Capstone Project

Relocation Support

Yash Pratap Singh

April 2024

Introduction

This notebook contains the Capstone Project as the Week 5 peer-graded assignment for the Course IBM Applied Data Science Capstone on Coursera, which requires to develop a solution for the problem proposed by the learner in his Project Proposal Week 4, the stablished condition requires to make use of **API Foursquare** to solve the proposed problem.

The proposed problem for this project is to support people to relocate giving them directions by creating a districts, or neighborhoods, rank for the target location using the relocator profile.

Step 1 - Define the *Relocator Profile* and *Target Location*

For this project the **Relocator Profile** will be arbitrary defined to simulate a family relocating to also arbitrary defined **Target Location São Paulo, Brazil**.

Family components;

- 2 adults
- 2 kids in primary school age
- 1 dog as pet

Family priorities;

- **Primary school** for the kids
- **Outdoor park** to go with the kids, walk with the dog and jogging
- **Supermarket** for the daily life
- **Pharmacy** in case of emergencies, specially having kids
- **Subway or metro station** to avoid traffic

Housing wishes;

- Apartment
- 2 or 3 bedrooms

- 80 m^2 approximately
- 1 garage spot

Rental budget;

- BRL 2,000.00 / month

Based on the Relocator Profile, define objects to be used to rank the districts of the target location.

Family priorities will be translated into a *dictionary* containing venues categories according to the **API Foursquare**, the *dictionary keys* will be **CategoryId** from **API Foursquare** and *dictionary values* the categories names. Housing wishes and Rental budget will be converted in a *variable* containg the rental budget per square meter - BRL/m^2. Enabling some flexibility and opening more options to the Relocator, a tolerance for rental prices will be set to +-5%.

In [1]:
```
# create priorities list according to API Foursquare categories {'CategoryId': 'Catego
prio_list = {'4f4533804b9074f6e4fb0105': 'Elementary School',
             '4bf58dd8d48988d163941735': 'Park',
             '52f2ab2ebcbc57f1066b8b46': 'Supermarket',
             '4bf58dd8d48988d10f951735': 'Pharmacy',
             '4bf58dd8d48988d1fd931735': 'Metro Station'}

# calculate rental budget per square meter based in target area and budget
target_area = 80
target_budget = 2000
tolera_budget = 0.05

budget_sqm = target_budget / target_area
budget_sqm_min = budget_sqm * (1 - tolera_budget)
budget_sqm_max = budget_sqm * (1 + tolera_budget)

# print objects
print('Family priorities are:', list(prio_list.values()), '\n')
print('Monthly rental budget is BRL {:,.2f} for a {} sqm apartment = BRL {:,.2f}/sqm/m
      format(target_budget, target_area, budget_sqm))
print('Rental prices will be considered ranging from {:,.2f}/sqm/month up to {:,.2f}/s
      format(budget_sqm_min, budget_sqm_max))
```

```
Family priorities are: ['Elementary School', 'Park', 'Supermarket', 'Pharmacy', 'Metr
o Station']

Monthly rental budget is BRL 2,000.00 for a 80 sqm apartment = BRL 25.00/sqm/month

Rental prices will be considered ranging from 23.75/sqm/month up to 26.25/sqm/month
```

Step 2 - Acquire Districts Data of *Target Location*

The city of São Paulo is divided in 5 geographical regions, 32 boroughs, 96 districts and hundreds of neighborhoods. The city division up to the district level can be found in the official city web site, its is stored in a XLSX table format into a HTML page. The project will be based on

district level to create the rank. The current table is from 2017 and can be viewed at the following link;

Prefeitura de São Paulo (SP) Regiões, Prefeituras Regionas e Distritos

To create a dataframe for the Districts of São Paulo, a request to the above URL will be done, its content parsed, cleaned and organized, to be stored in so called **Districts Dataframe**.

First thing, import required libraries.

1. **Pandas**: manipulate dataframe objects
2. **Numpy**: manipulate arrays and matrices calculations
3. **Requests**: send and receive url requests
4. **BeautifulSoup**: parse url content

In [2]:
```
import pandas as pd
import numpy as np
import requests
from bs4 import BeautifulSoup as bs
```

Send request to the above URL using *requests.get* method.

In [3]:
```
# define a variable for the url
url = 'http://www.prefeitura.sp.gov.br/cidade/secretarias/upload/urbanismo/infocidade/

# send a request to the URL and store the response
raw = requests.get(url)

# check if data was loaded [status 200 means success]
if raw:
    print('Data loaded, status', raw.status_code)
else:
    print('Error loading data', raw.status_code)
```

```
Data loaded, status 200
```

Parse the URL content using *BeautifulSoup* with *HTML* parser.

In [4]:
```
# parse the raw data
par = bs(raw.text, 'html.parser')
print('URL content parsed.')
```

```
URL content parsed.
```

Check the returned content. Initially, how many tables it contains, the HTML tag *table* will be the reference for counting.

In [5]:
```
# print number of tables
print('{} table(s) found in the parsed URL.'.format(len(par.find_all('table'))))
```

```
1 table(s) found in the parsed URL.
```

As it is a table in *XLSX* format, all its content is stored in one unique table, including table description, headers, data, summary and footnotes. The data should be extract from this unique

table, but, first get the table out of the parsed content, HTML tag *table* will be the reference as well.

In [6]:
```
# get the table out of the parsed content
par_table = par.find_all('table')[0]
```

Check if there is any table header and how many rows the table contains. HTML tags *th* and *tr* will be respectively used for counting.

In [7]:
```
# print the number of headers and rows
print('{} header(s) and {} row(s) found in the table.'.format(len(par_table.find_all('
                                                                 len(par_table.find_all('
```

```
0 header(s) and 110 row(s) found in the table.
```

From the 110 rows, 96 of them should be the required districts data, the remaining rows should be titles, summaries and footnotes rows.

Extract the data from the table to a list and check where the required data is located. This will be done running a nested loop through the rows, on the first level, and columns, on the second level. HTML tag *tr* will be used to extract rows and *td* to extract the columns.

In [8]:
```
# create an empty list for the entire table
tabletmp = []

# run a loop by row [tag 'tr']
for i, row in enumerate(par_table.find_all('tr')):
    # create an empty list for the current row
    celltmp = []

    # run a loop by column for the current row [tag 'td']
    for j, column in enumerate(row.find_all('td')):
        # append the text of current cell to the list
        celltmp.append(column.get_text())

    # append current line to the list
    tabletmp.append(celltmp)

# inform the number of rows loaded
print('{} rows loaded.'.format(len(tabletmp)))
```

```
110 rows loaded.
```

All the 110 rows have been loaded. Check the head and tail of the list to define the range of required data.

In [9]:
```
# print the first 10 rows
print('Head 10 rows')
tabletmp[:10]
```

```
Head 10 rows
```

Out[9]:
```
[['Regiões, Prefeituras Regionais e Distritos Municipais',
  '\xa0',
  '',
  '',
  '',
  '',
  ''],
 ['Município de São Paulo', '\xa0', '', '', '', ''],
 ['2017', '\xa0', '\xa0', '', '', '', ''],
 ['', '', '', '', '', ''],
 ['Regiões',
  'Prefeituras\r\n  Regionais',
  'Distritos',
  'Área (ha)',
  'Área (km²)',
  '',
  '',
  ''],
 ['Centro', 'Sé', 'Bela Vista', '271,77', '2,72', '', '', ''],
 ['Bom Retiro', '420,54', '4,21', '', '', ''],
 ['Cambuci', '392,42', '3,92', '', '', ''],
 ['Consolação', '381,51', '3,82', '', '', ''],
 ['Liberdade', '365,07', '3,65', '', '', '']]
```

The first 4 rows are the table titles and can be ignored, row 5 contains the columns headers.

Check the tail of the table.

In [10]:
```
#print the last 10 rows
print('Tail 10 rows')
tabletmp[-10:]
```

Out[10]:
```
Tail 10 rows
[['Vila Mariana', '859,56', '8,60', '', '', '', ''],
 ['Município de São\r\n  Paulo', '', '152.753,58', '1.527,54', '', '', '', ''],
 ['', '', '', '', '', '', '', ''],
 ['Fonte:\r\n  Prefeitura do Município de São Paulo. /\xa0\r\n  Instituto\xa0 Geográf
ico e\r\n  Cartográfico\xa0 do Estado de São Paulo.',
  '\xa0',
  '',
  '',
  '\xa0'],
 ['Elaboração:\r\n  SMUL/Deinfo', '\xa0', '\xa0', '', '', '\xa0'],
 ['Nota: Distritos Lei\r\n  nº 11.220/1992', '', '', ''],
 ['Subprefeituras\r\n  Lei nº 13.399/2002, alterada pelas Leis nº 13.682/2003 e nº 1
5.764/2013',
  '',
  '',
  '',
  ''],
 ['Base\r\n  de cálculo das áreas: Mapa Digital da Cidade (MDC) - UTM/SAD69-96.',
  '',
  '',
  '',
  ''],
 ['', '', '', '', '', '', '', ''],
 ['', '', '', '', '', '', '', '', '']]
```

The last 8 rows are the columns summary and footnotes, will also be ignored.

Check the columns headers, row 5 (index 4).

In [11]:
```
# check the columns headers
tabletmp[4]
```

Out[11]:
```
['Regiões',
 'Prefeituras\r\n  Regionais',
 'Distritos',
 'Área (ha)',
 'Área (km²)',
 '',
 '',
 '']
```

There are 8 columns, but the last 3 are empty, the first 5 are the following:

- **Region** (Regiões), the first column is the city Region to which the District belongs
- **Borough** (Prefeituras Regionais), the second column is the Borough location
- **District** (Distritos), the third column is the District name
- Area ha (Área (ha)), the fourth column is the neighborhood land area in hectares, it is not relevant for this project and it will be ignored
- **Area km^2** (Área (km^2)), the fifth column is the neighborhood land area in square kilometers

Define the columns names for the **Districts Dataframe**.

In [12]:
```
# define the columns names
column_names = ['region', 'borough', 'district', 'area_sqkm']
```

At the top are 4 title rows and 1 header row, the first 5 rows at the top will be ignored. At the bottom 8 rows will be ignored, 1 summary and 7 footnotes rows. District data ranges from row 6 (index 5) to row 101 (index 100), counting 96 districts in São Paulo.

Extract only the data rows from the table and count the number of rows left.

In [13]:
```
# extract only required rows
tabletmp = tabletmp[5:101]

# print number rows left
print('{} rows left.'.format(len(tabletmp)))
```

```
96 rows left.
```

Before extracting the data, check the head and tail to see how the data is stored.

In [14]:
```
# print the first 10 rows
print('Head 10 rows')
tabletmp[:10]
```

```
Head 10 rows
```

Out[14]:
```
[['Centro', 'Sé', 'Bela Vista', '271,77', '2,72', '', '', ''],
 ['Bom Retiro', '420,54', '4,21', '', '', ''],
 ['Cambuci', '392,42', '3,92', '', '', ''],
 ['Consolação', '381,51', '3,82', '', '', ''],
 ['Liberdade', '365,07', '3,65', '', '', ''],
 ['República', '239,67', '2,40', '', '', ''],
 ['Santa Cecília', '375,92', '3,76', '', '', ''],
 ['Sé', '219,36', '2,19', '', '', ''],
 ['Leste',
  'Aricanduva/Formosa/Carrão',
  'Aricanduva',
  '695,83',
  '6,96',
  '',
  '',
  ''],
 ['Carrão', '790,12', '7,90', '', '', '']]
```

Check the tail.

