Average Monthly Net Salary (After␣
↪Tax)', kind='bar', figsize=(14, 14), legend = False, ax=ax4)
ax4.set_title('Salary')
ax4.axes.get_xaxis().set_visible(False)

df_eastern_europe.plot(x = 'City', y = 'Rent Index', kind='bar', figsize=(14,␣
↪14), legend = False, ax=ax5)
ax5.set_title('Rent Index')
ax5.axes.get_xaxis().set_visible(False)

df_eastern_europe.plot(x = 'City', y = 'Cost of Living Index', kind='bar',␣
↪figsize=(14, 14), legend = False, ax=ax6)
ax6.set_title('Cost of Living Index')
ax6.set_xlabel('')

df_eastern_europe.plot(x = 'City', y = 'Groceries Index', kind='bar',␣
↪figsize=(14, 14), legend = False, ax=ax7)
ax7.set_title('Groceries Index')
ax7.set_xlabel('')
```

```
df_eastern_europe.plot(x = 'City', y = 'Restaurant Price Index', kind='bar',␣
 ↪figsize=(14, 14), legend = False, ax=ax8)
ax8.set_title('Restaurant Price Index')
ax8.set_xlabel('')
```

[26]: Text(0.5, 0, '')



[ ]: