Assignment 1

Team: 18 2016 fall Mingxuan Han, Yu Fu, Teng Jin

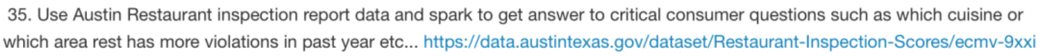
Jupyter notebook link:

https://console.ng.bluemix.net/data/notebooks/e560d4b7-25ee-460b-95b3-2793b318cae7/view?access_token=465d3f3a8a5ce27da400c00da5ba1b68f0729e4e5c14511dbce0262d5db9b4bf

Introduction:

In Bluemix Spark, we acquire an Apache Spark Service.

We chose a use case



35. Use Austin Restaurant inspection report data and spark to get answer to critical consumer questions such as which cuisine or which area rest has more violations in past year etc... https://data.austintexas.gov/dataset/Restaurant-Inspection-Scores/ecmv-9xxi

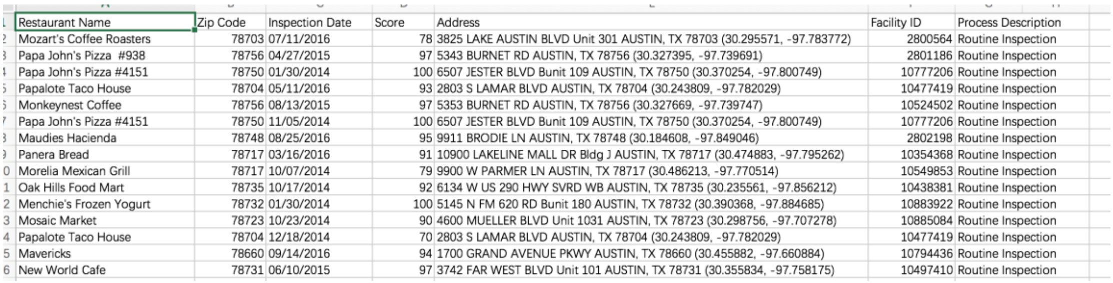
Implementation:

We did the assignment in 5 major steps.

1. Data reschedule.

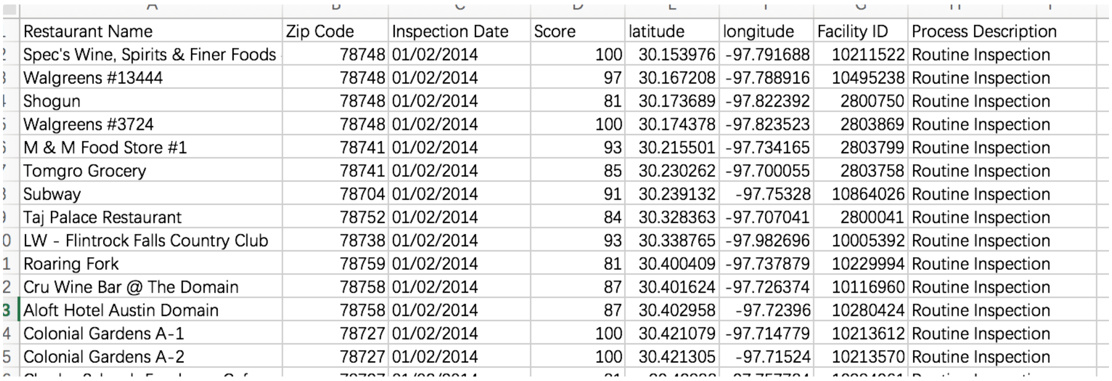
First we download the csv file and reschedule the data.

The original data is like this.