In [15]:
```
#print the last 10 rows
print('Tail 10 rows')
tabletmp[-10:]
```

Out[15]:
```
Tail 10 rows
[["M'Boi Mirim", 'Jardim Ângela', '3.741,13', '37,41', '', '', ''],
 ['Jardim São Luís', '2.604,72', '26,05', '', '', ''],
 ['Parelheiros', 'Marsilac', '20.818,52', '208,19', '', '', ''],
 ['Parelheiros', '15.260,75', '152,61', '', '', ''],
 ['Santo Amaro', 'Campo Belo', '876,98', '8,77', '', '', ''],
 ['Campo Grande', '1.295,08', '12,95', '', '', ''],
 ['Santo Amaro', '1.603,53', '16,04', '', '', '', ''],
 ['Vila Mariana', 'Moema', '907,87', '9,08', '', '', '', ''],
 ['Saúde', '931,12', '9,31', '', '', '', ''],
 ['Vila Mariana', '859,56', '8,60', '', '', '', '']]
```

The table seems to be structured in a group format, which is normally done for good visualization in *XLSX* format tables. The region name, e.g. **Centro**, appears only once at its first borough **Sé**, and so the borough **Sé** for its first district **Bela Vista**. There are rows with 5 columns, rows with 4 columns and rows with 3 columns, it makes things interesting.

Something else to notice is that the columns aligned on the left side, in some of the rows the first column contains the region name, in some it contains the borough, and in many of them it contains the district, meaning that the columns are not at the same positions for all rows. To extract the data the reading should consider the number of features each row contains, and for rows missing region and/or borough information, they should be added, even more interesting.

The last notice here is that the last columns are empty, for some rows 3 and for some 4 empty columns, they will be ignored.

Extract the data to a temporary list, taking all the remarks above into consideration. This will be achieved running a loop through the rows. Actually it is one list object, which represents the source table, its elements are lists as well, each of them representing the rows of the table, and each element of the inner lists represents the cells of the table.

In [16]:
```
# create an empty list to store data temporary
listtmp = []

# run a loop through the rows [list represents the table]
for i, row in enumerate(tabletmp):
    # check quantity of features in the current row [list represents a row]
    nfeatures = len(row)-row.count('')

    # check the number of features
    if nfeatures == 5:
        # five features means complete row with new region and new borough, store them
        vregion = row[0]
        vborough = row[1]
    elif nfeatures == 4:
        # four features means same region and new borough, store it in a variable
        vborough = row[0]

    # 3 features means same region and borough, the variable above will be used to sto

    # append the current row to the temporary list
    listtmp.append([vregion, vborough, row[nfeatures-3], row[nfeatures-1]])
```

Check the resulting list, head and tail.

In [17]:
```
# check results
print('Head 10 rows')
listtmp[:10]
```

Out[17]:
```
Head 10 rows
[['Centro', 'Sé', 'Bela Vista', '2,72'],
 ['Centro', 'Sé', 'Bom Retiro', '4,21'],
 ['Centro', 'Sé', 'Cambuci', '3,92'],
 ['Centro', 'Sé', 'Consolação', '3,82'],
 ['Centro', 'Sé', 'Liberdade', '3,65'],
 ['Centro', 'Sé', 'República', '2,40'],
 ['Centro', 'Sé', 'Santa Cecília', '3,76'],
 ['Centro', 'Sé', 'Sé', '2,19'],
 ['Leste', 'Aricanduva/Formosa/Carrão', 'Aricanduva', '6,96'],
 ['Leste', 'Aricanduva/Formosa/Carrão', 'Carrão', '7,90']]
```

In [18]:
```
#print results
print('Tail 10 rows')
listtmp[-10:]
```

Out[18]:
```
Tail 10 rows
[['Sul', "M'Boi Mirim", 'Jardim Ângela', '37,41'],
 ['Sul', "M'Boi Mirim", 'Jardim São Luís', '26,05'],
 ['Sul', 'Parelheiros', 'Marsilac', '208,19'],
 ['Sul', 'Parelheiros', 'Parelheiros', '152,61'],
 ['Sul', 'Santo Amaro', 'Campo Belo', '8,77'],
 ['Sul', 'Santo Amaro', 'Campo Grande', '12,95'],
 ['Sul', 'Santo Amaro', 'Santo Amaro', '16,04'],
 ['Sul', 'Vila Mariana', 'Moema', '9,08'],
 ['Sul', 'Vila Mariana', 'Saúde', '9,31'],
 ['Sul', 'Vila Mariana', 'Vila Mariana', '8,60']]
```

The list is ready to be stored in a dataframe. Create the **Districts Dataframe**, using *pandas.DataFrame* method.

In [19]:
```
# create the dataframe
df_districts = pd.DataFrame(data=listtmp, columns=column_names)

# print results
df_districts
```

Out[19]:

	region	borough	district	area_sqkm
0	Centro	Sé	Bela Vista	2,72
1	Centro	Sé	Bom Retiro	4,21
2	Centro	Sé	Cambuci	3,92
3	Centro	Sé	Consolação	3,82
4	Centro	Sé	Liberdade	3,65
...	...	...	...	...
91	Sul	Santo Amaro	Campo Grande	12,95
92	Sul	Santo Amaro	Santo Amaro	16,04
93	Sul	Vila Mariana	Moema	9,08
94	Sul	Vila Mariana	Saúde	9,31
95	Sul	Vila Mariana	Vila Mariana	8,60

96 rows × 4 columns

Data in **area_sqkm** column doesn't have a good fit, as the decimal separator is comma, the Brazilian standard. For this reason it contains strings when it should contain numbers, *float* in this case.

Convert the data type of **area_sqkm** to *float*. This will be done replacing the decimal separator, using *pandas.DataFrame.apply* metohd, and then converting the column data type to *float* in the dataframe using *pandas.dataframe.astype* method.

In [20]:
```
# change decimal separator
df_districts['area_sqkm'] = df_districts['area_sqkm'].apply(lambda x : x.replace(',',

# check the head
df_districts
```

Out[20]:

	region	borough	district	area_sqkm
0	Centro	Sé	Bela Vista	2.72
1	Centro	Sé	Bom Retiro	4.21
2	Centro	Sé	Cambuci	3.92
3	Centro	Sé	Consolação	3.82
4	Centro	Sé	Liberdade	3.65
...	...	...	...	...
91	Sul	Santo Amaro	Campo Grande	12.95
92	Sul	Santo Amaro	Santo Amaro	16.04
93	Sul	Vila Mariana	Moema	9.08
94	Sul	Vila Mariana	Saúde	9.31
95	Sul	Vila Mariana	Vila Mariana	8.60

96 rows × 4 columns

Check the dataframe columns data types before converting, change the data type of **area_sqkm** from *object* to *float*, and check the dataframe columns data types after converting.

In [21]:
```
# print the columns data types
print('df_districts data types before converting:\n', df_districts.dtypes, '\n')

# convert the area column to float
df_districts = df_districts.astype({'area_sqkm': 'float64'})

# print the columns data types
print('df_districts data types after converting:\n', df_districts.dtypes)
```

```
df_districts data types before converting:
 region       object
borough      object
district     object
area_sqkm    object
dtype: object

df_districts data types after converting:
 region        object
borough       object
district      object
area_sqkm    float64
dtype: object
```

Check results.

In [22]:
```
# print results
df_districts
```

Out[22]:

	region	borough	district	area_sqkm
0	Centro	Sé	Bela Vista	2.72
1	Centro	Sé	Bom Retiro	4.21
2	Centro	Sé	Cambuci	3.92
3	Centro	Sé	Consolação	3.82
4	Centro	Sé	Liberdade	3.65
...	...	...	...	...
91	Sul	Santo Amaro	Campo Grande	12.95
92	Sul	Santo Amaro	Santo Amaro	16.04
93	Sul	Vila Mariana	Moema	9.08
94	Sul	Vila Mariana	Saúde	9.31
95	Sul	Vila Mariana	Vila Mariana	8.60

96 rows × 4 columns

Step 3 - Acquire Rental Prices Data of *Target Location*

Searching on the internet, there are several real estate agencies websites in São Paulo, but to find a rental prices list per district or neighborhood is a hard task. A mixed rental prices list per district and neighborhood could be found in a real estate agency website called **Blog SP Imóvel** (www.spimovel.com.br), which provides its services all around the city. It hosts additional four websites covering the regions of the city, as follwing;

- **Blog ZN Imóvel** (www.znimovel.com.br) for region *Norte*
- **Blog ZS Imóvel** (www.zsimovel.com.br) for region *Sul*
- **Blog ZL Imóvel** (www.zlimovel.com.br) for region *Leste*
- **Blog ZO Imóvel** (www.zoimovel.com.br) for region *Oeste*

The websites contain each of them a list mixed with districts and neighborhoods names, it is not the complete city neighborhoods, but all the districts are represented. Unfortunately there is a missing relation between districts and neighborhoods, what could not be found, and for this reason missing data is expected to occour, and they will be treated accordingly.

To create the **Rental Prices Dataframe** a request will be sent to each of the mentioned websites, their content parsed, cleaned and organized, to be combined and stored into a unique data frame.

The lists can be viewed at the following links;

Blog ZN Imóvel list from 2020.02.26

Blog ZS Imóvel list from 2020.03.06

Blog ZL Imóvel list from 2020.02.28

Blog ZO Imóvel list from 2020.02.28

Region *Norte* - Send request to the URL using *requests.get* method.

In [23]:
```
# define a variable for the url
url = 'https://www.znimovel.com.br/blog/qual-o-valor-do-metro-quadrado-do-aluguel-dos-

# send a request to the URL and store the response
raw = requests.get(url)

# check if data was loaded [status 200 means success]
if raw:
    print('Data loaded, status', raw.status_code)
else:
    print('Error loading data', raw.status_code)
```

```
Data loaded, status 200
```

Parse the URL content using *BeautifulSoup* with *HTML* parser.

In [24]:
```
# parse the raw data
par = bs(raw.text, 'html.parser')
print('URL content parsed.')
```

```
URL content parsed.
```

Check the returned content, initially, how many tables it contains, the HTML tag *table* will be the reference for counting.

In [25]:
```
# print number of tables
print('{} table(s) found in the parsed content.'.format(len(par.find_all('table'))))
```

```
3 table(s) found in the parsed content.
```

Check the titles of each table to know which of them is the relevant one to be used. The HTML tag *table* will be used to select the tables, and the tag *tr* to select the first row form the table.

In [26]:
```
# run a loop through the tables [tag table]
for i, title in enumerate(par.find_all('table')):
    # print the first row [tag tr]
    print('Title of Table', i)
    print(title.find_all('tr')[0].get_text())
```

```
Title of Table 0

Valor médio do metro quadrado do Aluguel
                        Apartamentos 1, 2 e 3 dormitórios
1 Vaga de Garagem
                        Zona Norte - São Paulo

Title of Table 1

Valor médio do metro quadrado do Aluguel
                        Apartamentos 2 e 3 dormitórios
2 Vagas de Garagem
                        Zona Norte - São Paulo

Title of Table 2

Valor médio do metro quadrado do Aluguel
                        Apartamentos Alto Padrão com 3 SUÌTES ou 4 dormitórios
3 ou mais  Vagas de Garagem
                        Zona Norte - São Paulo
```

Translating results.

Table 0 contains prices for apartments with 1, 2 or 3 bedrooms and 1 garage spot

Table 1 contains prices for apartments with 2 or 3 bedrooms and 2 garage spots

Table 2 contains prices for high standard apartments with 3 bedrooms with private suites or 4 bedrooms and 3 or more garage spots

For this project the **Table 0** will be used as it is the best fit for **Relocator Profile**.

Extract the data out of the table to a list and check where the required data is located. This will be done running a nested loop through the rows, on the first level, and columns, on the second level. HTML tag *tr* will be used to extract rows and *td* to extract the columns.

