Projet Big Data | E-Commerce Brézilien

Legris Corentin
M2 ACSI
06/02/19

But du projet : Je souhaite installer une plateforme e-commerce au brésil, quelle stratégie adopter ? (Choix du produit, localisation...)

Projet réalisé avec spark (2.4.0) sur jupyter notebook, donc en python (pyspark).

J'ai choisis spark car je me suis familiarisé avec durant les cours, et je l'ai utilisé avec jupyter notebook, qui est un outil aussi vu en cours, et qui permet d'importer des librairies en fonction des besoins tout en permettant de rééxecuter le code très facilement.

Reflexion

La première phase du projet a été d'analyser les différents fichiers csv qu'on nous as donné donnant diverses informations sur le e-commerce au Brésil.

On a donc directement analyser les fichiers un par un, en nommant les colonnes qui nous seront importantes pour mettre en place une stratégie.

Pas de problème de ce côté là, plusieurs questions en sont resorties telles que :

- Quel produit nous favorisons ?
- Quel qualité de produit ?
- À quel endroit ?
- Quel stratégie choisie ?

Nous allons donc répondre à ces diverses questions en utilisant spark et des librairies pour la partie graphique.

Où mettre les stocks ?

En effet, le choix de la position des stocks est cruciale, plus le stock est proche de nombreux clients, plus on pourra livrer de clients.

> L'idée est donc de calculer le nombre de client par zone, afin de déterminer quelles sont les zones les plus "hot"

In [20]:

```
import sys
from pyspark import SparkContext

sc = SparkContext.getOrCreate()
```

J'aimerais récupérer seulement les colonnes qui m'intéressent :

- Le code postale (customer_zip_code_prefix)
- La ville (customer_city)
- La région (customer_state)

Le client en particulier ne nous intéresse pas, on va additionner les occurences des codes postaux, des villes, des régions pour connaitre le nombre de client pour chacune de ces granularités.

Je vais commencer par lire le fichier CSV contenant les informations sur les clients, et je vais créé une dataframe à partir de cette lecture. Ensuite je supprime la colonne "Id" et la colonne "Id unique" qui ne me serve à rien.

Il est à noter qu'à l'aide du paramètre schema sur la lecture, je renomme directement les différentes colonnes tout en appliquant un type à ceux-ci, visuellement c'est plus parlant.

In [21]:

```
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import DoubleType, IntegerType, StringType

colonnes = StructType([
    StructField("Id", StringType()),
    StructField("Id unique", StringType()),
    StructField("Code postal", StringType()),
    StructField("Ville", StringType()),
    StructField("Region", StringType())
])

df_clients = spark.read.csv('/Users/COL/Documents/Miage\ M2/Principe\ fondamentaux\ des\ données/Données/olist_customers_dataset.csv', header=True, schema=colonnes)
df_clients = df_clients.drop("Id", "Id unique")
df_clients.show(6)
```

```
+-----------+--------------------+------+
|Code postal|               Ville|Region|
+-----------+--------------------+------+
|      14409|              franca|    SP|
|      09790|sao bernardo do c...|    SP|
|      01151|           sao paulo|    SP|
|      08775|     mogi das cruzes|    SP|
|      13056|            campinas|    SP|
|      89254|      jaragua do sul|    SC|
+-----------+--------------------+------+
only showing top 6 rows
```

Maintenant que nous avons seulement les colonnes qui nous intéressent dans une dataframe, on souhaite connaitre le nombre de client pour chacune de ces colonnes.

In [22]:

```
from pyspark.sql.functions import desc
df_clients_code_postal = df_clients.groupby("Code postal").count().sort(desc("co
unt"))
df_clients_code_postal.show(6)
```

```
+-----------+-----+
|Code postal|count|
+-----------+-----+
|      22790|  142|
|      24220|  124|
|      22793|  121|
|      24230|  117|
|      22775|  110|
|      29101|  101|
+-----------+-----+
only showing top 6 rows
```

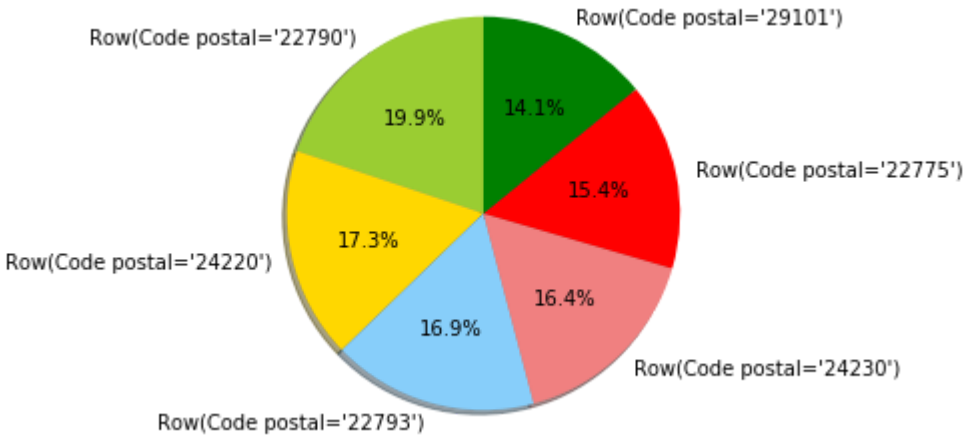
Il ne me reste plus qu'à représenter le dataframe sous forme graphique à l'aide d'une librairie (matplotlib.pyplot) afin de pouvoir analyser ces informations

In [23]:

```
import matplotlib.pyplot as plt
code_postaux = df_clients_code_postal.select("Code postal").limit(6).collect()
nombre_client_code_postal = df_clients_code_postal.select("count").limit(6).coll
ect()
couleurs = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral', 'red', 'green']

plt.pie(nombre_client_code_postal, labels=code_postaux, colors=couleurs,
        autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')
plt.show()
```



Ces données au premier abord sur les codes postaux n'est pas très parlant, j'ai donc fait une recherche pour savoir où ils se situent.

- De 20000-000 à 28999-999, ce sont les villes présent dans la region de Rio de Janeiro

On remarque ici très facilement que ce sont les codes postaux représentant la region de Rio de Janeiro qui possèdent le plus de client, donc ces code postaux sont une cible, mais ce n'est pas pour autant qu'il faut délaissé les autres localisations puisqu'une seule ville peut être séparée en plusieurs code postaux, donc la somme des clients de ces codes, soit la somme des clients d'une ville par exemple, est un autre indicateur très important, et c'est ce que nous allons voir dès maintenant.

In [24]:

```
df_clients_ville = df_clients.groupby("Ville").count().sort(desc("count"))
df_clients_ville.show(6)
```

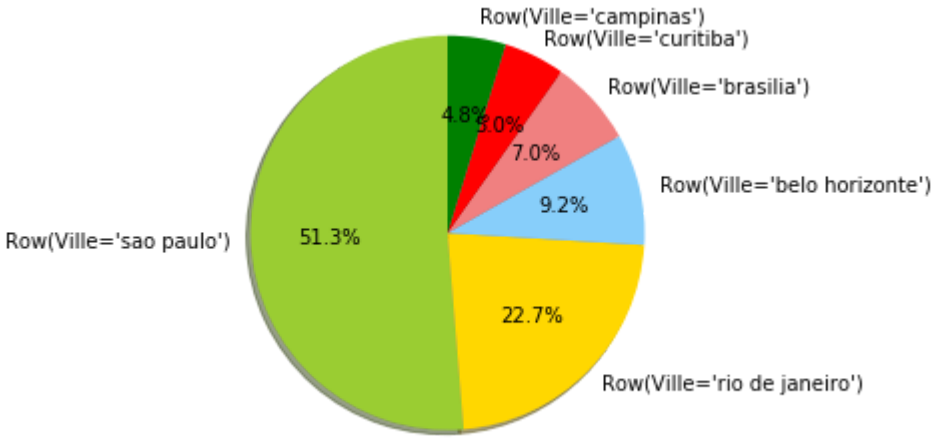
```
+--------------+-----+
|         Ville|count|
+--------------+-----+
|     sao paulo|15540|
|rio de janeiro| 6882|
|belo horizonte| 2773|
|      brasilia| 2131|
|      curitiba| 1521|
|      campinas| 1444|
+--------------+-----+
only showing top 6 rows
```

In [26]:

```
import matplotlib.pyplot as plt
villes = df_clients_ville.select("Ville").limit(6).collect()
nombre_client_ville = df_clients_ville.select("count").limit(6).collect()
couleurs = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral', 'red', 'green']

plt.pie(nombre_client_ville, labels=villes, colors=couleurs,
        autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')
plt.show()
```



Alors qu'auparavant nous avons remarqué que les code postaux de la region de Rio de Janeiro possèdent le plus de client, on remarque dorénavant que si on souhaite toucher le plus de client, il vaut mieux se rapprocher de la ville de Sao Paulo, qui possède une énorme part des clients. Il est à noté que la ville de Rio de Jeniro possède néanmoins beaucoup de clients aussi. Vu que nous possédons l'information sur la region, il est intéressant de voir la répartition sur ceux-ci, même si vu le graphique précédent, il est clair que la ville de sao paulo et de rio de janeiro vont peser dans la balance.

In [24]:

```
df_clients_region = df_clients.groupby("Region").count().sort(desc("count"))
df_clients_region.show(6)
```

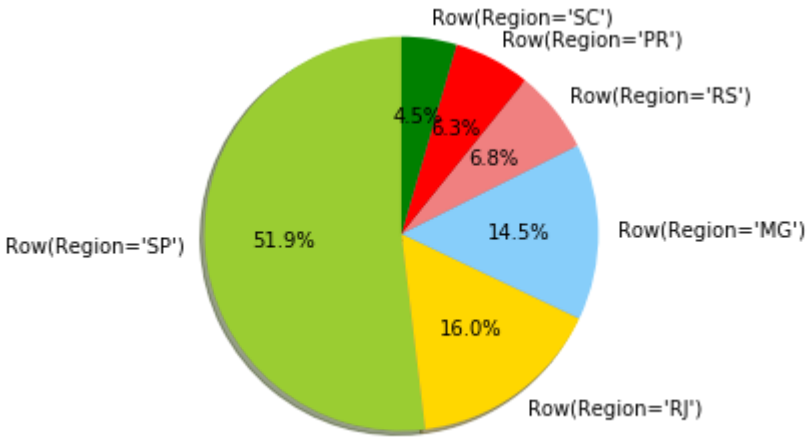
```
+------+-----+
|Region|count|
+------+-----+
|    SP|41746|
|    RJ|12852|
|    MG|11635|
|    RS| 5466|
|    PR| 5045|
|    SC| 3637|
+------+-----+
only showing top 6 rows
```

In [25]:

```
import matplotlib.pyplot as plt
regions = df_clients_region.select("Region").limit(6).collect()
nombre_client_region = df_clients_region.select("count").limit(6).collect()
couleurs = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral', 'red', 'green']

plt.pie(nombre_client_region, labels=regions, colors=couleurs,
        autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')
plt.show()
```



Sans surprise, la region de Sao Paulo est la region qui possède le plus de client, avec par la suite celle de Rio de Janeiro. Il est à noté que la région Minas Gerais possède aussi un nombre certains de clients, du probablement à la somme des clients de plusieurs villes possèdant un nombre correcte de client.

Conclusion :

Pour choisir stratégiquement où placer le stock, il va falloir prendre une décision en fonction du budget que l'on a (prix du loyer pour le stockage), et de la quantité de clients. Si on ne peut pas livrer sur de long trajet, il est préférable de se positionner à côté de zone possédant de grande quantité de clients, comme certains codes postaux de la region de rio de janeiro. Sinon, si on peut livrer sur de long trajet, se positionner sur la region de Sao Paulo est une bonne solution, car elle possède une énorme parti des clients. Si le loyer est trop chère dans ces deux cas, la region de Minas Gerais est une bonne solution, car son nombre de clients n'est pas négligeable, et est beaucoup moins connu que les deux autres.

Une localisation plus précise

On a grâce à l'analyse précédente une vague idée de où se positionner. La region de Sao Paulo et est celle qui possède le plus de clients, ainsi que la ville de Sao Paulo. Au premier abord, il serait donc une bonne idée de s'y installer, mais on ne sait pas précisément où.

On possède un fichier csv contenant les géolocalisations par rapport à un code postal. Et un client est rattaché à un code postal, donc il est possible de relié les clients à ces géolocalisations.

Je ne peux pas représenter tous les clients sur une carte, ça serait beaucoup trop long, mais il est possible d'y appliquer l'algorithme des k-means qui fonctionne sur le principe des clusters.

Sans rentrer dans le détail, au lieu de représenter l'ensemble des clients sur une carte, nous allons représenter des localisations qui seront le centre de groupes qui eux contiennent un ensemble de clients possédant des similarités, et ces groupes seront formés à l'aide de cet algorithme.

Donc chacunes des localisations qui seront représentées est une localisation qui contient un ensemble de client tout autour de celui-ci.

Pour commencer, je vais donc simplement créér les dataframes dont j'ai besoin (geolocalisation et client) pour ensuite les reliées entre eux et pour finalement n'avoir que les latitudes et longitudes de chacun des clients.

Un problème au départ était que lorsque je faisais la jointure entre les clients et la géolocalisation, je possédais un nombre immense de données. C'était due au dataframe de la geolocalisation, qui contient pour un même code postal des latitudes et longitudes différentes. Pour éviter ce problème, j'ai supprimé les duplications sur le code postal, donc cela revient à dire que je suppose que lorsque deux clients se situent dans un même code postal, ils se situent à la même position.

In [2]:

```
import folium
import pandas as pd
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import DoubleType, IntegerType, StringType, FloatType

colonnes = StructType([
    StructField("Code Postal", StringType()),
    StructField("Latitude", FloatType()),
    StructField("Longitude", FloatType())
])

df_geolocalisation = spark.read.csv('/Users/COL/Documents/Miage\ M2/Principe\ fo
ndamentaux\ des\ données/Données/olist_geolocation_dataset.csv', header=True, sc
hema=colonnes)
df_geolocalisation = df_geolocalisation.drop_duplicates(["Code Postal"])


colonnes = StructType([
    StructField("Id", StringType()),
    StructField("Id unique", StringType()),
    StructField("Code Postal", StringType()),
    StructField("Ville", StringType()),
    StructField("Region", StringType())
])

df_clients = spark.read.csv('/Users/COL/Documents/Miage\ M2/Principe\ fondamenta
ux\ des\ données/Données/olist_customers_dataset.csv', header=True, schema=colon
nes)
df_clients = df_clients.drop("Id", "Id unique")


df_clients_geolocalisation = df_clients.join(df_geolocalisation, "Code Postal")
df_clients_geolocalisation = df_clients_geolocalisation.drop("Code Postal", "Vil
le", "Region")
df_clients_geolocalisation.show()
```

```
+----------+----------+
|  Latitude| Longitude|
+----------+----------+
|-23.510427| -46.60118|
|-23.510427| -46.60118|
|-23.510427| -46.60118|
|-23.510427| -46.60118|
|-23.510427| -46.60118|
|-23.510427| -46.60118|
|-23.510427| -46.60118|
|-23.510427| -46.60118|
| -23.47649|-46.724968|
| -23.47649|-46.724968|
|-23.544317|-46.538773|
|-23.536623| -46.52901|
|-23.536623| -46.52901|
|-23.536623| -46.52901|
| -23.57776| -46.51762|
| -23.57776| -46.51762|
| -23.64743|-46.636955|
|-23.677742|-46.668648|
|-23.677742|-46.668648|
|-23.677742|-46.668648|
+----------+----------+
only showing top 20 rows
```

Après avoir ma dataframe avec une Latitude et une Longitude, j'importe toute les libraires dont je vais avoir besoin pour réaliser l'algorithme.

In [3]:

```
from __future__ import print_function

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.datasets.samples_generator import make_blobs
from pyspark import SparkContext
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
from pyspark.sql import SQLContext
```

In [4]:

```
FEATURES_COL = ['Latitude', 'Longitude']
df_clients_geolocalisation.na.drop = df_clients_geolocalisation.na.drop
```

Pour que je puisse ensuite réaliser l'algorithme, il est nécessaire de regrouper les colonnes Latitude et Longitude entre elle, avec un exemple courant dans une colonne nommée features.