	Restaurant Name	Zip Code	Inspection Date	Score	Address	Facility ID	Process Description
1	Restaurant Name	Zip Code	Inspection Date	Score	Address	Facility ID	Process Description
2	Mozart's Coffee Roasters	78703	07/11/2016	78	3825 LAKE AUSTIN BLVD Unit 301 AUSTIN, TX 78703 (30.295571, -97.783772)	2800564	Routine Inspection
3	Papa John's Pizza #938	78756	04/27/2015	97	5343 BURNET RD AUSTIN, TX 78756 (30.327395, -97.739691)	2801186	Routine Inspection
4	Papa John's Pizza #4151	78750	01/30/2014	100	6507 JESTER BLVD Bunit 109 AUSTIN, TX 78750 (30.370254, -97.800749)	10777206	Routine Inspection
5	Papalote Taco House	78704	05/11/2016	93	2803 S LAMAR BLVD AUSTIN, TX 78704 (30.243809, -97.782029)	10477419	Routine Inspection
6	Monkeynest Coffee	78756	08/13/2015	97	5353 BURNET RD AUSTIN, TX 78756 (30.327669, -97.739747)	10524502	Routine Inspection
7	Papa John's Pizza #4151	78750	11/05/2014	100	6507 JESTER BLVD Bunit 109 AUSTIN, TX 78750 (30.370254, -97.800749)	10777206	Routine Inspection
8	Maudies Hacienda	78748	08/25/2016	95	9911 BRODIE LN AUSTIN, TX 78748 (30.184608, -97.849046)	2802198	Routine Inspection
9	Panera Bread	78717	03/16/2016	91	10900 LAKELINE MALL DR Bldg J AUSTIN, TX 78717 (30.474883, -97.795262)	10354368	Routine Inspection
10	Morelia Mexican Grill	78717	10/07/2014	79	9900 W PARMER LN AUSTIN, TX 78717 (30.486213, -97.770514)	10549853	Routine Inspection
11	Oak Hills Food Mart	78735	10/17/2014	92	6134 W US 290 HWY SVRD WB AUSTIN, TX 78735 (30.235561, -97.856212)	10438381	Routine Inspection
12	Menchie's Frozen Yogurt	78732	01/30/2014	100	5145 N FM 620 RD Bunit 180 AUSTIN, TX 78732 (30.390368, -97.884685)	10883922	Routine Inspection
13	Mosaic Market	78723	10/23/2014	90	4600 MUELLER BLVD Unit 1031 AUSTIN, TX 78723 (30.298756, -97.707278)	10885084	Routine Inspection
14	Papalote Taco House	78704	12/18/2014	70	2803 S LAMAR BLVD AUSTIN, TX 78704 (30.243809, -97.782029)	10477419	Routine Inspection
15	Mavericks	78660	09/14/2016	94	1700 GRAND AVENUE PKWY AUSTIN, TX 78660 (30.455882, -97.660884)	10794436	Routine Inspection
16	New World Cafe	78731	06/10/2015	97	3742 FAR WEST BLVD Unit 101 AUSTIN, TX 78731 (30.355834, -97.758175)	10497410	Routine Inspection

The problem is the position (30.25571, -97.783772) can be saved in Spark DataFrame, but the format is string with the long address together and hard to be used to draw a map. At this point we reform the data and it displays like followed.