In [27]:
```
# get the table from the URL content [tag 'table']
par_table = par.find_all('table')[0]

# create an empty list for the entire table
tabletmp = []

# run a loop by row [tag 'tr']
for i, row in enumerate(par_table.find_all('tr')):
    # create an empty list for the current row
    celltmp = []

    # run a loop by column for the current row [tag 'td']
    for j, column in enumerate(row.find_all('td')):
        # append the text of current cell to the list
        celltmp.append(column.get_text())

    # append current line to the list
    tabletmp.append(celltmp)
```

```
# print results
print('{} rows loaded.'.format(len(tabletmp)))
```

```
28 rows loaded.
```

Check the head and tail of the list to define the range of required data.

In [28]:
```
# print the first 10 rows
print('Head 10 rows')
tabletmp[:10]
```

Out[28]:
```
Head 10 rows
[['Valor médio do metro quadrado do Aluguel\r\n\t\t\tApartamentos 1, 2 e 3 dormitório
s\xa0\n1 Vaga de Garagem\r\n\t\t\tZona Norte - São Paulo'],
 ['Bairros', 'Valor médio do m² Aluguel'],
 ['Santana', 'R$ 23,80'],
 ['Lauzane Paulista', 'R$ 22,10'],
 ['Mandaqui', 'R$ 21,70'],
 ['Tucuruvi', 'R$ 23,00'],
 ['Parada Inglesa', 'R$ 27,00'],
 ['Vila Guilherme', 'R$ 25,80'],
 ['Jardim São Paulo', 'R$ 22,40'],
 ['Vila Mazzei', 'R$ 23,60']]
```

The first row contains the table title, the second row the columns headers.

Check the tail of the table.

In [29]:
```
#print the last 10 rows
print('Tail 10 rows')
tabletmp[-10:]
```

Out[29]:
```
Tail 10 rows
[['Cachoeirinha', 'R$ 21,00'],
 ['Vila Amália', 'R$ 22,30'],
 ['Vila Gustavo', 'R$ 23,70'],
 ['Limão', 'R$ 23,30'],
 ['Vila Medeiros', 'R$ 22,20'],
 ['Vila Nova Cachoeirinha', 'R$ 21,80'],
 ['Trememb\xe9', 'R$ 20,10'],
 ['Horto Florestal', 'R$ 20,40'],
 ['Alto de Santana', 'R$ 23,80'],
 ['Dados Fevereiro 2020\r\n\t\t\tPortal ZN Imóvel']]
```

The last row contais the footnote.

There are 1 title and 1 header row, the first 2 rows at the top will be ignored. At the bottom 1 footnote row will be ignored. The data ranges from row 3 (index 2) to row 27 (index 26), counting 25 neighborhoods.

Extract only the data rows out of the table, store them into a temporary prices list and count the number of rows left.

In [30]:
```
# extract only data rows and store in temporary prices list
listtmp = tabletmp[2:-1]

# print results
print('{} rows in total.'.format(len(listtmp)))
```

```
25 rows in total.
```

Check the columns headers in row 2.

In [31]:
```
# check columns headers
tabletmp[1]
```

Out[31]:
```
['Bairros', 'Valor médio do m² Aluguel']
```

There are 2 columns, an english header will be defined for the **Rental Prices Dataframe** as following:

- **neighborhood** (Bairros), the first column is the Neighborhood name
- **mean_price_sqm** (Valor médio do m^2 Aluguel), the second column is the mean rental price per m^2 in BRL (BRL/m^2)

In [32]:
```
# define the columns names
column_names = ['neighborhood','mean_price_sqm']
```

For regions *Sul*, *Leste* and *Oeste* the task should be the same, but they will be performed less didatically as it is only repetition.

Region *Sul* - Send request to the URL and check its return.

In [33]:
```
# define a variable for the url
url = 'https://www.zsimovel.com.br/blog/qual-o-valor-do-metro-quadrado-do-aluguel-dos-

# send a request to the URL and store the response
raw = requests.get(url)

# check if data was loaded [status 200 means success]
if raw:
    print('Data loaded, status', raw.status_code)
else:
    print('Error loading data', raw.status_code)
```

```
Data loaded, status 200
```

Parse the URL content, check how many tables it contains and check the titles of each table to know which of them is the relevant one to be used.

In [34]:
```
# parse the raw data
par = bs(raw.text, 'html.parser')

# print number of tables
print('{} table(s) found in the parsed URL.\n'.format(len(par.find_all('table'))))

# run a loop through the tables [tag table]
for i, title in enumerate(par.find_all('table')):
    # print the first row [tag tr]
    print('Title of Table', i)
    print(title.find_all('tr')[0].get_text())
```

```
4 table(s) found in the parsed URL.

Title of Table 0

Valor médio do metro quadrado do Aluguel
                        Apartamentos 1 ou 2 dormitórios
1 Vaga de Garagem
                        Zona Sul - São Paulo

Title of Table 1

Valor médio do metro quadrado do Aluguel
                        Apartamentos 2 ou 3 dormitórios
2 Vagas de Garagem
                        Zona Sul - São Paulo

Title of Table 2

Valor médio do metro quadrado do Aluguel
                        Apartamentos 3 Suítes ou 4 dormitórios
3 Vagas de Garagem
                        Zona Sul - São Paulo

Title of Table 3

Valor médio do metro quadrado do Aluguel
                        Apartamentos 1 dormitório
SEM VAGA de Garagem
                        Zona Sul - São Paulo
```

Translating results.

Table 0 contains prices for apartments with 1 or 2 bedrooms and 1 garage spot

Table 1 contains prices for apartments with 2 or 3 bedrooms and 2 garage spots

Table 2 contains prices for apartments with 3 bedrooms with private suites or 4 bedrooms and 3 garage spots

Table 3 contains prices for apartments with 1 bedroom and no garage spot

For this project the **Table 0** will be used as it is the best fit for **Relocator Profile**.

Extract the data out of the table to a list and confirm where the required data is located.

In [35]:
```python
# get the table out of the parsed content [tag 'table']
par_table = par.find_all('table')[0]

# create an empty list for the entire table
tabletmp = []

# run a loop by row [tag 'tr']
for i, row in enumerate(par_table.find_all('tr')):
    # create an empty list for the current row
    celltmp = []
```

```
    # run a loop by column for the current row [tag 'td']
    for j, column in enumerate(row.find_all('td')):
        # append the text of current cell to the list
        celltmp.append(column.get_text())

    # append current line to the list
    tabletmp.append(celltmp)

# inform the number of rows loaded
print('{} rows loaded.'.format(len(tabletmp)))

# print content of three first row
print('Content of three first rows:\n', tabletmp[:3], '\n')

# print content of last two row
print('Content of 2 last rows:\n', tabletmp[-2:])
```

```
20 rows loaded.
Content of three first rows:
 [['Valor médio do metro quadrado do Aluguel\r\n\t\t\tApartamentos 1 ou 2 dormitórios
\n1 Vaga de Garagem\r\n\t\t\tZona Sul - São Paulo'], ['Bairros', 'Valor médio m² Alug
uel'], ['Vila Mariana', 'R$ 29,90']]

Content of 2 last rows:
 [['Campo Limpo', 'R$ 24,50'], ['Dados Março 2020\r\n\t\t\tPortal ZS Imóvel']]
```

The table structure is the same, 1 title, 1 header and 1 footnote row, the first 2 and the last 1 rows will be ignored. Add the data to the temporary prices list.

In [36]:
```
# store quantity of rows before adding new rows
vlen = len(listtmp)

# extend the temporary prices list with new data
listtmp.extend(tabletmp[2:-1])

# print results
print('{} rows added / {} rows in total.'.format(len(listtmp)-vlen, len(listtmp)))
```

```
17 rows added / 42 rows in total.
```

Region *Leste* - Send request to the URL and check its return.

In [37]:
```
# define a variable for the url
url = 'https://www.zlimovel.com.br/blog/qual-o-valor-do-metro-quadrado-do-aluguel-dos-

# send a request to the URL and store the response
raw = requests.get(url)

# check if data was loaded [status 200 means success]
if raw:
    print('Data loaded, status', raw.status_code)
else:
    print('Error loading data', raw.status_code)
```

```
Data loaded, status 200
```

Parse the URL content, check how many tables it contains and check the titles of each table to know which of them is the relevant one to be used.

In [38]:
```
# parse the raw data
par = bs(raw.text, 'html.parser')

# print number of tables
print('{} table(s) found in the parsed URL.\n'.format(len(par.find_all('table'))))

# run a loop through the tables [tag table]
for i, title in enumerate(par.find_all('table')):
    # print the first row [tag tr]
    print('Title of Table', i)
    print(title.find_all('tr')[0].get_text())
```

```
4 table(s) found in the parsed URL.

Title of Table 0

Valor médio do metro quadrado do Aluguel
                        Apartamentos 1, 2 e 3 dormitórios
1 Vaga de Garagem
                        Zona Leste - São Paulo

Title of Table 1

Valor médio do metro quadrado do Aluguel
Apartamentos 2 e 3 dormitórios
2 Vagas de Garagem
                        Zona Leste - São Paulo

Title of Table 2

Valor médio do metro quadrado do Aluguel
Apartamentos 3 Suítes ou 4 Dormitórios
3 ou mais Vagas de Garagem
                        Zona Leste - São Paulo

Title of Table 3

Valor médio do metro quadrado do Aluguel
Apartamentos 1, 2 e 3 dormitórios
1 Vaga de Garagem
                        Cohab, Zona Leste - São Paulo
```

Translating results.

Table 0 contains prices for apartments with 1, 2 or 3 bedrooms and 1 garage spot

Table 1 contains prices for apartments with 2 or 3 bedrooms and 2 garage spots

Table 2 contains prices for apartments with 3 bedrooms with private suites or 4 bedrooms and 3 or more garage spots

Table 3 contains prices for apartments with 1, 2 or 3 bedrooms and 1 garage spot from *Cohab*, which is government habitational support program

For this project the **Table 0** will be used as it is the best fit for **Relocator Profile**.

Extract the data out of the table to a list and confirm where the required data is located.

In [39]:
```
# get the table out of the parsed content [tag 'table']
par_table = par.find_all('table')[0]

# create an empty list for the entire table
tabletmp = []

# run a loop by row [tag 'tr']
for i, row in enumerate(par_table.find_all('tr')):
    # create an empty list for the current row
    celltmp = []

    # run a loop by column for the current row [tag 'td']
    for j, column in enumerate(row.find_all('td')):
        # append the text of current cell to the list
        celltmp.append(column.get_text())

    # append current line to the list
    tabletmp.append(celltmp)

# inform the number of rows loaded
print('{} rows loaded.'.format(len(tabletmp)))

# print content of three first row
print('Content of three first rows:\n', tabletmp[:3], '\n')

# print content of last two row
print('Content of 2 last rows:\n', tabletmp[-2:])
```

```
28 rows loaded.
Content of three first rows:
 [['Valor médio do metro quadrado do Aluguel\r\n\t\t\tApartamentos 1, 2 e 3 dormitóri
os\n1 Vaga de Garagem\r\n\t\t\tZona Leste - São Paulo'], ['Bairros', '\xa0Valor médio
m² Aluguel'], ['Tatuapé', 'R$ 25,70']]

Content of 2 last rows:
 [['Jd. Nove de Julho', 'R$ 21,90'], ['Dados Fevereiro 2020\r\n\t\t\tPortal ZL Imóve
l']]
```

The table structure is the same, 1 title, 1 header and 1 footnote row, the first 2 and the last 1 rows will be ignored. Add the data to the temporary prices list.