In [5]:

```
vecAssembler = VectorAssembler(inputCols=FEATURES_COL, outputCol="features")
df_kmeans = vecAssembler.transform(df_clients_geolocalisation).select('features'
)
df_kmeans.show()
```

```
+--------------------+
|            features|
+--------------------+
|[-23.510427474975...|
|[-23.510427474975...|
|[-23.510427474975...|
|[-23.510427474975...|
|[-23.510427474975...|
|[-23.510427474975...|
|[-23.510427474975...|
|[-23.510427474975...|
|[-23.476490020751...|
|[-23.476490020751...|
|[-23.544317245483...|
|[-23.536623001098...|
|[-23.536623001098...|
|[-23.536623001098...|
|[-23.577760696411...|
|[-23.577760696411...|
|[-23.647430419921...|
|[-23.677742004394...|
|[-23.677742004394...|
|[-23.677742004394...|
+--------------------+
only showing top 20 rows
```

C'est la partie la plus longue de l'algorithme, celle du choix du nombre de cluster. Une méthode utilisée souvent est de lancer l'algorithme avec différentes valeurs de K, et reproduire une courbe qui réprésente le degré de séparation entre les clusters en fonction de K. Généralement on souhaite une séparation distincte, en essayant d'avoir k le plus petit possible.

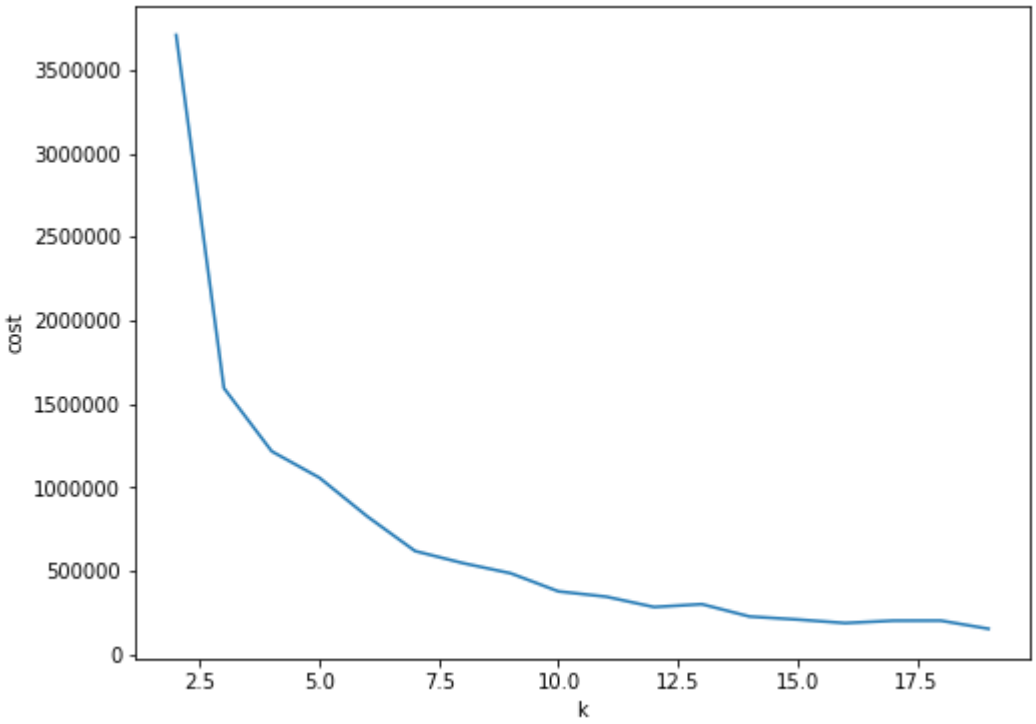
In [9]:

```
cost = np.zeros(20)
for k in range(2,20):
    kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("features")
    model = kmeans.fit(df_kmeans.sample(False,0.1, seed=42))
    cost[k] = model.computeCost(df_kmeans)

fig, ax = plt.subplots(1,1, figsize =(8,6))
ax.plot(range(2,20),cost[2:20])
ax.set_xlabel('k')
ax.set_ylabel('cost')
```

Out[9]:

```
Text(0, 0.5, 'cost')
```



Même si pour moi le coût n'est pas très explicite, je peux néanmoins en déduire de cette courbe que lorsque k atteint la valeur 12, la séparation est claire entre les clusters, et c'est exactement ce que je cherche.

Je peux donc ensuire réalisé l'algorithme avec k clusters, donc ici 12, et calculer le centre de ces clusters, ce qui nous intéresse pour la map.

In [6]:

```
k = 12
kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("features")
model = kmeans.fit(df_kmeans)
centers = model.clusterCenters()

print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Cluster Centers:
[-23.39573981 -46.67779541]
[ -6.67640557 -36.73872358]
[ -3.37521701 -46.50154581]
[-17.25112813 -55.11695126]
[-19.31783053 -41.39568888]
[-21.23721215 -49.11888868]
[-21.90062964 -43.48949855]
[ -7.38507463 -62.03660727]
[-12.60682304 -39.0807548 ]
[-15.9654008  -48.16857284]
[-29.56272391 -51.89524786]
[-25.71894332 -50.20709673]
```

Une fois que j'ai le centre de mes 12 clusters, je vais donc les représenters sur une map.

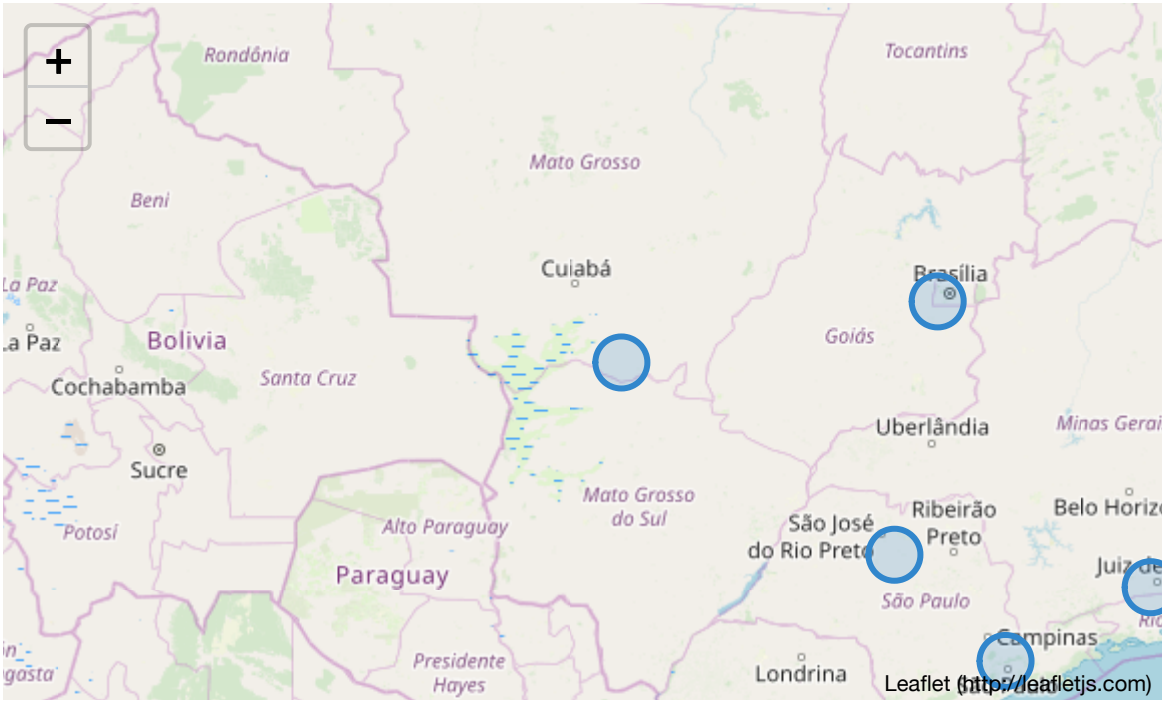
In [7]:

```
m = folium.Map(location=[-23, -46], zoom_start=5)


for index, valeur in enumerate(centers):
    folium.Circle(
        location=centers[index],
        radius=60000,
        color='#3186cc',
        fill=True,
        fill_color='#3186cc'
    ).add_to(m)

m
```

Out[7]:



Grâce à cette map, et le centre des clusters, on a donc une représentation très précise de l'endroit où on devrait chercher à se situer, si on veut un maximum de clients autour. Mais pour ajouter une précision à cette map, il serait intéressant de modifier la tailles des cercles en fonction du nombre de clients dans chaque cluster. Et pour cela, on peut utiliser la prediction, qui pour chaque géolocalisation donne une prediction sur le cluster dans lequel elle devrait faire partie.