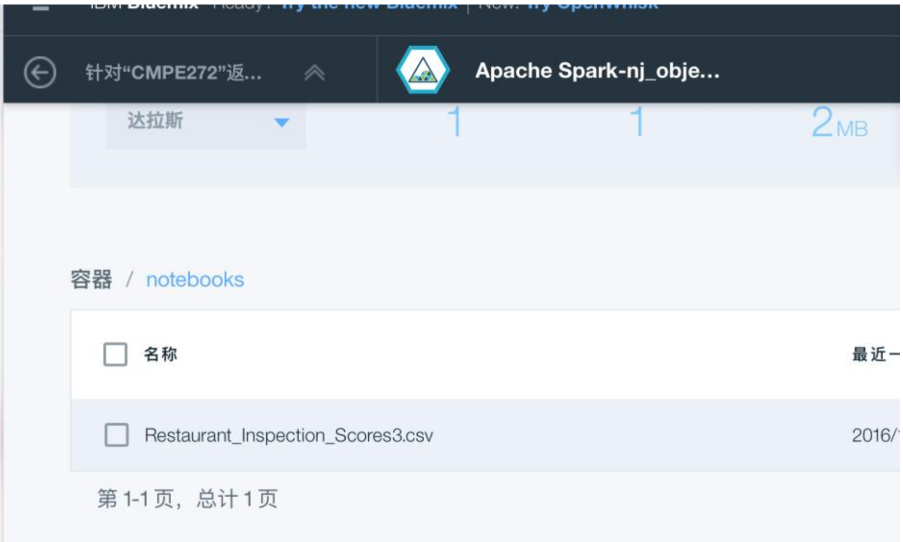


Restaurant Name	Zip Code	Inspection Date	Score	latitude	longitude	Facility ID	Process Description
Spec's Wine, Spirits & Finer Foods	78748	01/02/2014	100	30.153976	-97.791688	10211522	Routine Inspection
Walgreens #13444	78748	01/02/2014	97	30.167208	-97.788916	10495238	Routine Inspection
Shogun	78748	01/02/2014	81	30.173689	-97.822392	2800750	Routine Inspection
Walgreens #3724	78748	01/02/2014	100	30.174378	-97.823523	2803869	Routine Inspection
M & M Food Store #1	78741	01/02/2014	93	30.215501	-97.734165	2803799	Routine Inspection
Tomgro Grocery	78741	01/02/2014	85	30.230262	-97.700055	2803758	Routine Inspection
Subway	78704	01/02/2014	91	30.239132	-97.75328	10864026	Routine Inspection
Taj Palace Restaurant	78752	01/02/2014	84	30.328363	-97.707041	2800041	Routine Inspection
LW - Flintrock Falls Country Club	78738	01/02/2014	93	30.338765	-97.982696	10005392	Routine Inspection
Roaring Fork	78759	01/02/2014	81	30.400409	-97.737879	10229994	Routine Inspection
Cru Wine Bar @ The Domain	78758	01/02/2014	87	30.401624	-97.726374	10116960	Routine Inspection
Aloft Hotel Austin Domain	78758	01/02/2014	87	30.402958	-97.72396	10280424	Routine Inspection
Colonial Gardens A-1	78727	01/02/2014	100	30.421079	-97.714779	10213612	Routine Inspection
Colonial Gardens A-2	78727	01/02/2014	100	30.421305	-97.71524	10213570	Routine Inspection

Now the data Score, Latitude, Longitude are all perfect to use.

2. Upload the data

We found the data is better to be saved on cloud like this.