In [40]:
```
# store quantity rows before adding new rows
vlen = len(listtmp)

# extend the temporary prices list with new data
listtmp.extend(tabletmp[2:-1])

# print results
print('{} rows added / {} rows in total rows.'.format(len(listtmp)-vlen, len(listtmp)))
```

```
25 rows added / 67 rows in total rows.
```

Region *Oeste* - Send request to the URL and check its return.

In [41]:
```
# define a variable for the url
url = 'https://www.zoimovel.com.br/blog/qual-o-valor-do-metro-quadrado-do-aluguel-dos-
```

```
# send a request to the URL and store the response
raw = requests.get(url)

# check if data was loaded [status 200 means success]
if raw:
    print('Data loaded, status', raw.status_code)
else:
    print('Error loading data', raw.status_code)
```

```
Data loaded, status 200
```

Parse the URL content, check how many tables it contains and check the titles of each table to know which of them is the relevant one to be used.

In [42]:
```
# parse the raw data
par = bs(raw.text, 'html.parser')

# print number of tables
print('{} table(s) found in the parsed URL.\n'.format(len(par.find_all('table'))))

# run a loop through the tables [tag table]
for i, title in enumerate(par.find_all('table')):
    # print the first row [tag tr]
    print('Title of Table', i)
    print(title.find_all('tr')[0].get_text())
```

```
4 table(s) found in the parsed URL.

Title of Table 0

Valor médio do metro quadrado do Aluguel
                        Apartamentos 1 e 2 dormitórios
1 Vaga de Garagem
                        Zona Oeste - São Paulo

Title of Table 1

Valor médio do metro quadrado do Aluguel
Apartamentos 2 e 3 dormitórios
2 Vagas de Garagem
                        Zona Oeste - São Paulo

Title of Table 2

Valor médio do metro quadrado do Aluguel
Apartamentos 3 e 4 dormitórios
3 Vagas de Garagem
                        Zona Oeste - São Paulo

Title of Table 3

Valor médio do metro quadrado do Aluguel
Apartamentos ou Kitnets 1 dormitório
SEM VAGA de Garagem
                        Zona Oeste - São Paulo
```

Translating results.

Table 0 contains prices for apartments with 1 or 2 bedrooms and 1 garage spot

Table 1 contains prices for apartments with 2 or 3 bedrooms and 2 garage spots

Table 2 contains prices for apartments with 3 or 4 bedrooms and 3 garage spots

Table 3 contains prices for apartments with 1 bedroom and no garage spot

For this project the **Table 0** will be used as it is the best fit for **Relocator Profile**.

Extract the data out of the table to a list and confirm where the required data is located.

In [43]:
```
# get the table out of the parsed content [tag 'table']
par_table = par.find_all('table')[0]

# create an empty list for the entire table
tabletmp = []

# run a loop by row [tag 'tr']
for i, row in enumerate(par_table.find_all('tr')):
    # create an empty list for the current row
    celltmp = []

    # run a loop by column for the current row [tag 'td']
    for j, column in enumerate(row.find_all('td')):
        # append the text of current cell to the list
        celltmp.append(column.get_text())

    # append current line to the list
    tabletmp.append(celltmp)

# inform the number of rows loaded
print('{} rows loaded.'.format(len(tabletmp)))

# print content of three first row
print('Content of three first rows:\n', tabletmp[:3], '\n')

# print content of last two row
print('Content of 2 last rows:\n', tabletmp[-2:])
```

```
24 rows loaded.
Content of three first rows:
 [['Valor médio do metro quadrado do Aluguel\r\n\t\t\tApartamentos 1 e 2 dormitórios
\n1 Vaga de Garagem\r\n\t\t\tZona Oeste - São Paulo'], ['Bairros', 'Valor médio de m
²'], ['Perdizes', 'R$ 27,80']]

Content of 2 last rows:
 [['Vila Leopoldina', 'R$ 33,50'], ['Dados Fevereiro 2020\r\n\t\t\tPortal ZO Imóve
l']]
```

The table structure is the same, 1 title, 1 header and 1 footnote row, the first 2 and the last 1 rows will be ignored. Add the data to the temporary prices list.

In [44]:
```
# store quantity of rows before adding new rows
vlen = len(listtmp)

# extend the temporary prices list with new data
```

```
listtmp.extend(tabletmp[2:-1])

# print results
print('{} rows added / {} rows in total.'.format(len(listtmp)-vlen, len(listtmp)))
```

```
21 rows added / 88 rows in total.
```

The list is ready to be stored in a dataframe. Create the **Rental Prices Dataframe**, using *pandas.DataFrame* method.

In [45]:
```
# create the rental prices dataframe
df_rentalprices = pd.DataFrame(data=listtmp, columns=column_names)

# print results
df_rentalprices
```

Out[45]:

	neighborhood	mean_price_sqm
0	Santana	R$ 23,80
1	Lauzane Paulista	R$ 22,10
2	Mandaqui	R$ 21,70
3	Tucuruvi	R$ 23,00
4	Parada Inglesa	R$ 27,00
...	...	...
83	Pirituba	R$ 23,70
84	Vila Sônia	R$ 25,90
85	Vila São Francisco (ZO)	*Prejudicado
86	Jardins	R$ 32,20
87	Vila Leopoldina	R$ 33,50

88 rows × 2 columns

For neighborhoods which did not have enough samples to measure the mean rental price, the text **"*Prejudicado"** has been added instead of the mean price. Check the observations whitout mean rental price.

In [46]:
```
# check entries with text in mean price column
df_rentalprices[df_rentalprices['mean_price_sqm']=='*Prejudicado']
```

Out[46]:

	neighborhood	mean_price_sqm
80	República	*Prejudicado
85	Vila São Francisco (ZO)	*Prejudicado

Drop the observations withou mean rental price, as they cannot be used. This will be done using *pandas.DataFrame.drop* method.

In [47]:
```
# select the rows index to drop
vidx = list(df_rentalprices[df_rentalprices['mean_price_sqm']=='*Prejudicado'].index)

# drop entries with text in mean price column by index
df_rentalprices.drop(vidx, axis=0, inplace=True)

# print results
df_rentalprices
```

Out[47]:

	neighborhood	mean_price_sqm
0	Santana	R$ 23,80
1	Lauzane Paulista	R$ 22,10
2	Mandaqui	R$ 21,70
3	Tucuruvi	R$ 23,00
4	Parada Inglesa	R$ 27,00
...	...	...
82	Liberdade	R$ 30,70
83	Pirituba	R$ 23,70
84	Vila Sônia	R$ 25,90
86	Jardins	R$ 32,20
87	Vila Leopoldina	R$ 33,50

86 rows × 2 columns

Data in **mean_price_sqm** column doesn't have a good fit, as it contains the currency symbol and the decimal separator is comma, the Brazilian standard.

Convert the price *string* to *float* format. This will be done firstly removing the currency symbol, then replacing the decimal separator, both using *pandas.Dataframe.apply* method, and finally converting *string* to *float* in the dataframe using *pandas.DataFrame.astype* method.

In [48]:
```
# remove currency symbol and change decimal separator
df_rentalprices['mean_price_sqm'] = df_rentalprices['mean_price_sqm'].apply(lambda x :
df_rentalprices['mean_price_sqm'] = df_rentalprices['mean_price_sqm'].apply(lambda x :

# convert the price column to float
df_rentalprices = df_rentalprices.astype({'mean_price_sqm': 'float64'})

# print the columns data types
print('Dataframedata types:\n', df_rentalprices.dtypes, '\n')
```

```
Dataframedata types:
 neighborhood       object
mean_price_sqm    float64
dtype: object
```

Check results

In [49]:
```
# print results
df_rentalprices
```

Out[49]:

	neighborhood	mean_price_sqm
0	Santana	23.8
1	Lauzane Paulista	22.1
2	Mandaqui	21.7
3	Tucuruvi	23.0
4	Parada Inglesa	27.0
...	...	...
82	Liberdade	30.7
83	Pirituba	23.7
84	Vila Sônia	25.9
86	Jardins	32.2
87	Vila Leopoldina	33.5

86 rows × 2 columns

Check if there is any duplicated neighborhood in **Rental Prices Dataframe**. It will be done with *pandas.DataFrame.groupby* method, grouping the dataframe by neighborhood and counting the mean prices.

In [50]:
```
# group the rental prices by neighborhoods counting the mean price
df_group = df_rentalprices.groupby('neighborhood', sort=True).count().reset_index()

# check for counting greater then 1
df_group[df_group['mean_price_sqm'] > 1]
```

Out[50]:

	neighborhood	mean_price_sqm

All the observations in **Rental Prices Dataframe** are unique.

Merge **Districts Dataframe** (**district** as key) with **Rental Prices Dataframe** (**neighborhood** as key) using *pandas.merge* method. The resulting dataframe will be **Districts Dataframe** with mean rental prices. As stated before, missing data is expected to occour.

In [51]:
```
# merge districts and rental prices dataframes
df_districts = pd.merge(df_districts, df_rentalprices, how='left', left_on='district',

# print results
df_districts
```

Out[51]:

	region	borough	district	area_sqkm	neighborhood	mean_price_sqm
0	Centro	Sé	Bela Vista	2.72	Bela Vista	30.6
1	Centro	Sé	Bom Retiro	4.21	Bom Retiro	29.1
2	Centro	Sé	Cambuci	3.92	NaN	NaN
3	Centro	Sé	Consolação	3.82	NaN	NaN
4	Centro	Sé	Liberdade	3.65	Liberdade	30.7
...	...	...	...	...	...	...
91	Sul	Santo Amaro	Campo Grande	12.95	NaN	NaN
92	Sul	Santo Amaro	Santo Amaro	16.04	Santo Amaro	26.8
93	Sul	Vila Mariana	Moema	9.08	Moema	32.6
94	Sul	Vila Mariana	Saúde	9.31	Saúde	28.2
95	Sul	Vila Mariana	Vila Mariana	8.60	Vila Mariana	29.9

96 rows × 6 columns

The column **neighborhood** contains the same information as **district**, it will be dropped from the data dataframe using *pandas.DataFrame.drop* method.

In [52]:
```
# drop neighborhood column
df_districts.drop('neighborhood', axis=1, inplace=True)

# print results
df_districts
```

Out[52]:

	region	borough	district	area_sqkm	mean_price_sqm
0	Centro	Sé	Bela Vista	2.72	30.6
1	Centro	Sé	Bom Retiro	4.21	29.1
2	Centro	Sé	Cambuci	3.92	NaN
3	Centro	Sé	Consolação	3.82	NaN
4	Centro	Sé	Liberdade	3.65	30.7
...	...	...	...	...	...
91	Sul	Santo Amaro	Campo Grande	12.95	NaN
92	Sul	Santo Amaro	Santo Amaro	16.04	26.8
93	Sul	Vila Mariana	Moema	9.08	32.6
94	Sul	Vila Mariana	Saúde	9.31	28.2
95	Sul	Vila Mariana	Vila Mariana	8.60	29.9

96 rows × 5 columns

Last step to have dataframe ready is to treat the **missing values** in column **mean_price_sqm**. The missing values will be replaced by the mean price of respective region, but first for reference, calculate the mean price for each region, using *pandas.DataFrame.grouby* method.

In [53]:
```
# group dataframe by region calculating the mean price
df_districts.groupby('region', sort=True).mean().reset_index()
```

Out[53]:

	region	area_sqkm	mean_price_sqm
0	Centro	3.333750	29.925000
1	Leste	10.045758	23.053846
2	Norte	16.658333	22.725000
3	Oeste	8.598667	29.444444
4	Sul	33.662727	28.388889

Replace the missing **mean_price_sqm** by its respective region mean price. It will be done firstly defining a function to return the region mean price taking the region name as parameter, *pandas.DataFrame.group* method will be used for this. Having the function defined, *pandas.DataFrame.apply* method will be used to call the function for the observations where mean price is *NaN*, *numpy.isnan* method will be used to chech for *NaN*.