In [8]:

```
transformed = model.transform(df_kmeans)
transformed.show()
```

```
+--------------------+----------+
|            features|prediction|
+--------------------+----------+
|[-23.510427474975...|         0|
|[-23.510427474975...|         0|
|[-23.510427474975...|         0|
|[-23.510427474975...|         0|
|[-23.510427474975...|         0|
|[-23.510427474975...|         0|
|[-23.510427474975...|         0|
|[-23.510427474975...|         0|
|[-23.476490020751...|         0|
|[-23.476490020751...|         0|
|[-23.544317245483...|         0|
|[-23.536623001098...|         0|
|[-23.536623001098...|         0|
|[-23.536623001098...|         0|
|[-23.577760696411...|         0|
|[-23.577760696411...|         0|
|[-23.647430419921...|         0|
|[-23.677742004394...|         0|
|[-23.677742004394...|         0|
|[-23.677742004394...|         0|
+--------------------+----------+
only showing top 20 rows
```

Par exemple, ici on voit les 20 premières géolocalisations (clients) qui devraient faire partie du cluster 0. Je vais donc récupérer le nombre de client pour chaque cluster.

In [9]:

```
from pyspark.sql.functions import *
tab_nombre_client = []
tab_nombre_client.append(transformed.where(col("prediction")=="0").count())
tab_nombre_client.append(transformed.where(col("prediction")=="1").count())
tab_nombre_client.append(transformed.where(col("prediction")=="2").count())
tab_nombre_client.append(transformed.where(col("prediction")=="3").count())
tab_nombre_client.append(transformed.where(col("prediction")=="4").count())
tab_nombre_client.append(transformed.where(col("prediction")=="5").count())
tab_nombre_client.append(transformed.where(col("prediction")=="6").count())
tab_nombre_client.append(transformed.where(col("prediction")=="7").count())
tab_nombre_client.append(transformed.where(col("prediction")=="8").count())
tab_nombre_client.append(transformed.where(col("prediction")=="9").count())
tab_nombre_client.append(transformed.where(col("prediction")=="10").count())
tab_nombre_client.append(transformed.where(col("prediction")=="11").count())
tab_nombre_client.append(transformed.where(col("prediction")=="12").count())
```

Il ne me reste plus qu'à afficher la map, en faisant varier le rayon de chaque cercle en fonction du nombre de client.

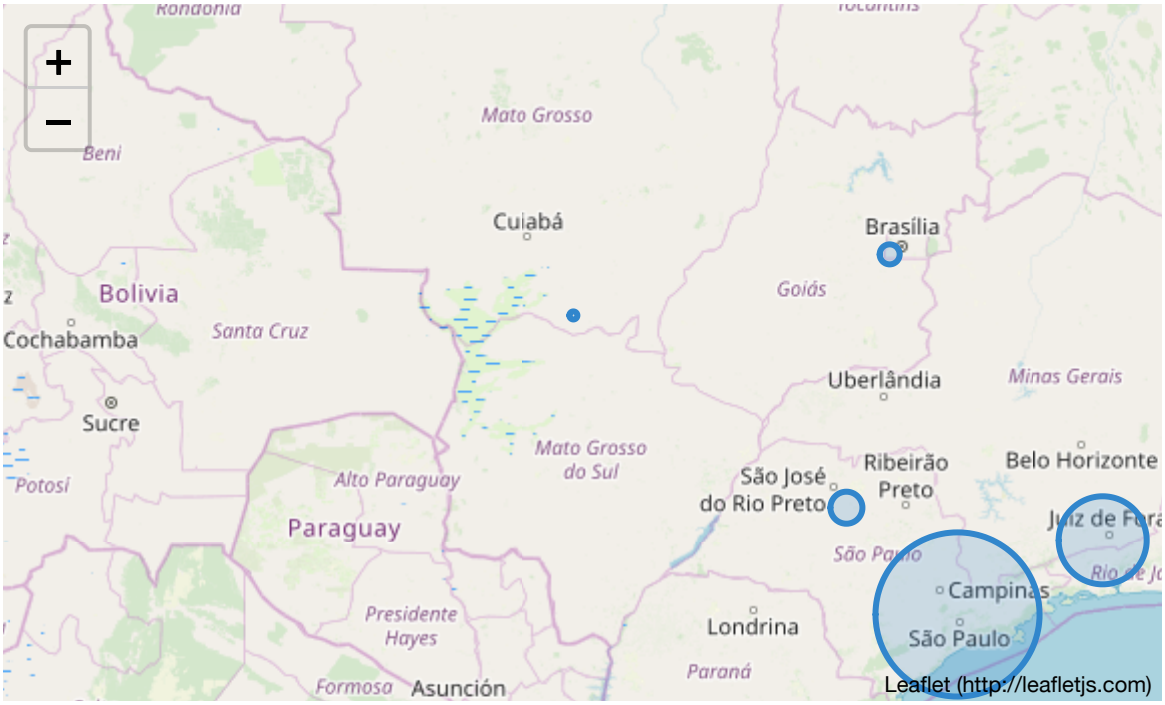
In [10]:

```
m = folium.Map(location=[-23, -46], zoom_start=5)


for index, valeur in enumerate(centers):
    folium.Circle(
        location=centers[index],
        radius=10 * tab_nombre_client[index] / 2,
        color='#3186cc',
        fill=True,
        fill_color='#3186cc'
    ).add_to(m)

m
```

Out[10]:



On a donc une géolocalisation précise des différents endroits que l'on devrait choisir pour toucher un maximum de clients. Et la map réaffirme ce qu'on avait dit précedemment, que la ville de Sao Paulo possède le plus de clients avec ensuite Rio de janeiro. Donc si on suit uniquement le nombre de clients, il est clair que privilégier la ville de Sao Paulo est une bonne stratégie.

Vers quels produits se tourner ?

Si on souhaite obtenir un maximum de clients, il est nécessaire de cibler des produits qui se vendent très bien (en terme de quantité, et de prix).

> L'idée est donc de calculer le nombre de produits et la somme générée pour chacunes des catégories de produit

In [27]:

```
import sys
from pyspark import SparkContext

sc = SparkContext.getOrCreate()
```

J'aimerais récupérer seulement les colonnes qui m'intéressent :

- Le produit (product_id)
- Le prix (price)

On va donc pouvoir addition les occurences des produits, et le prix total généré par chacun des produits, on va donc pouvoir déterminer les produits qui se vendent le plus, et ceux qui génèrent le plus d'argent.

In [28]:

```
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import DoubleType, IntegerType, StringType

colonnes = StructType([
    StructField("Id de la commande", StringType()),
    StructField("Numéro de produit", StringType()),
    StructField("Id du produit", StringType()),
    StructField("Id du vendeur", StringType()),
    StructField("Date stock", StringType()),
    StructField("Prix", DoubleType()),
    StructField("Cout de transport", StringType())
])

df_produits = spark.read.csv('/Users/COL/Documents/Miage\ M2/Principe\ fondamentaux\ des\ données/Données/olist_order_items_dataset.csv', header=True, schema=colonnes)
df_produits = df_produits.drop("Id de la commande", "Numéro de produit", "Id du vendeur", "Date stock", "Cout de transport")
df_produits.show(6)
```

```
+--------------------+-----+
|       Id du produit| Prix|
+--------------------+-----+
|4244733e06e7ecb49...| 58.9|
|e5f2d52b802189ee6...|239.9|
|c777355d18b72b67a...|199.0|
|7634da152a4610f15...|12.99|
|ac6c3623068f30de0...|199.9|
|ef92defde845ab845...| 21.9|
+--------------------+-----+
only showing top 6 rows
```

Maintenant que nous avons seulement les colonnes qui nous intéressent dans une dataframe, on souhaite connaitre le nombre de produits.

In [29]:

```
from pyspark.sql.functions import desc
df_produits_nombre = df_produits.groupby("Id du produit").count().sort(desc("cou
nt"))
df_produits_nombre.show(6)
```

```
+--------------------+-----+
|       Id du produit|count|
+--------------------+-----+
|aca2eb7d00ea1a7b8...|  527|
|99a4788cb24856965...|  488|
|422879e10f4668299...|  484|
|389d119b48cf3043d...|  392|
|368c6c730842d7801...|  388|
|53759a2ecddad2bb8...|  373|
+--------------------+-----+
only showing top 6 rows
```

Dans cette dataframe on possède donc les produits qui se vendent le plus en terme de quantité. Il serait intéressant maintenant de connaitre la somme d'argent générée totale par chacun des produits.