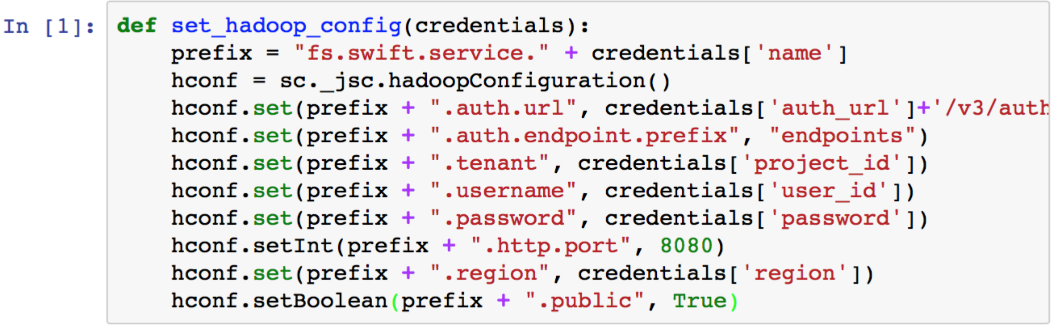


容器 / notebooks

□ 名称	最近一
□ Restaurant_Inspection_Scores3.csv	2016/

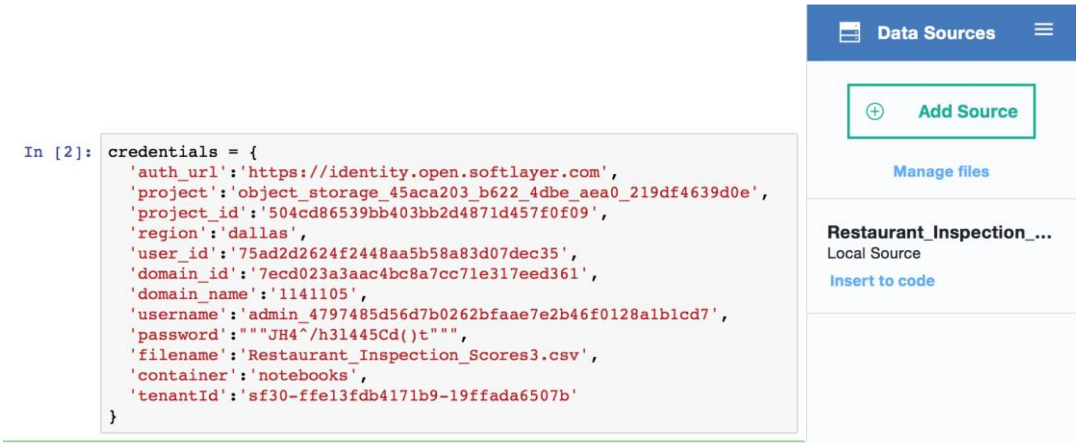
第 1-1 页，总计 1 页

then we configured the Hadoop like this.



```
In [1]: def set_hadoop_config(credentials):
            prefix = "fs.swift.service." + credentials['name']
            hconf = sc._jsc.hadoopConfiguration()
            hconf.set(prefix + ".auth.url", credentials['auth_url']+'/v3/auth
            hconf.set(prefix + ".auth.endpoint.prefix", "endpoints")
            hconf.set(prefix + ".tenant", credentials['project_id'])
            hconf.set(prefix + ".username", credentials['user_id'])
            hconf.set(prefix + ".password", credentials['password'])
            hconf.setInt(prefix + ".http.port", 8080)
            hconf.set(prefix + ".region", credentials['region'])
            hconf.setBoolean(prefix + ".public", True)
```

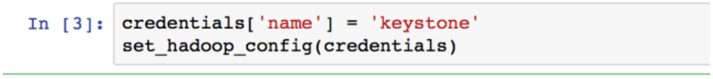
then we assert the data which saved on the cloud to this Spark notebook.



```
In [2]: credentials = {
          'auth_url':'https://identity.open.softlayer.com',
          'project':'object_storage_45aca203_b622_4dbe_aea0_219df4639d0e',
          'project_id':'504cd86539bb403bb2d4871d457f0f09',
          'region':'dallas',
          'user_id':'75ad2d2624f2448aa5b58a83d07dec35',
          'domain_id':'7ecd023a3aac4bc8a7cc71e317eed361',
          'domain_name':'1141105',
          'username':'admin_4797485d56d7b0262bfaae7e2b46f0128a1b1cd7',
          'password':"""JH4^/h31445Cd()t""",
          'filename':'Restaurant_Inspection_Scores3.csv',
          'container':'notebooks',
          'tenantId':'sf30-ffe13fdb4171b9-19ffada6507b'
        }
```