In [54]:
```
# define a funciton to return the region mean price
def get_region_mean(r):
    m = df_districts[df_districts['region']==r].groupby('region', sort=True).mean().re
    return round(float(m),2)

# use apply to call the defined function when mean price is missing
df_districts['mean_price_sqm'] = df_districts.apply(lambda x : get_region_mean(x.regio

# inform when it is finished
print('Replace finished.')

# print results
df_districts
```

```
Replace finished.
```

Out[54]:

	region	borough	district	area_sqkm	mean_price_sqm
0	Centro	Sé	Bela Vista	2.72	30.60
1	Centro	Sé	Bom Retiro	4.21	29.10
2	Centro	Sé	Cambuci	3.92	29.93
3	Centro	Sé	Consolação	3.82	29.93
4	Centro	Sé	Liberdade	3.65	30.70
...	...	...	...	...	...
91	Sul	Santo Amaro	Campo Grande	12.95	28.39
92	Sul	Santo Amaro	Santo Amaro	16.04	26.80
93	Sul	Vila Mariana	Moema	9.08	32.60
94	Sul	Vila Mariana	Saúde	9.31	28.20
95	Sul	Vila Mariana	Vila Mariana	8.60	29.90

96 rows × 5 columns

Step 4 - Exploratory Data Analysis on *Districts Dataframe*

Import addional libraries.

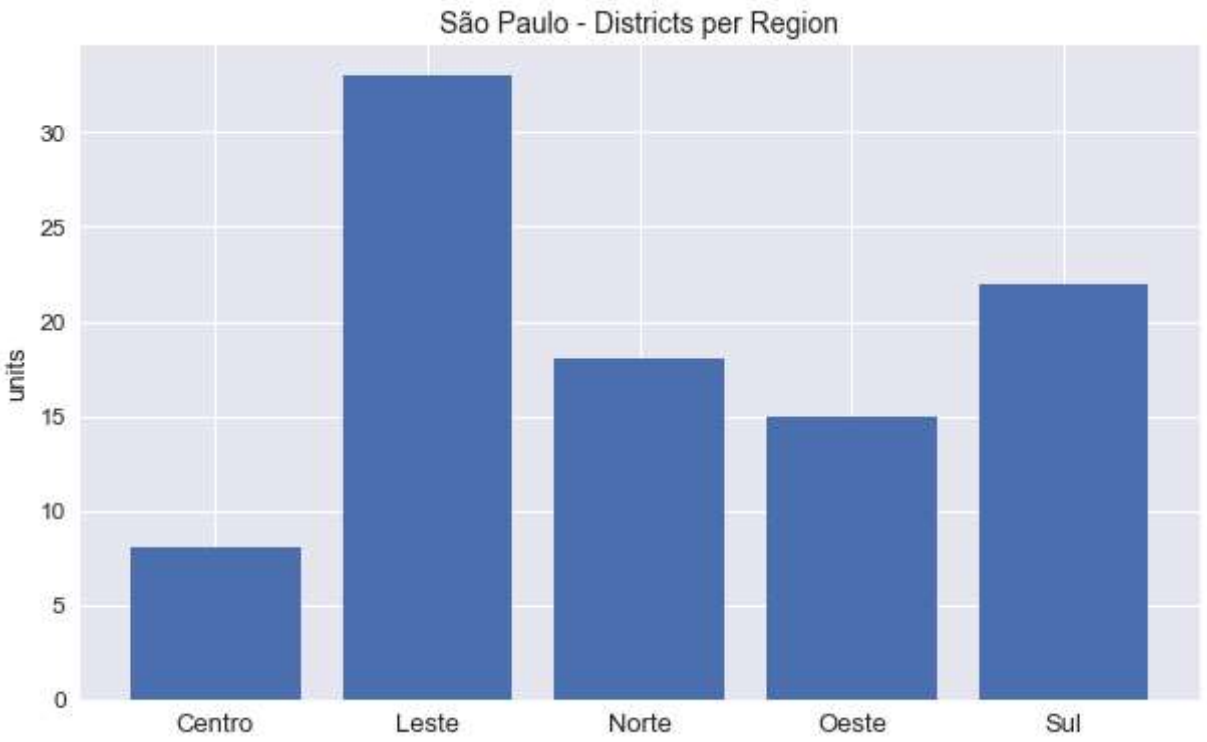
1. **Matplotlib**: create plots

In [55]:
```
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
%matplotlib inline
```

Check how many districts each region contains. First the **Districts Dataframe** will be grouped by **region** with *pandas.DataFrame.groupby* method, then a **bar plot** will be constructed using *Matplotlib*.

In [56]:
```
# group dataframe by region counting the group observations
vdf = df_districts.groupby('region').count().reset_index()

# define plot style
plt.style.use('seaborn')

# set font size for axis ticks
plt.rc('xtick', labelsize=13)
plt.rc('ytick', labelsize=12)

# create figure object
fig, ax = plt.subplots(figsize=(10,6))

# add bar plot to the figure
ax.bar(vdf['region'], vdf['district'])
```

```
# add plot title and ylable
ax.set_title('São Paulo - Districts per Region', {'fontsize':14})
ax.set_ylabel('units', {'fontsize':13})

# show the figure
plt.show()
```



Check **Districts Dataframe** statistics using *pandas.DataFrame.describe* method.

In [57]:
```
df_districts.describe(include='all')
```

Out[57]:

	region	borough	district	area_sqkm	mean_price_sqm
count	96	96	96	96.000000	96.000000
unique	5	32	96	NaN	NaN
top	Leste	Sé	Ponte Rasa	NaN	NaN
freq	33	8	1	NaN	NaN
mean	NaN	NaN	NaN	15.912396	25.785521
std	NaN	NaN	NaN	26.968078	3.597908
min	NaN	NaN	NaN	2.190000	17.700000
25%	NaN	NaN	NaN	7.395000	23.050000
50%	NaN	NaN	NaN	9.780000	25.150000
75%	NaN	NaN	NaN	13.560000	28.467500
max	NaN	NaN	NaN	208.190000	33.500000

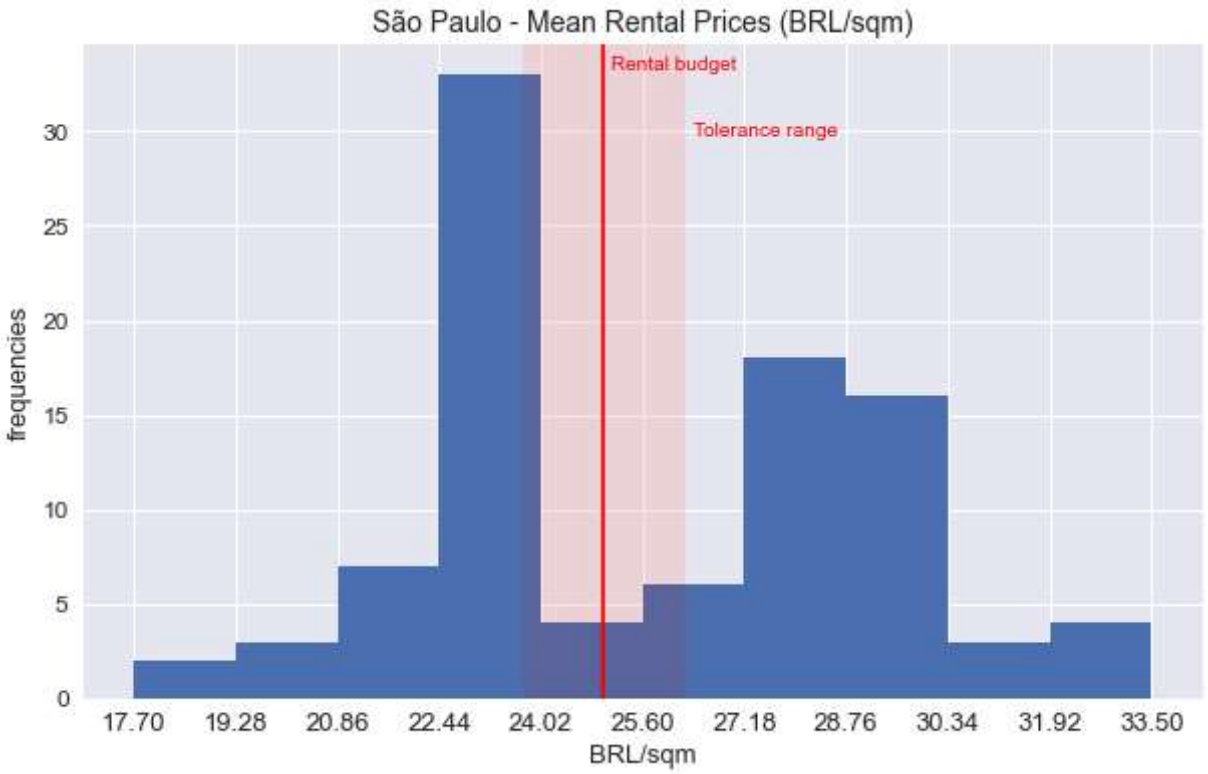
Interesting to notice the variation of rental prices across the city, what is expected for metropolis like São Paulo. The minimum mean rental price is **BRL 17.70/m^2** and the maximum

BRL 33.50/m^2, almost double. Mean at **25.78** is balanced with the median at **25.15**. The range between **Q1 = 23.05** and **Q3 = 28.47** is not wide, but the relocator budget at **BRL 25.00/m^2** should not be an issue.

There is also huge variation in districts land area, ranging from **2.19 km^2** to **208.19 km^2**, what should be considered when looking for venues with **API Fousquare**, as the radius cannot be the same for all districts, to minimize overlaping and skiping areas.

Check the mean prices frequencies with a histograms to have an idea on how is the Relocator Rental Budget positioned, *Matplotlib* will be used.