In [30]:

```
from pyspark.sql.types import IntegerType
df_produits_prix = df_produits.groupby("Id du produit").sum("prix").orderBy("sum
(prix)",ascending=False)
df_produits_prix.show(6)
```

```
+--------------------+------------------+
|       Id du produit|         sum(prix)|
+--------------------+------------------+
|bb50f2e236e5eea01...|           63885.0|
|6cdd53843498f9289...|54730.199999999975|
|d6160fb7873f18409...|          48899.34|
|d1c427060a0f73f6b...| 47214.50999999997|
|99a4788cb24856965...| 43025.55999999991|
|3dd2a17168ec895c7...| 41082.59999999995|
+--------------------+------------------+
only showing top 6 rows
```

Je vais maintenant relié ces deux dataframes.

In [32]:

```
df_produits_nombre_prix = df_produits_nombre.join(df_produits_prix, "Id du produ
it")
df_produits_nombre_prix.show(6)
```

```
+--------------------+-----+------------------+
|       Id du produit|count|         sum(prix)|
+--------------------+-----+------------------+
|bb50f2e236e5eea01...|  195|           63885.0|
|6cdd53843498f9289...|  156|54730.199999999975|
|d6160fb7873f18409...|   35|           48899.34|
|d1c427060a0f73f6b...|  343| 47214.50999999997|
|99a4788cb24856965...|  488| 43025.55999999991|
|3dd2a17168ec895c7...|  274| 41082.59999999995|
+--------------------+-----+------------------+
only showing top 6 rows
```

On remarque déjà que ce ne sont pas les produits vendus en plus grandes quantitées qui ont généré le plus d'argent. L'Id du produit ne nous parle pas trop, pour faire une analyse complète, nous allons utilisés les données provenant de products, qui vont donner une information sur la catégorie du produit.

In [33]:

```
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import DoubleType, IntegerType, StringType

colonnes = StructType([
    StructField("Id du produit", StringType()),
    StructField("Nom de la categorie du produit", StringType())
])

df_produits_information = spark.read.csv('/Users/COL/Documents/Miage\ M2/Princip
e\ fondamentaux\ des\ données/Données/olist_products_dataset.csv', header=True,
schema=colonnes)
df_produits_information.show(6)
```

```
+--------------------+----------------------------+
|       Id du produit|Nom de la categorie du produit|
+--------------------+----------------------------+
|1e9e8ef04dbcff454...|                   perfumaria|
|3aa071139cb16b67c...|                       artes|
|96bd76ec8810374ed...|                esporte_lazer|
|cef67bcfe19066a93...|                       bebes|
|9dc1a7de274444849...|          utilidades_domest...|
|41d3672d4792049fa...|          instrumentos_musi...|
+--------------------+----------------------------+
only showing top 6 rows
```

Le nom de nous parlant pas trop, car non traduit, nous allons utiliser les données provenant de product_category_name_translation, qui fournit le nom des différentes catégories en anglais.

In [35]:

```
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import DoubleType, IntegerType, StringType

colonnes = StructType([
    StructField("Nom de la categorie du produit", StringType()),
    StructField("Nom de la categorie du produit en anglais", StringType())
])

df_categories_produit = spark.read.csv('/Users/COL/Documents/Miage\ M2/Principe\
 fondamentaux\ des\ données/Données/product_category_name_translation.csv', head
er=True, schema=colonnes)
df_produits_information_anglais = df_categories_produit.join(df_produits_informa
tion, "Nom de la categorie du produit")
df_produits_information_anglais = df_produits_information_anglais.drop("Nom de l
a categorie du produit")
df_produits_information_anglais.show(6)
```

```
+----------------------------------------+--------------------+
|Nom de la categorie du produit en anglais|       Id du produit|
+----------------------------------------+--------------------+
|                               perfumery|1e9e8ef04dbcff454...|
|                                     art|3aa071139cb16b67c...|
|                          sports_leisure|96bd76ec8810374ed...|
|                                    baby|cef67bcfe19066a93...|
|                               housewares|9dc1a7de274444849...|
|                     musical_instruments|41d3672d4792049fa...|
+----------------------------------------+--------------------+
only showing top 6 rows
```

Il ne nous reste plus qu'à faire le lien avec df_produits_nombre_prix.

In [36]:

```
df_produits_finale = df_produits_nombre_prix.join(df_produits_information_anglais, "Id du produit").orderBy("sum(prix)",ascending=False)
df_produits_finale.show(6)
```

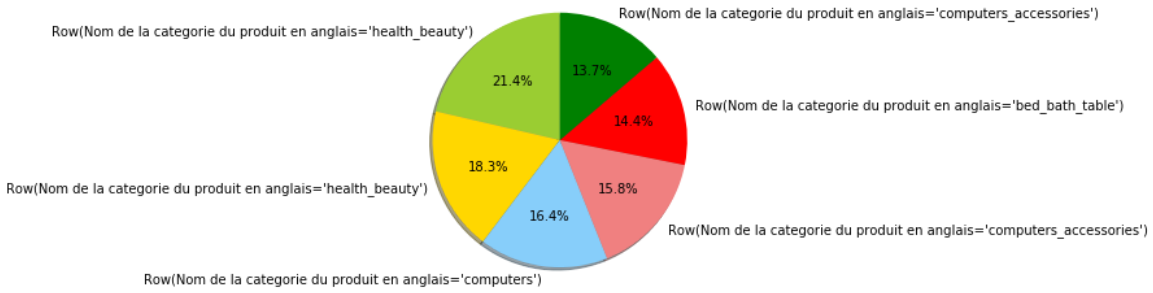
```
+--------------------+-----+------------------+--------------------
--------------------+
|       Id du produit|count|         sum(prix)|Nom de la categorie d
u produit en anglais|
+--------------------+-----+------------------+--------------------
--------------------+
|bb50f2e236e5eea01...|  195|           63885.0|
       health_beauty|
|6cdd53843498f9289...|  156|54730.199999999975|
       health_beauty|
|d6160fb7873f18409...|   35|           48899.34|
          computers|
|d1c427060a0f73f6b...|  343| 47214.50999999997|
computers_accesso...|
|99a4788cb24856965...|  488| 43025.55999999991|
      bed_bath_table|
|3dd2a17168ec895c7...|  274| 41082.59999999995|
computers_accesso...|
+--------------------+-----+------------------+--------------------
--------------------+
only showing top 6 rows
```

In [37]:

```
import matplotlib.pyplot as plt
categories = df_produits_finale.select("Nom de la categorie du produit en anglais").limit(6).collect()
somme_argent = df_produits_finale.select("sum(prix)").limit(6).collect()
couleurs = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral', 'red', 'green']

plt.pie(somme_argent, labels=categories, colors=couleurs,
        autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')
plt.show()
```



On peut donc visualiser les catégories de produits qui génèrent le plus d'argent. On remarque néanmoins qu'il y a des catégories en doublon, on va donc effectuer un traitement supplémentaire permettant de les regroupers.

In [47]:

```
df_produits_finale_sans_doublon_prix = df_produits_finale.groupBy("Nom de la categorie du produit en anglais").sum("sum(prix)").orderBy("sum(sum(prix))",ascending=False)
df_produits_finale_sans_doublon_prix.show(6)
```

```
+----------------------------------------+------------------+
|Nom de la categorie du produit en anglais|    sum(sum(prix))|
+----------------------------------------+------------------+
|                          health_beauty|1258681.3399999992|
|                          watches_gifts| 1205005.679999998|
|                        bed_bath_table|1036988.6799999995|
|                        sports_leisure| 988048.9700000003|
|                  computers_accesso...| 911954.3200000002|
|                       furniture_decor| 729762.4900000001|
+----------------------------------------+------------------+
only showing top 6 rows
```

In [48]:

```
df_produits_finale_sans_doublon_count = df_produits_finale.groupBy("Nom de la categorie du produit en anglais").sum("count").orderBy("sum(count)",ascending=False)
df_produits_finale_sans_doublon_count.show(6)
```

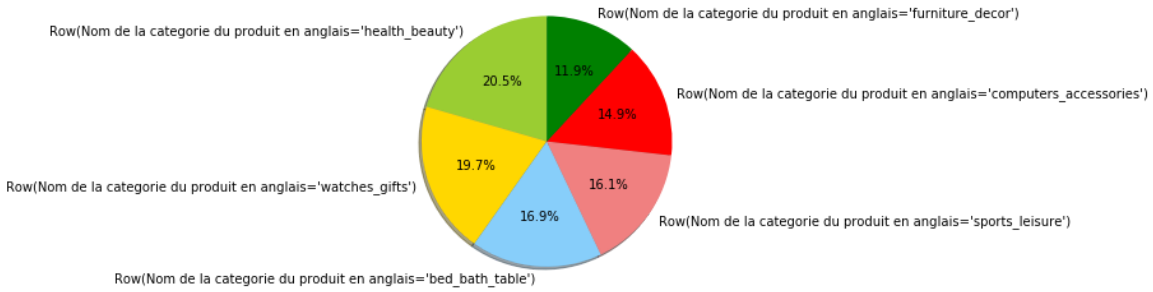
```
+----------------------------------------+----------+
|Nom de la categorie du produit en anglais|sum(count)|
+----------------------------------------+----------+
|                          bed_bath_table|     11115|
|                           health_beauty|      9670|
|                          sports_leisure|      8641|
|                         furniture_decor|      8334|
|                   computers_accesso...|      7827|
|                              housewares|      6964|
+----------------------------------------+----------+
only showing top 6 rows
```