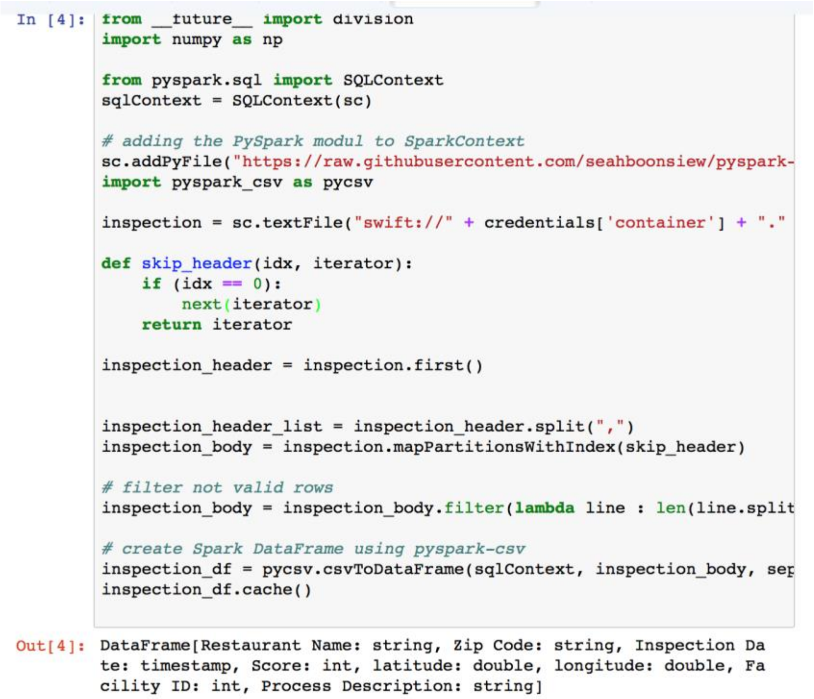
when click the “insert to code” button the code is automatically generated, and we run this part of code, the data form is included.

Then we configured the form like this.



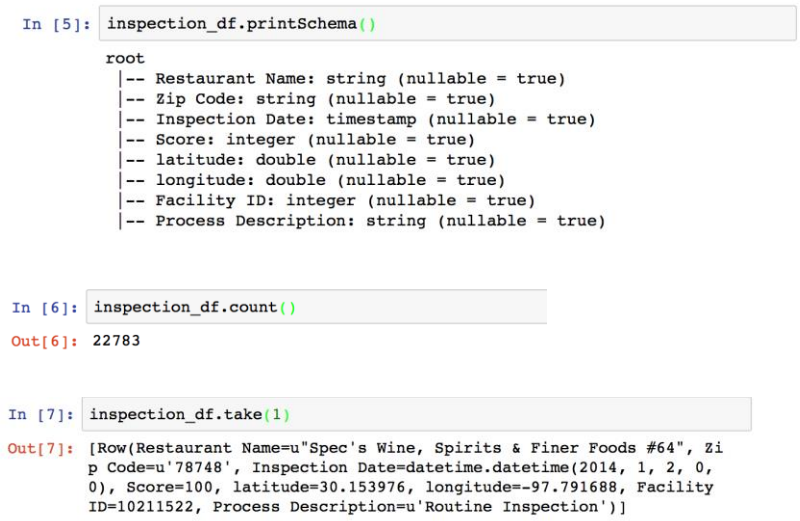
```
In [3]: credentials['name'] = 'keystone'
        set_hadoop_config(credentials)
```

The next part is the most important one. We use pyspark_csv and SQLContext (pyspark.sql). After finish the process the data is stored in the inspection_df. The out[4] shows the DataFrame. We are happy to see the latitude and longitude are type double as we expected.



```
In [4]: from __future__ import division
        import numpy as np

        from pyspark.sql import SQLContext
        sqlContext = SQLContext(sc)

        # adding the PySpark modul to SparkContext
        sc.addPyFile("https://raw.githubusercontent.com/seahboonsiew/pyspark-
        import pyspark_csv as pycsv

        inspection = sc.textFile("swift://" + credentials['container'] + "."

        def skip_header(idx, iterator):
            if (idx == 0):
                next(iterator)
            return iterator

        inspection_header = inspection.first()


        inspection_header_list = inspection_header.split(",")
        inspection_body = inspection.mapPartitionsWithIndex(skip_header)

        # filter not valid rows
        inspection_body = inspection_body.filter(lambda line : len(line.split

        # create Spark DataFrame using pyspark-csv
        inspection_df = pycsv.csvToDataFrame(sqlContext, inspection_body, sep
        inspection_df.cache()

Out[4]: DataFrame[Restaurant Name: string, Zip Code: string, Inspection Da
        te: timestamp, Score: int, latitude: double, longitude: double, Fa
        cility ID: int, Process Description: string]
```