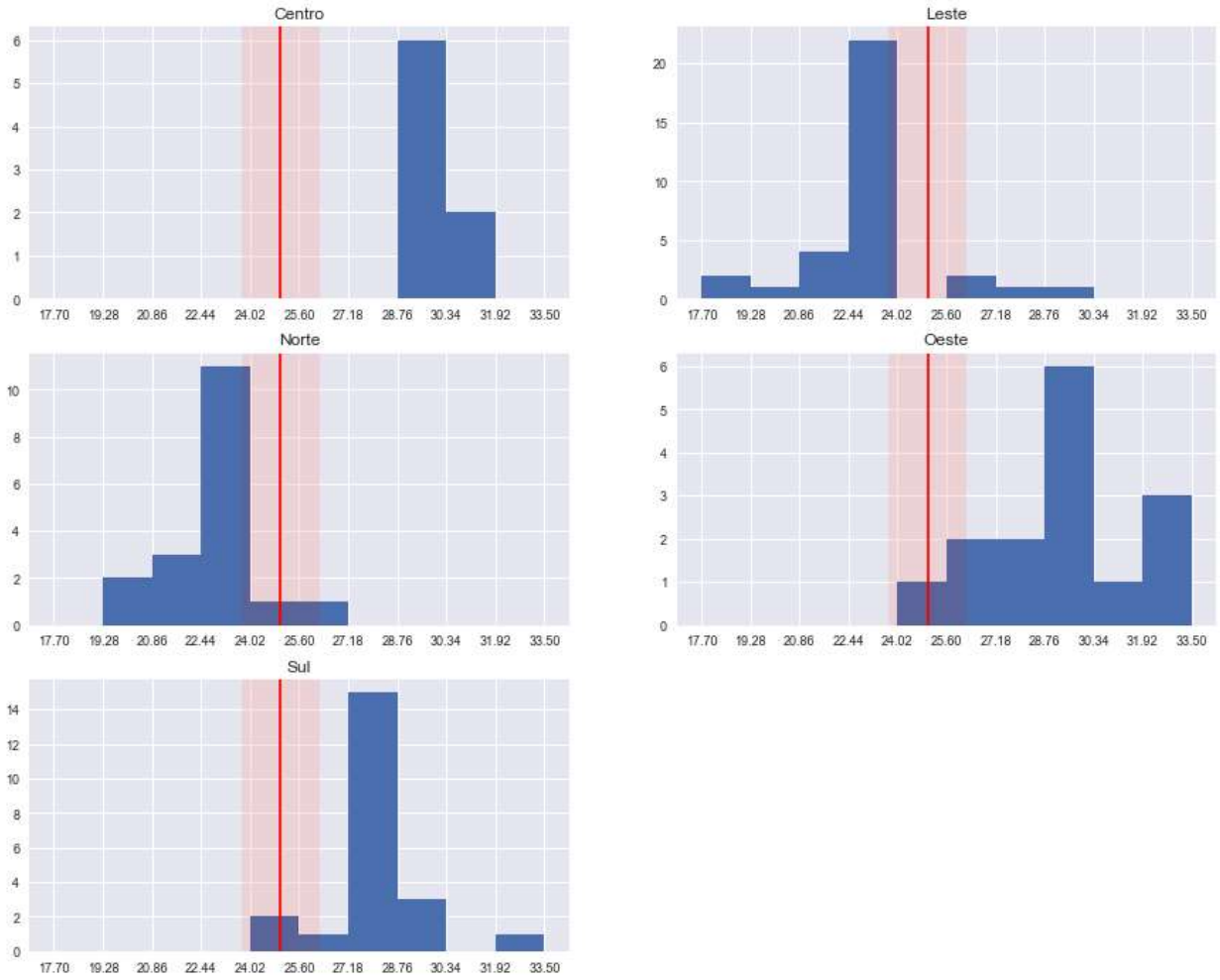
In [58]:
```
# caculate bins edges
cnt, bins = np.histogram(df_districts['mean_price_sqm'])

# create figure object
fig, ax = plt.subplots(figsize=(10,6))

# add histogram to the figure
ax.hist(df_districts['mean_price_sqm'], bins=bins)

# define xticks values to match bins edges
ax.set_xticks(bins)

# get yticks (values on y-axis)
ylocs, ylabels = plt.yticks()

# add a vertical line for rental budget and annotate a text for it
plt.axvline(x=budget_sqm, ymin=min(ylocs), ymax=max(ylocs), color='r')
ax.annotate('Rental budget', xy=(budget_sqm*1.005, max(ylocs)*0.95), xycoords='data',

# add a rectangle for the tolerance range and annotate a text for it
rect = plt.Rectangle((budget_sqm_min, min(ylocs)), (budget_sqm_max-budget_sqm_min), (m
ax.add_patch(rect)
ax.annotate('Tolerance range', xy=(budget_sqm_max*1.005, max(ylocs)*0.85), xycoords='d

# add plot title and ylable
ax.set_title('São Paulo - Mean Rental Prices (BRL/sqm)', {'fontsize':14})
ax.set_xlabel('BRL/sqm', {'fontsize':13})
ax.set_ylabel('frequencies', {'fontsize':13})

# show the figure
plt.show()
```



Check the mean prices frequencies, but split between district, with the same bin setup for better comparison.

In [88]:
```
# create a list of unique regions
reglist = list(df_districts['region'].unique())

# set the font size for axis ticks (values)
plt.rc('xtick', labelsize=9)
plt.rc('ytick', labelsize=9)

# create figure object
fig = plt.figure(figsize=(15,12))

# run a loop through the regions list
for i, r in enumerate(reglist):
    # add plot for current region
    ax = fig.add_subplot(3, 2, i+1)
    ax.hist(df_districts[df_districts['region']==r]['mean_price_sqm'], bins=bins)

    # define the xticks to match the bins edges of current region
    ax.set_xticks(bins)

    # get yticks (values on y-axis)
    ylocs, ylabels = plt.yticks()

    # add a vertical line for rental budget
    plt.axvline(x=budget_sqm, ymin=min(ylocs), ymax=max(ylocs), color='r')

    # add a rectangle for the tolerance range
    rect = plt.Rectangle((budget_sqm_min, min(ylocs)), (budget_sqm_max-budget_sqm_min)
    ax.add_patch(rect)

    # add plot title for the current region
    ax.set_title(r, {'fontsize':12})
```

```
# show the figure
plt.show()
```




```
<Figure size 576x396 with 0 Axes>
```

From the regions histograms, it is visible that each region has a different rental prices range, with some overlaping in middle area, exception is *Centro* region (downtown), which has higher rental prices starting from **BRL28.76/m^2**. The high frequency between **22.44 and 24.02**, found in previous histogram, is concentrated in regions *Leste* and *Norte*. Looking only at the rental prices, except from *Centro*, all the other regions are candidates for relocation.

Step 5 - Create a map to show the Districts of *Target Location*

Import addional libraries.

1. **Folium**: create maps
2. **Geopy**: get geographical data

In [60]:
```
import folium
from geopy.geocoders import Nominatim
```

To point the districts on the map, their coordinates are required. Get districts coodinates will be done with *geopy.geocoders.Nominatim*, for better accuracy, the search will be done with **district** and **borough** followed by **"São Paulo, BR**", if it is not found, **borough** will be removed for a second search.

In [61]:
```
# define geolocator user agent
geolc = Nominatim(user_agent="sp_search")

# create a temporary list
coord = []

# run a loop through the dataframe
for b, d in zip(df_districts['borough'], df_districts['district']):
    # set a variable to search location
    vloc = d + ', ' + b + ', São Paulo, BR'

    # send the geocode search
    loct = geolc.geocode(vloc)

    # if it fails, try without borough
    if loct == None:
        vloc = d + ', São Paulo, BR'
        loct = geolc.geocode(vloc)

    # append results to the temporary list
    if loct != None:
        # append results into the list
        coord.append([d, round(loct.latitude,4), round(loct.longitude,4)])

# check results and print status
if len(coord) > 0:
    print('{} coordinates acquired.'.format(len(coord)))
else:
    print('Fail to acquire coordinates.')
```

```
96 coordinates acquired.
```

Store coordinates into **Coordinates Dataframe** using *pandas.DataFrame* method.

In [62]:
```
# store coordinates into dataframe
df_coord = pd.DataFrame(coord, columns=['district', 'latitude', 'longitude'])

# print results
df_coord
```

Out[62]:

	district	latitude	longitude
0	Bela Vista	-23.2970	-46.0267
1	Bom Retiro	-23.5271	-46.6368
2	Cambuci	-23.5661	-46.6137
3	Consolação	-23.5578	-46.6605
4	Liberdade	-23.5552	-46.6356
...	...	...	...
91	Campo Grande	-23.6578	-46.6986
92	Santo Amaro	-23.6562	-46.7191
93	Moema	-23.5827	-46.6490
94	Saúde	-23.5973	-46.6359
95	Vila Mariana	-23.5892	-46.6347

96 rows × 3 columns

Merge **Districts Dataframe** with **Coordinates Dataframe** using *pandas.merge* method. The resulting dataframe will be **Districts Dataframe** with coordinates.

In [63]:
```
# merge Districts and Coordinates dataframes
df_districts = pd.merge(df_districts, df_coord, how='inner', on='district')

# print results
df_districts
```

Out[63]:

	region	borough	district	area_sqkm	mean_price_sqm	latitude	longitude
0	Centro	Sé	Bela Vista	2.72	30.60	-23.2970	-46.0267
1	Centro	Sé	Bom Retiro	4.21	29.10	-23.5271	-46.6368
2	Centro	Sé	Cambuci	3.92	29.93	-23.5661	-46.6137
3	Centro	Sé	Consolação	3.82	29.93	-23.5578	-46.6605
4	Centro	Sé	Liberdade	3.65	30.70	-23.5552	-46.6356
...	...	...	...	...	...	...	...
91	Sul	Santo Amaro	Campo Grande	12.95	28.39	-23.6578	-46.6986
92	Sul	Santo Amaro	Santo Amaro	16.04	26.80	-23.6562	-46.7191
93	Sul	Vila Mariana	Moema	9.08	32.60	-23.5827	-46.6490
94	Sul	Vila Mariana	Saúde	9.31	28.20	-23.5973	-46.6359
95	Sul	Vila Mariana	Vila Mariana	8.60	29.90	-23.5892	-46.6347

96 rows × 7 columns

Define a **regions colors** for better visualization on the map. Color definition will be done with *Matplotlib* and then they will be attached to **Districts Dataframe** using *pandas.merge* method.

In [64]:
```
# create a list of unique boroughs
reglist = df_districts['region'].unique()

# create a color list by borough [RGBA format]
trgb = plt.cm.tab10(np.linspace(0, 1, len(reglist)))

# convert the color from RGBA to HEX
thex = []
for i in range(len(trgb)):
    thex.append(mcolors.rgb2hex(trgb[i][:3]))

# create a temporary list to assign colors to regions
tmplist = list()
for b, c in zip(reglist, thex):
    tmplist.append([b,c])

# create a dataframe from the temporary list
colortable = pd.DataFrame(columns=['region','region_color'], data=tmplist)

# merge Districts and color table data frames
df_districts = pd.merge(df_districts, colortable, how='inner', on='region')

# print results
df_districts
```

Out[64]:

	region	borough	district	area_sqkm	mean_price_sqm	latitude	longitude	region_color
0	Centro	Sé	Bela Vista	2.72	30.60	-23.2970	-46.0267	#1f77b4
1	Centro	Sé	Bom Retiro	4.21	29.10	-23.5271	-46.6368	#1f77b4
2	Centro	Sé	Cambuci	3.92	29.93	-23.5661	-46.6137	#1f77b4
3	Centro	Sé	Consolação	3.82	29.93	-23.5578	-46.6605	#1f77b4
4	Centro	Sé	Liberdade	3.65	30.70	-23.5552	-46.6356	#1f77b4
...	...	...	...	...	...	...	...	...
91	Sul	Santo Amaro	Campo Grande	12.95	28.39	-23.6578	-46.6986	#17becf
92	Sul	Santo Amaro	Santo Amaro	16.04	26.80	-23.6562	-46.7191	#17becf
93	Sul	Vila Mariana	Moema	9.08	32.60	-23.5827	-46.6490	#17becf
94	Sul	Vila Mariana	Saúde	9.31	28.20	-23.5973	-46.6359	#17becf
95	Sul	Vila Mariana	Vila Mariana	8.60	29.90	-23.5892	-46.6347	#17becf

96 rows × 8 columns

Create a map of São Paulo pointing its Districts, it will be done *Folium*.

Important notice:

Unfortunately the Folium maps are not displayed in GitHub, they are interactive objects not supported, as an option to see the maps the Jupyter nbviewer can be used. Once it is opened just need to paste the URL of the GitHub document.

In [65]:
```
# get São Paulo coordenates to start the map
loct = geolc.geocode('São Paulo, BR')
latsp = round(loct.latitude, 4)
lngsp = round(loct.longitude, 4)

# create the map
sp_map = folium.Map(location=[latsp, lngsp], width='70%', height='70%',
                    min_zoom=9, max_zoom=12, zoom_start=10, control_scale=True)

# run a loop through the dataframe creating the points (circle marks)
for lat, lng, reg, dst, clr in zip(df_districts['latitude'], df_districts['longitude']
                                   df_districts['district'], df_districts['region_colo
    # create marker label
    label = '{} ({})'.format(dst, reg)
    label = folium.Popup(label, parse_html=True)

    # create the circle mark
    folium.CircleMarker([lat, lng],
                        popup = label,
                        radius = 5,
                        color = clr,
                        fill = True,
                        fill_color = clr,
                        fill_opacity = 0.3,
                        parse_html = False,
                        no_touch=True).add_to(sp_map)

# show the map
sp_map
```

Out[65]: Make this Notebook Trusted to load map: File -> Trust Notebook

---

Step 6 - Get Districts Venues Information

Import addional libraries.