In [41]:

```
import matplotlib.pyplot as plt
categories = df_produits_finale_sans_doublon_prix.select("Nom de la categorie du
 produit en anglais").limit(6).collect()
somme_argent = df_produits_finale_sans_doublon_prix.select("sum(sum(prix))").lim
it(6).collect()
couleurs = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral', 'red', 'green']

plt.pie(somme_argent, labels=categories, colors=couleurs,
        autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')
plt.show()
```



Le fait d'avoir regrouper les catégories entre elle est inévitable, car si on fait la comparaison avec le schéma précedent, la categorie watches_gifts n'apparaissait même pas.

Conclusion :

Il est évident que privilégier les produits de santé-beautés semble être une bonne idée vu la propotion de cette catégorie niveau prix, de la même manière que la catégorie objets de décoration qui est très présente, et achetée en très grande quantité. Malheureusement on n'en sait pas plus sur le produit (par exemple si c'est le coussin qui est le plus vendu...)

Quels moyens de paiement proposer ?

Il est très important sur notre site de proposer les moyens de paiement les plus utilisers, afin de ne rater aucun client.

> L'idée est donc de calculer les moyens de paiement les plus utilisés

In [50]:

```
import sys
from pyspark import SparkContext

sc = SparkContext.getOrCreate()
```

J'aimerais récupérer seulement la colonne qui m'intéresse :

- Le type de paiement (payment_type)

On va donc pouvoir addition les occurences des moyens de paiement, on va donc pouvoir déterminer les moyens de paiement qui sont le plus utilisés.

In [51]:

```
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import DoubleType, IntegerType, StringType

colonnes = StructType([
    StructField("Numéro de commande", StringType()),
    StructField("Numéro de moyens de paiement", StringType()),
    StructField("Moyen de paiement", StringType())
])

df_moyen_paiement = spark.read.csv('/Users/COL/Documents/Miage\ M2/Principe\ fondamentaux\ des\ données/Données/olist_order_payments_dataset.csv', header=True, schema=colonnes)
df_moyen_paiement = df_moyen_paiement.drop("Numéro de commande", "Numéro de moyens de paiement")
df_moyen_paiement.show(6)
```

```
+-----------------+
|Moyen de paiement|
+-----------------+
|      credit_card|
|      credit_card|
|      credit_card|
|      credit_card|
|      credit_card|
|      credit_card|
+-----------------+
only showing top 6 rows
```

In [52]:

```
from pyspark.sql.functions import *
df_moyen_paiement = df_moyen_paiement.groupBy("Moyen de paiement").count().sort(
desc("count"))
df_moyen_paiement = df_moyen_paiement.select(col("Moyen de paiement"), col("coun
t").alias("Nombre"))
df_moyen_paiement.show()
```

```
+-----------------+------+
|Moyen de paiement|Nombre|
+-----------------+------+
|      credit_card| 76795|
|           boleto| 19784|
|          voucher|  5775|
|       debit_card|  1529|
|      not_defined|     3|
+-----------------+------+
```

On remarque donc très facilement que c'est la carte de crédit qui est le plus utilisé, ce qui n'est pas étonnant.

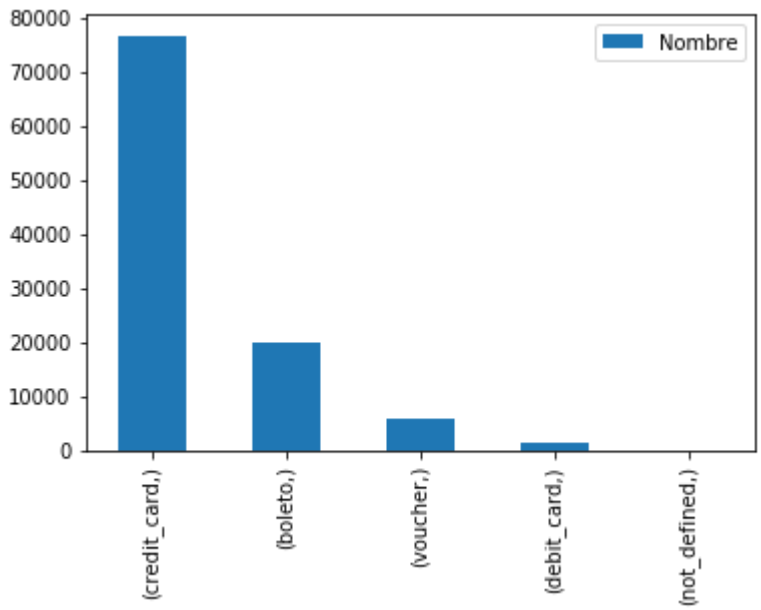
On va représenter cette information à l'aide d'un graphique, à l'aide de la librarie pandas.

In [55]:

```
import pandas as pd
tab_occurence_moyen_paiement = [int(row.Nombre) for row in df_moyen_paiement.sel
ect("Nombre").collect()]
pd_moyen_paiement_panda = pd.DataFrame({'Nombre': tab_occurence_moyen_paiement},
                                       index=df_moyen_paiement.select("Moyen de pai
ement").collect())
pd_moyen_paiement_panda.plot.bar()
```

Out[55]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1d01b4a8>
```



On remarque très facilement que c'est la carte de crédit qui est le moyen de paiement le plus utilisé, avec le chèque. Nous allons donc privilégier ces deux moyens de paiement pour toucher un maximum de client.

Conclusion :

Pour cette analyse, nous avons choisis de privilégier deux moyens de paiement à intégrer sur le site grâce à leur popularité. Néanmoins il serait possible aussi de déterminer les moyens de paiement qui incluent souvent de payer en plusieurs fois, ce qui nous permettrait de gagner de l'argent sur les intérêts (paiement en 8 fois...). Cela permettrait aussi de rassurer le client sur la possibilité d'acheter le produit.

Faire face à la concurrence ?

Même si nous avons une idée des catégories de produit à privilégier, il est important de savoir si la concurrence n'est pas trop importante, pour cela on peut vérifier par exemple si leurs produits n'ont pas une bonne image.

> L'idée est donc de calculer la moyenne des notes pour chaque catégorie de produits

In [50]:

```
import sys
from pyspark import SparkContext

sc = SparkContext.getOrCreate()
```

J'aimerais récupérer seulement la colonne qui m'intéresse :

- L'id de la commande (order_id)
- La note (review_score)

On va donc pouvoir recevoir la note de chaque commande. Il est à noté que malheureusement la note représente une note commune pour chacun des produits dedans, donc il ne représente pas forcément la qualité d'un des produits, mais vu que majoritairement les commandes possèdent une seul produit, ou que généralement on commande des produits de la même catégorie en même temps, il nous parait quand même pertinent de réalisé ce test.

In [51]:

```
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import DoubleType, IntegerType, StringType

colonnes = StructType([
    StructField("Numéro de revue", StringType()),
    StructField("Numéro de commande", StringType()),
    StructField("Note", IntegerType())
])

df_notes = spark.read.csv('/Users/COL/Documents/Miage\ M2/Principe\ fondamentaux
\ des\ données/Données/olist_order_reviews_dataset.csv', header=True, schema=col
onnes)
df_notes = df_notes.drop("Numéro de revue")
df_notes.show(6)
```

```
+--------------------+----+
|  Numéro de commande|Note|
+--------------------+----+
|73fc7af87114b3971...|   4|
|a548910a1c6147796...|   5|
|f9e4b658b201a9f2e...|   5|
|658677c97b385a9be...|   5|
|8e6bfb81e283fa7e4...|   5|
|b18dcdf73be663668...|   1|
+--------------------+----+
only showing top 6 rows
```

On possède donc le dataframe associant une note à une commande.
Il faut maintenant que je récupère les produits associés à cette note.

In [52]:

```
colonnes = StructType([
    StructField("Numéro de commande", StringType()),
    StructField("Numéro de produit", StringType()),
    StructField("Id du produit", StringType()),
    StructField("Id du vendeur", StringType()),
    StructField("Date stock", StringType()),
    StructField("Prix", DoubleType()),
    StructField("Cout de transport", StringType())
])

df_produits = spark.read.csv('/Users/COL/Documents/Miage\ M2/Principe\ fondamentaux\ des\ données/Données/olist_order_items_dataset.csv', header=True, schema=colonnes)
df_notes = df_notes.join(df_produits, "Numéro de commande")
df_notes = df_notes.drop("Id du vendeur", "Date stock", "Prix", "Cout de transport")
df_notes.show(6)
```

```
+--------------------+----+-----------------+--------------------+
|  Numéro de commande|Note|Numéro de produit|       Id du produit|
+--------------------+----+-----------------+--------------------+
|73fc7af87114b3971...|   4|                2|fd25ab760bfbba13c...|
|73fc7af87114b3971...|   4|                1|fd25ab760bfbba13c...|
|a548910a1c6147796...|   5|                1|be0dbdc3d67d55727...|
|f9e4b658b201a9f2e...|   5|                1|d1c427060a0f73f6b...|
|658677c97b385a9be...|   5|                1|52c80cedd4e90108b...|
|8e6bfb81e283fa7e4...|   5|                1|3880d25d502b15b1d...|
+--------------------+----+-----------------+--------------------+
only showing top 6 rows
```

Pour chaque numéro de commande, on a la note associée, ainsi que le produit.

Il ne nous reste plus qu'à récupérer la catégorie de chaque produit.

In [53]:

```
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import DoubleType, IntegerType, StringType

colonnes = StructType([
    StructField("Id du produit", StringType()),
    StructField("Nom de la categorie du produit", StringType())
])

df_produits_information = spark.read.csv('/Users/COL/Documents/Miage\ M2/Principe\ fondamentaux\ des\ données/Données/olist_products_dataset.csv', header=True, schema=colonnes)
df_notes = df_notes.join(df_produits_information, "Id du produit")

colonnes = StructType([
    StructField("Nom de la categorie du produit", StringType()),
    StructField("Nom de la categorie du produit en anglais", StringType())
])

df_categories_produit = spark.read.csv('/Users/COL/Documents/Miage\ M2/Principe\ fondamentaux\ des\ données/Données/product_category_name_translation.csv', header=True, schema=colonnes)
df_notes = df_notes.join(df_categories_produit, "Nom de la categorie du produit")
df_notes = df_notes.drop("Nom de la categorie du produit")

df_notes.show(6)
```

```
+--------------------+--------------------+----+-----------------+--
---------------------------------------+
|       Id du produit|  Numéro de commande|Note|Numéro de produit|No
m de la categorie du produit en anglais|
+--------------------+--------------------+----+-----------------+--
---------------------------------------+
|fd25ab760bfbba13c...|73fc7af87114b3971...|   4|                2|
                          sports_leisure|
|fd25ab760bfbba13c...|73fc7af87114b3971...|   4|                1|
                          sports_leisure|
|be0dbdc3d67d55727...|a548910a1c6147796...|   5|                1|
                    computers_accesso...|
|d1c427060a0f73f6b...|f9e4b658b201a9f2e...|   5|                1|
                    computers_accesso...|
|52c80cedd4e90108b...|658677c97b385a9be...|   5|                1|
                           garden_tools|
|3880d25d502b15b1d...|8e6bfb81e283fa7e4...|   5|                1|
                          sports_leisure|
+--------------------+--------------------+----+-----------------+--
---------------------------------------+
only showing top 6 rows
```

J'ai donc maintenant une note associée à des produits, qui eux-même sont associé à une catégorie de produit. Je vais maintenant faire une moyenne de la note pour chaque catégorie.

In [55]:

```
from pyspark.sql.functions import desc
df_notes_avg = df_notes.groupBy("Nom de la categorie du produit en anglais").avg
("Note").sort(desc("avg(Note)"))
df_notes_avg.show(6)
```

```
+----------------------------------------+-----------------+
|Nom de la categorie du produit en anglais|        avg(Note)|
+----------------------------------------+-----------------+
|                        cds_dvds_musicals|4.642857142857143|
|                     fashion_childrens...|              4.5|
|                     books_general_int...|4.439421338155515|
|                          books_imported|              4.4|
|                     costruction_tools...|4.359223300970874|
|                          books_technical|4.338289962825279|
+----------------------------------------+-----------------+
only showing top 6 rows
```

J'ai donc une moyenne des notes en fonction des différentes catégories. Les deux catégories qui nous intéressent sont "health_beauty" et "bed_bath_table" (vu précedemment).

In [61]:

```
from pyspark.sql.functions import *
df_notes_avg_health_beauty = df_notes_avg.where(col("Nom de la categorie du prod
uit en anglais")=="health_beauty")
df_notes_avg_health_beauty.show()
```

```
+----------------------------------------+-----------------+
|Nom de la categorie du produit en anglais|        avg(Note)|
+----------------------------------------+-----------------+
|                            health_beauty|4.124280427631579|
+----------------------------------------+-----------------+
```

In [62]:

```
from pyspark.sql.functions import *
df_notes_avg_health_beauty = df_notes_avg.where(col("Nom de la categorie du prod
uit en anglais")=="bed_bath_table")
df_notes_avg_health_beauty.show()
```

```
+----------------------------------------+-----------------+
|Nom de la categorie du produit en anglais|        avg(Note)|
+----------------------------------------+-----------------+
|                           bed_bath_table|3.871185237757275|
+----------------------------------------+-----------------+
```

On remarque donc que les produits provenant des deux catégories ont plutôt une bonne image, donc la concurrence est forte, il va être difficile de se faire une place parmis les produits de qualités déjà présent, mais ce n'est pas impossible.

Conclusion :

Les deux catégories de produits choisis ont plutôt une bonne moyenne de note (au dessus de la moyenne), mais ce n'est pas étonnant vu qu'ils font partis des meilleurs catégories de produits qui se vendent. Si on souhaite s'intégrer au marché, il va falloir proposer des produits de meilleurs qualités, ou de même qualité, mais on va pouvoir aussi s'intégrer en proposant des produits différents de ce qui se fait dans ces catégories.

Et la localisation de la concurrence ?

On a estimé la localisation que l'on pourrait choisir pour notre stock, mais on peut comparé notre choix avec celui des concurrents, et sécuriser notre choix.

> L'idée est donc de calculer le nombre de vendeurs par zone, pour analyser leur stratégie

In [65]:

```
import sys
from pyspark import SparkContext

sc = SparkContext.getOrCreate()
```

J'aimerais récupérer seulement les colonnes qui m'intéressent :

- Le code postal du vendeur (seller_zip_code_prefix)
- La ville du vendeur (seller_city)
- La région du vendeur (seller_state)

De la même manière que pour les clients, on va additionner les occurences des codes postaux, des villes, des régions pour connaitre le nombre de vendeurs pour chacune de ces granularités.

In [67]:

```
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import DoubleType, IntegerType, StringType

colonnes = StructType([
    StructField("Id du vendeur", StringType()),
    StructField("Code postal", StringType()),
    StructField("Ville", StringType()),
    StructField("Region", StringType())
])

df_vendeurs = spark.read.csv('/Users/COL/Documents/Miage\ M2/Principe\ fondamentaux\ des\ données/Données/olist_sellers_dataset.csv', header=True, schema=colonnes)
df_vendeurs = df_vendeurs.drop("Id du vendeur")
df_vendeurs.show(6)
```

```
+-----------+----------------+------+
|Code postal|           Ville|Region|
+-----------+----------------+------+
|      13023|        campinas|    SP|
|      13844|      mogi guacu|    SP|
|      20031|  rio de janeiro|    RJ|
|      04195|       sao paulo|    SP|
|      12914|braganca paulista|    SP|
|      20920|  rio de janeiro|    RJ|
+-----------+----------------+------+
only showing top 6 rows
```

Maintenant que nous avons seulement les colonnes qui nous intéressent dans une dataframe, on souhaite connaitre le nombre de vendeurs pour chacune de ces colonnes.

In [68]:

```
from pyspark.sql.functions import desc
df_vendeurs_code_postal = df_vendeurs.groupby("Code postal").count().sort(desc("count"))
df_vendeurs_code_postal.show(6)
```

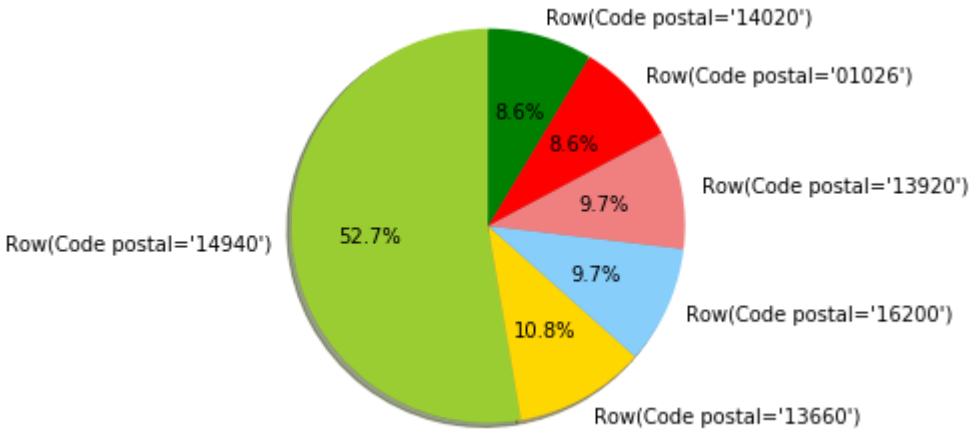
```
+-----------+-----+
|Code postal|count|
+-----------+-----+
|      14940|   49|
|      13660|   10|
|      13920|    9|
|      16200|    9|
|      01026|    8|
|      14020|    8|
+-----------+-----+
only showing top 6 rows
```

In [70]:

```
import matplotlib.pyplot as plt
code_postaux = df_vendeurs_code_postal.select("Code postal").limit(6).collect()
nombre_vendeurs_code_postal = df_vendeurs_code_postal.select("count").limit(6).c
ollect()
couleurs = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral', 'red', 'green']

plt.pie(nombre_client_code_postal, labels=code_postaux, colors=couleurs,
        autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')
plt.show()
```



Il y a énormément de vendeurs situé dans le code postal **14940**. C'est le code postal de Ibitinga, ville provenant de la region de Sao Paulo. Ce choix est compréhensible, car au finale la grande partie des clients se situent dans la region de Sao Paulo.

In [71]:

```
df_vendeurs_ville = df_vendeurs.groupby("Ville").count().sort(desc("count"))
df_vendeurs_ville.show(6)
```

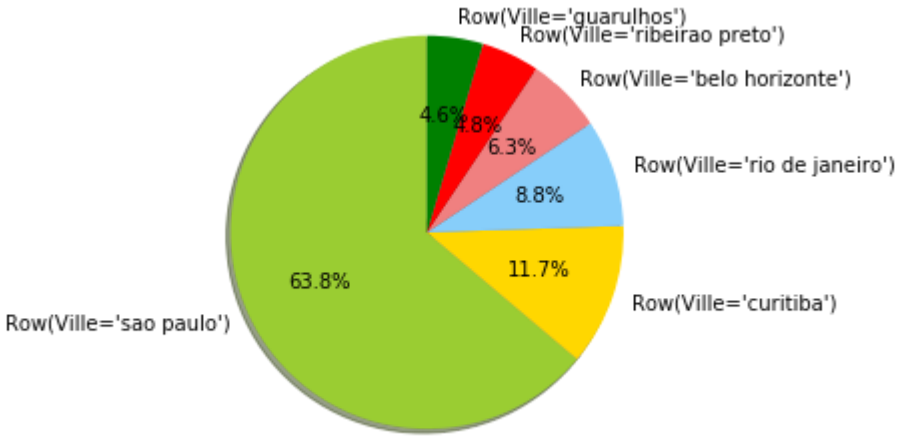
```
+--------------+-----+
|         Ville|count|
+--------------+-----+
|     sao paulo|  694|
|      curitiba|  127|
|rio de janeiro|   96|
|belo horizonte|   68|
|ribeirao preto|   52|
|     guarulhos|   50|
+--------------+-----+
only showing top 6 rows
```

In [73]:

```
import matplotlib.pyplot as plt
villes = df_vendeurs_ville.select("Ville").limit(6).collect()
nombre_vendeurs_ville = df_vendeurs_ville.select("count").limit(6).collect()
couleurs = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral', 'red', 'green']

plt.pie(nombre_vendeurs_ville, labels=villes, colors=couleurs,
        autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')
plt.show()
```



Ce graphique ne propose toujours pas un résultat inattendu, la ville de Sao Paulo est celle qui comprend le plus de vendeurs. Il est quand même à noter que très peu de vendeurs sont présent dans la ville de rio de janeiro, alors que pourtant il y a un nombre assez important de clients situés las-bas.

In [74]:

```
df_vendeurs_region = df_vendeurs.groupby("Region").count().sort(desc("count"))
df_vendeurs_region.show(6)
```

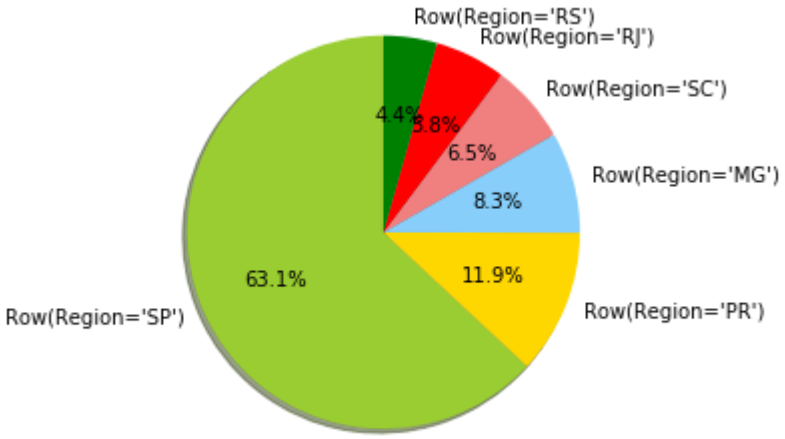
```
+------+-----+
|Region|count|
+------+-----+
|    SP| 1849|
|    PR|  349|
|    MG|  244|
|    SC|  190|
|    RJ|  171|
|    RS|  129|
+------+-----+
only showing top 6 rows
```

In [75]:

```
import matplotlib.pyplot as plt
regions = df_vendeurs_region.select("Region").limit(6).collect()
nombre_vendeurs_region = df_vendeurs_region.select("count").limit(6).collect()
couleurs = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral', 'red', 'green']

plt.pie(nombre_vendeurs_region, labels=regions, colors=couleurs,
        autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')
plt.show()
```



La region de Sao Paulo est la plus représentée en terme de vendeurs, comme vu précedemment. Rio de Janeiro est quand à elle très peut représentée.

Conclusion :

La stratégie de localisation choisie par les vendeurs suit plutôt bien notre première idée, il y a de nombreux clients dans cette zone, mais justement cela pose peut-être un problème de concurrence ? Il est possible pour nous de nous localiser vers la region de Sao Paulo, qui comporte aussi de nombreux clients, et surtout qui semblent être délaissé par les autres vendeurs.

Conclusion Projet Big Data | E-Commerce Brézilien

Le projet a été très intéressant côté spark et côté python a réalisé. On a pu découvrir la puissance de spark, de comment on pouvait facilement manipuler les données avec, et le côté python qui a permis sans logiciel comme Tableau de pouvoir avoir des rendus graphique (la map par exemple).

L'idée pour moi dans ce projet était surtout de manipuler ces deux technologies, de découvrir des choses, c'est d'ailleurs pour cela que je n'ai pas utiliser Tableau pour des rendus graphique par exemple. Et je trouve aussi que de pouvoir avoir des rendus graphique sans un autre logiciel est une bonne pratique.

Le bémol du projet se porte sur le côté analyse de la stratégie. C'était très intéressant de voir avec mes collègues économistes les indicateurs qui allaient déterminer notre stratégie pour mettre en place notre e-commerce, mais vu qu'il n'y a pas eu de suivi ensuite, ce n'était pas évident de choisir par exemple si il vallait mieux par exemple privilégier les produits qui se vendent bien, ou au contraire privilégier les produits qui se vendent mal pour les mettres en valeur et créer une nouvelle tendance.

Parmis les indicateurs qui ont été analysés, je considère que c'est les plus importants pour déterminer une stratégie, mais il est évident qu'on aurait pu analyser encore beaucoup d'autres paramètres.