1. **Credentials**: API credentials, local user defined package
2. **Json**: manipulate files in *json* format
3. **Math**: mathematical formulas

In [66]:
```
import credentials
import json
import math
```

Search for districts venues. The **API Foursquare** will be used to search venues based on the districts coordinates, for each district a request will be sent to the API using *requests.get* method and responses will be interpreted with *json* method.

Get **API Foursquare credentials**. The connection to **API Foursquare** requires private user credentials, they have previouly been requested and stored into the local user defined package *Credentials*.

In [67]:
```
# get the credentials, token will not be used
client_id, client_secret, access_token = credentials.get_foursquare()
```

Will also be defined the **API version** (required), **limit of venues** to be returned and the **search radius** around the location.

API version and limit of venues will be fixed, radius will be set according to each district area (km^2), limited to 2000 meters, as following:

$$r = \frac{\sqrt{area}}{2} \times 1000 = \sqrt{area} \times 500$$

The search will be limited to the Relocator **Priority List**, the *dictionary indexes* will be converted into a single string, having each of the *indexes* separated by comma and without blank spaces to be used as input for the API request.

In [68]:
```
# define API version and limit
api_version = '20180605'
limit = 50

# convert priority list to a single string
priocateg = str(list(prio_list.keys())).strip('[]').replace('\'','').replace(' ','')

# create an empty list to store venues information
venueslist = []

# run a loop through the dataframe
for d, lat, lng, a in zip(df_districts['district'], df_districts['latitude'], df_distr
    # define radius for current district
    radius = int(math.sqrt(a)*500) if round(math.sqrt(a)*500,0)<=2000 else 2000

    # create the API Foursquare request URL
    url='https://api.foursquare.com/v2/venues/search?&client_id={}&client_secret={}&v=
    format(client_id, client_secret, api_version, lat, lng, radius, limit, priocateg)

    # make the GET request and store in JSON format
    vresponse = requests.get(url).json()['response']['venues']

    # add to temporary list only relevant information for each venue
    for v in vresponse:
        venueslist.append([d, v['name'], v['categories'][0]['name']])

# print the quantity of venues returned
print('{} venues returned.'.format(len(venueslist)))
```

```
3650 venues returned.
```

Create **Venues Dataframe** to store the response from **API Foursquare**, using *pandas.DataFrame*.

In [69]:
```
# create Venues Dataframe
df_venues = pd.DataFrame(data=venueslist, columns=['district',
                                                   'venue_name',
                                                   'venue_category'])


# print results
df_venues
```

Out[69]:

	district	venue_name	venue_category
0	Bom Retiro	Extra Supermercado	Supermarket
1	Bom Retiro	Estação Tiradentes (Metrô)	Metro Station
2	Bom Retiro	Estação Luz (Metrô)	Metro Station
3	Bom Retiro	Estação Santa Cecília (Metrô)	Metro Station
4	Bom Retiro	Estação Armênia (Metrô)	Metro Station
...	...	...	...
3645	Vila Mariana	Droga Raia	Pharmacy
3646	Vila Mariana	Praça Kant	Park
3647	Vila Mariana	Master Supermercados	Supermarket
3648	Vila Mariana	Colégio Cristo Rei	School
3649	Vila Mariana	Med Tec Cirúrgica	Miscellaneous Shop

3650 rows × 3 columns

Check how many unique **categories** have been returned, using *pandas.DataFrame.unique* method.

In [70]:
```
# print quantity of unique categories
print('{} unique categories returned.'.format(len(df_venues['venue_category'].unique()
```

```
43 unique categories returned.
```

When specifying categories to **API Foursquare**, all the sub-categories are returned in response. There are more categories than the five in **Priority List**. Remove venues of not required categories, using *pandas.DataFrame.isin* method.

In [71]:
```
# count observations before removing
vlen = df_venues.shape[0]

# remove venues of not required categories
df_venues = df_venues[df_venues['venue_category'].isin(list(prio_list.values()))]

# print quantity of removed observations
print('{} venues removed.'.format(vlen-df_venues.shape[0]))
```

```
175 venues removed.
```

Create **Venues Count Dataframe** with the venues quantity for each district, using _pandas.pivot*table* method.

In [72]:
```
# create Venues Count dataframe
df_venues_count = pd.pivot_table(df_venues, values='venue_name', index='district', col

# print results
df_venues_count
```

Out[72]:

venue_category district	Elementary School	Metro Station	Park	Pharmacy	Supermarket
Alto de Pinheiros	2.0	1.0	13.0	24.0	4.0
Anhanguera	3.0	NaN	3.0	NaN	1.0
Aricanduva	6.0	NaN	NaN	12.0	3.0
Artur Alvim	4.0	4.0	NaN	25.0	3.0
Barra Funda	4.0	3.0	2.0	22.0	12.0
...	...	...	...	...	...
Vila Matilde	6.0	2.0	3.0	25.0	12.0
Vila Medeiros	6.0	NaN	1.0	29.0	2.0
Vila Prudente	5.0	3.0	7.0	23.0	8.0
Vila Sônia	7.0	NaN	4.0	29.0	9.0
Água Rasa	4.0	1.0	3.0	30.0	9.0

94 rows × 5 columns

Change the columns names removing capitals and blank spaces, using *pandas.DataFrame,rename* method.

In [73]:
```
# replace upper case by lower case and remove blank spaces
df_venues_count.rename(columns={'Elementary School': 'elementary_school',
                                'Metro Station': 'metro_station',
                                'Park': 'park',
                                'Pharmacy': 'pharmacy',
                                'Supermarket': 'supermarket'}, inplace=True)

# print results
df_venues_count
```

Out[73]:

venue_category district	elementary_school	metro_station	park	pharmacy	supermarket
Alto de Pinheiros	2.0	1.0	13.0	24.0	4.0
Anhanguera	3.0	NaN	3.0	NaN	1.0
Aricanduva	6.0	NaN	NaN	12.0	3.0
Artur Alvim	4.0	4.0	NaN	25.0	3.0
Barra Funda	4.0	3.0	2.0	22.0	12.0
...	...	...	...	...	...
Vila Matilde	6.0	2.0	3.0	25.0	12.0
Vila Medeiros	6.0	NaN	1.0	29.0	2.0
Vila Prudente	5.0	3.0	7.0	23.0	8.0
Vila Sônia	7.0	NaN	4.0	29.0	9.0
Água Rasa	4.0	1.0	3.0	30.0	9.0

94 rows × 5 columns

Not all of the 96 districts have venues categories in **Priority List**. Merge the **Districts Dataframe** with **Venues Count Dataframe** to have all the 96 districts available for ranking, *pandas.merge* will be used.

In [74]:
```
# merge Districts and Venues Count dataframes
df_venues_count = pd.merge(left=df_districts['district'], right=df_venues_count, how='

# print results
df_venues_count
```

Out[74]:

	district	elementary_school	metro_station	park	pharmacy	supermarket
0	Bela Vista	NaN	NaN	NaN	NaN	NaN
1	Bom Retiro	2.0	5.0	4.0	19.0	5.0
2	Cambuci	3.0	NaN	1.0	21.0	3.0
3	Consolação	NaN	4.0	1.0	28.0	12.0
4	Liberdade	NaN	5.0	1.0	29.0	10.0
...	...	...	...	...	...	...
91	Campo Grande	3.0	4.0	5.0	25.0	8.0
92	Santo Amaro	2.0	4.0	5.0	28.0	8.0
93	Moema	NaN	4.0	3.0	28.0	11.0
94	Saúde	NaN	7.0	2.0	31.0	8.0
95	Vila Mariana	1.0	5.0	2.0	26.0	12.0

96 rows × 6 columns

Categories which have not been returned from **API Foursquare** have been set as *NaN* when creating the *Pivot Table* dataframe. Replace *NaN* entries by *0* (zero) using *pandas.DataFrame.fillna* method.

In [75]:
```
# replace NaN entries by 0
df_venues_count.fillna(value=0, inplace=True)

# print results
df_venues_count
```

Out[75]:

	district	elementary_school	metro_station	park	pharmacy	supermarket
0	Bela Vista	0.0	0.0	0.0	0.0	0.0
1	Bom Retiro	2.0	5.0	4.0	19.0	5.0
2	Cambuci	3.0	0.0	1.0	21.0	3.0
3	Consolação	0.0	4.0	1.0	28.0	12.0
4	Liberdade	0.0	5.0	1.0	29.0	10.0
...	...	...	...	...	...	...
91	Campo Grande	3.0	4.0	5.0	25.0	8.0
92	Santo Amaro	2.0	4.0	5.0	28.0	8.0
93	Moema	0.0	4.0	3.0	28.0	11.0
94	Saúde	0.0	7.0	2.0	31.0	8.0
95	Vila Mariana	1.0	5.0	2.0	26.0	12.0

96 rows × 6 columns

Step 7 - Create the *Districts Rank* according to the *Relocator Profile*

Import additional libraries.

- **Scikit-Learn**: machine learning package

In [76]:
```
from sklearn.cluster import DBSCAN
```

The rank will be based on groups of districts, created on their similarities. To group them **Machine Learning Techniques** will be used, for this project **Clustering Technique** with **DBSCAN algorithm**, which stands for *Density-Based Spatial Clustering of Applications with Noise*. The choice for DBSCAN was done because it is capable to detect clusters with different densities, sizes and shapes, it does not require to define a number of clusters to start and it can also identify noise and outliers among the data.

Create a **Clusters Dataframe** to run the clustering process. The clustering will run based on categories from **Priority List** each district contains, what is ready into **Venues Count Dataframe** and also based on **mean rental prices** per district, what is ready in **Districts Dataframe**. To create the **Clusters Dataframe**, **Venues Count** and **Districts** dataframes will be mergeded using *pandas.merge* method.

In [77]:
```
# merge Venues Count and Districts dataframes
df_clusters = pd.merge(left=df_venues_count, right=df_districts[['district','mean_pric

# print results
df_clusters
```

Out[77]:

	district	elementary_school	metro_station	park	pharmacy	supermarket	mean_price_sqm
0	Bela Vista	0.0	0.0	0.0	0.0	0.0	30.60
1	Bom Retiro	2.0	5.0	4.0	19.0	5.0	29.10
2	Cambuci	3.0	0.0	1.0	21.0	3.0	29.93
3	Consolação	0.0	4.0	1.0	28.0	12.0	29.93
4	Liberdade	0.0	5.0	1.0	29.0	10.0	30.70
...	...	...	...	...	...	...	...
91	Campo Grande	3.0	4.0	5.0	25.0	8.0	28.39
92	Santo Amaro	2.0	4.0	5.0	28.0	8.0	26.80
93	Moema	0.0	4.0	3.0	28.0	11.0	32.60
94	Saúde	0.0	7.0	2.0	31.0	8.0	28.20
95	Vila Mariana	1.0	5.0	2.0	26.0	12.0	29.90

96 rows × 7 columns

Prepare data for clustering process. First, drop **district** column, as it is not part of the features to be used for clustering, *pandas.DataFrame.drop* will be used.

In [78]:
```
# drop district column
df_clusters.drop('district', axis=1, inplace=True)

# print results
df_clusters
```

Out[78]:

	elementary_school	metro_station	park	pharmacy	supermarket	mean_price_sqm
0	0.0	0.0	0.0	0.0	0.0	30.60
1	2.0	5.0	4.0	19.0	5.0	29.10
2	3.0	0.0	1.0	21.0	3.0	29.93
3	0.0	4.0	1.0	28.0	12.0	29.93
4	0.0	5.0	1.0	29.0	10.0	30.70
...	...	...	...	...	...	...
91	3.0	4.0	5.0	25.0	8.0	28.39
92	2.0	4.0	5.0	28.0	8.0	26.80
93	0.0	4.0	3.0	28.0	11.0	32.60
94	0.0	7.0	2.0	31.0	8.0	28.20
95	1.0	5.0	2.0	26.0	12.0	29.90

96 rows × 6 columns

Second, convert features values of venues categories count (the first five columns) as follwing:

- **0** when **value = 0**
- **1** when **value > 0**

Method *pandas.DataFrame.apply* will be used to convert features values.

In [79]:
```
# run a loop through datafame for first five columns
for i, c in enumerate(list(df_clusters.columns)):
    # convert features values
    if i < 5: df_clusters[c] = df_clusters[c].apply(lambda x : x if x == 0 else 1)

# print results
df_clusters
```

Out[79]:

	elementary_school	metro_station	park	pharmacy	supermarket	mean_price_sqm
0	0.0	0.0	0.0	0.0	0.0	30.60
1	1.0	1.0	1.0	1.0	1.0	29.10
2	1.0	0.0	1.0	1.0	1.0	29.93
3	0.0	1.0	1.0	1.0	1.0	29.93
4	0.0	1.0	1.0	1.0	1.0	30.70
...	...	...	...	...	...	...
91	1.0	1.0	1.0	1.0	1.0	28.39
92	1.0	1.0	1.0	1.0	1.0	26.80
93	0.0	1.0	1.0	1.0	1.0	32.60
94	0.0	1.0	1.0	1.0	1.0	28.20
95	1.0	1.0	1.0	1.0	1.0	29.90

96 rows × 6 columns

Third, convert **mean_price_sqm** to three groups as following:

- 0 when below rental prices tolerance range **(mean_price_sqm < 23.75)**
- 1 when inside rental prices tolerance range **(23.75 <= mean_price_sqm <= 26.25)**
- 2 when above rental prices tolerance range **(mean_price_sqm > 26.25)**

It will be done defining a function to assign a group taking **mean_price_sqm** and **tolerance range** as parameters. Having the function defined, *pandas.DataFrame.apply* method will be used to call the function and replace **mean_price_sqm** by its assigned group. Finally the column name will be changed to **rental_group**.

In [80]:
```
# define a functon to assign price group
def convert_price(p, tmin, tmax):
    if p < tmin:
        r = 0
    elif p > tmax:
        r = 2
    else:
        r = 1

    return r

# convert the values to rental groups
df_clusters['mean_price_sqm'] = df_clusters['mean_price_sqm'].apply(lambda x : convert

# change column name
df_clusters.rename(columns={'mean_price_sqm': 'rental_group'}, inplace=True)

# print results
df_clusters
```

Out[80]:

	elementary_school	metro_station	park	pharmacy	supermarket	rental_group
0	0.0	0.0	0.0	0.0	0.0	2
1	1.0	1.0	1.0	1.0	1.0	2
2	1.0	0.0	1.0	1.0	1.0	2
3	0.0	1.0	1.0	1.0	1.0	2
4	0.0	1.0	1.0	1.0	1.0	2
...	...	...	...	...	...	...
91	1.0	1.0	1.0	1.0	1.0	2
92	1.0	1.0	1.0	1.0	1.0	2
93	0.0	1.0	1.0	1.0	1.0	2
94	0.0	1.0	1.0	1.0	1.0	2
95	1.0	1.0	1.0	1.0	1.0	2

96 rows × 6 columns

Fourth, use **DBSCAN algorithm** to cluster districts according the their similarities.

In [81]:
```
# fit DBSCAN model with Clusters dataframe
vmodel = DBSCAN(eps=0.5, min_samples=3).fit(df_clusters)

# run the model
labels = vmodel.labels_

# (cluster lables
labels
```

Out[81]:
```
array([-1,  0,  1,  2,  2,  0, -1,  2,  3,  4,  5,  6,  4,  6,  4,  4,  4,
        4,  7,  6,  4,  4,  6,  0,  0,  6, -1,  8,  7,  6,  6,  6, -1,  6,
        4,  4,  4,  4,  6,  6,  8,  4,  3,  3,  4,  4, -1, -1, -1,  3,  4,
        4,  4,  7,  8,  7,  5,  8,  4,  8,  0,  1,  1,  5,  0,  1,  1,  0,
        1,  0,  0,  1,  2,  2,  8,  0,  1,  1, -1,  1,  0,  1,  1,  0,  0,
        8,  0,  0, -1,  1,  0,  0,  0,  2,  2,  0])
```

Step 8 - Cluster Analysis

Check clusters size, it wiill be done attaching *cluster labels* to the **Clusters Dataframe** and then group data by **cluster_label** and counting the districts, *pandas.DataFrame.groupby* will be used.

In [82]:
```
# attached cluster lable to Clusters dataframe
df_clusters['cluster_label'] = labels

# group data by cluster lable
df_clusters.groupby('cluster_label').count()
```

Out[82]:

cluster_label	elementary_school	metro_station	park	pharmacy	supermarket	rental_group
-1	9	9	9	9	9	9
0	19	19	19	19	19	19
1	13	13	13	13	13	13
2	7	7	7	7	7	7
3	4	4	4	4	4	4
4	19	19	19	19	19	19
5	3	3	3	3	3	3
6	11	11	11	11	11	11
7	4	4	4	4	4	4
8	7	7	7	7	7	7

Check results in **Clusters Dataframe**, as the features are *0 / 1* for venues categories and *0 / 1 / 2* for rental group, the mean calculated per cluster labels will provide a good results overview, it will be done grouping **Clusters Dataframe** by **cluster_label** and calculating *mean*.

In [83]:
```
# group data by cluster lable
df_clusters.groupby('cluster_label').mean()
```

Out[83]:

cluster_label	elementary_school	metro_station	park	pharmacy	supermarket	rental_group
-1	0.333333	0.222222	0.333333	0.555556	0.444444	0.888889
0	1.000000	1.000000	1.000000	1.000000	1.000000	2.000000
1	1.000000	0.000000	1.000000	1.000000	1.000000	2.000000
2	0.000000	1.000000	1.000000	1.000000	1.000000	2.000000
3	1.000000	0.000000	0.000000	1.000000	1.000000	0.000000
4	1.000000	0.000000	1.000000	1.000000	1.000000	0.000000
5	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000
6	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000
7	1.000000	1.000000	0.000000	1.000000	1.000000	0.000000
8	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Analysis of mean results:

- Venues Categories (Elementary School / Metro Station / Park / Pharmacy / Supermarket)

  mean = 0: cluster does not contain any districts with such category, see elementary_school in cluster 2

mean = 1: cluster contains only districts with such category, see elementary_school in cluster 0

- Rental Group

  mean = 0: cluster contains only districts with with rental group 0, see cluster 3

  mean = 1: cluster contains only districts with with rental group 1, see cluster 5

  mean = 2: cluster contains only districts with with rental group 2, see cluster 1

- Outliers

  mean < 0: , districts which does not fit in any other cluster

The goal is to find districts containing the five categories in **Priority List** and rental prices in tolerance range, **rental group 1**. From the summary above our goal is achieved by districts in ***cluster 8***, which contains 7 districts with all the required characteristics.

Check districts in selected cluster. The **cluster_label** will be attached to **Venues Count DataFrame** allowing to check the districts characteristics.

In [84]:
```
# attach Cluster Label to Venues Count dataframe
df_venues_count['cluster_label'] = labels

# check data for selected cluster
pd.merge(left=df_venues_count[df_venues_count['cluster_label']==8], right=df_districts
```

Out[84]:

	district	elementary_school	metro_station	park	pharmacy	supermarket	cluster_label	mean_pri
0	Tatuapé	1.0	6.0	5.0	26.0	9.0	8	
1	Vila Prudente	5.0	3.0	7.0	23.0	8.0	8	
2	Santana	2.0	5.0	2.0	31.0	8.0	8	
3	Vila Maria	8.0	1.0	4.0	26.0	5.0	8	
4	Butantã	1.0	4.0	7.0	28.0	7.0	8	
5	Campo Limpo	7.0	5.0	3.0	24.0	10.0	8	
6	Jabaquara	5.0	3.0	1.0	24.0	12.0	8	

Step 9 - Create a map to show the Reults

Create a map of São Paulo pointing Districts which presents all required characteristics. Initially create a dataframe with the required information from **Districts** and **Venues Count** dataframes. It will be done with *pandas.DataFrame.merge* method.

In [85]:
```
# create Cluster Map dataframe
df_cluster_map = pd.merge(left=df_districts[['region','district','mean_price_sqm','lat
                          right=df_venues_count, how='inner', on='district')

# print results
df_cluster_map
```

Out[85]:

	region	district	mean_price_sqm	latitude	longitude	elementary_school	metro_station	park
0	Centro	Bela Vista	30.60	-23.2970	-46.0267	0.0	0.0	0.0
1	Centro	Bom Retiro	29.10	-23.5271	-46.6368	2.0	5.0	4.0
2	Centro	Cambuci	29.93	-23.5661	-46.6137	3.0	0.0	1.0
3	Centro	Consolação	29.93	-23.5578	-46.6605	0.0	4.0	1.0
4	Centro	Liberdade	30.70	-23.5552	-46.6356	0.0	5.0	1.0
...	...	...	...	...	...	...	...	...
91	Sul	Campo Grande	28.39	-23.6578	-46.6986	3.0	4.0	5.0
92	Sul	Santo Amaro	26.80	-23.6562	-46.7191	2.0	4.0	5.0
93	Sul	Moema	32.60	-23.5827	-46.6490	0.0	4.0	3.0
94	Sul	Saúde	28.20	-23.5973	-46.6359	0.0	7.0	2.0
95	Sul	Vila Mariana	29.90	-23.5892	-46.6347	1.0	5.0	2.0

96 rows × 11 columns

Important notice:

Unfortunately the Folium maps are not displayed in GitHub, they are interactive objects not supported, as an option to see the maps the Jupyter nbviewer can be used. Once it is opened just need to paste the URL of the GitHub document.

Create ***Cluster Map*** to show the results. *Folium* will be used.

In [87]:
```
# create the map
sp_map = folium.Map(location=[latsp, lngsp], width='70%', height='70%',
                    min_zoom=9, max_zoom=12, zoom_start=11, control_scale=True)

# run a loop through the dataframe creating the points (circle marks)
```

```
for d,r,mp,es,ms,pk,ph,sm,lt,ln,cl in zip(df_cluster_map['district'],df_cluster_map['r
                                          df_cluster_map['elementary_school'],df_clust
                                          df_cluster_map['pharmacy'],df_cluster_map['s
                                          df_cluster_map['longitude'],df_cluster_map['
    # create marker label
    label = '{} ({})\n Mean Rental Price={:.2f}\n El.shool={}\n M.station={}\n Park={}
    format(d,r,mp,es,ms,pk,ph,sm)

    if cl == 8:
        # create the infopoint mark
        folium.Marker(location=[lt,ln], popup=label,
                      icon=folium.Icon(color='blue',icon='info-sign')).add_to(sp_map)
    else:
        # create circle mark
        folium.CircleMarker(location=[lt,ln], popup=label, radius=5, color='gray',
                      fill=True, fill_color='gray', fill_opacity=0.3).add_to(sp_map)

# show the map
sp_map
```

Out[87]: Make this Notebook Trusted to load map: File -> Trust Notebook