3. Check the data in DataFrame.

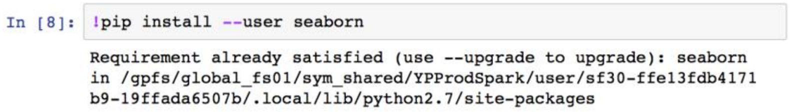


```
In [5]: inspection_df.printSchema()

        root
         |-- Restaurant Name: string (nullable = true)
         |-- Zip Code: string (nullable = true)
         |-- Inspection Date: timestamp (nullable = true)
         |-- Score: integer (nullable = true)
         |-- latitude: double (nullable = true)
         |-- longitude: double (nullable = true)
         |-- Facility ID: integer (nullable = true)
         |-- Process Description: string (nullable = true)
```

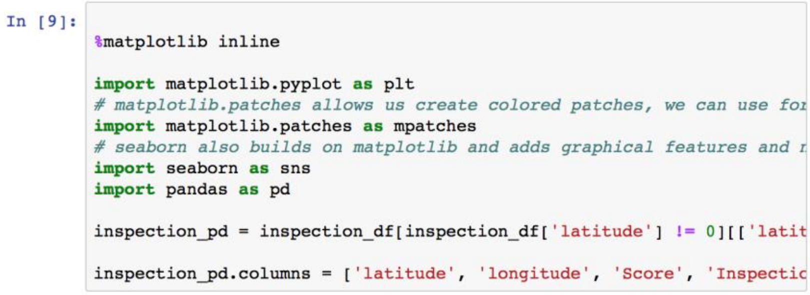
```
In [6]: inspection_df.count()

Out[6]: 22783
```

```
In [7]: inspection_df.take(1)

Out[7]: [Row(Restaurant Name=u"Spec's Wine, Spirits & Finer Foods #64", Zi
        p Code=u'78748', Inspection Date=datetime.datetime(2014, 1, 2, 0,
        0), Score=100, latitude=30.153976, longitude=-97.791688, Facility
        ID=10211522, Process Description=u'Routine Inspection')]
```

4. Preparation for draw

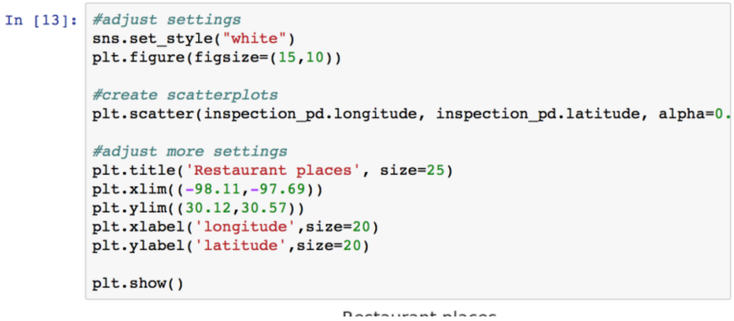


```
In [8]: !pip install --user seaborn

        Requirement already satisfied (use --upgrade to upgrade): seaborn
        in /gpfs/global_fs01/sym_shared/YPProdSpark/user/sf30-ffe13fdb4171
        b9-19ffada6507b/.local/lib/python2.7/site-packages
```

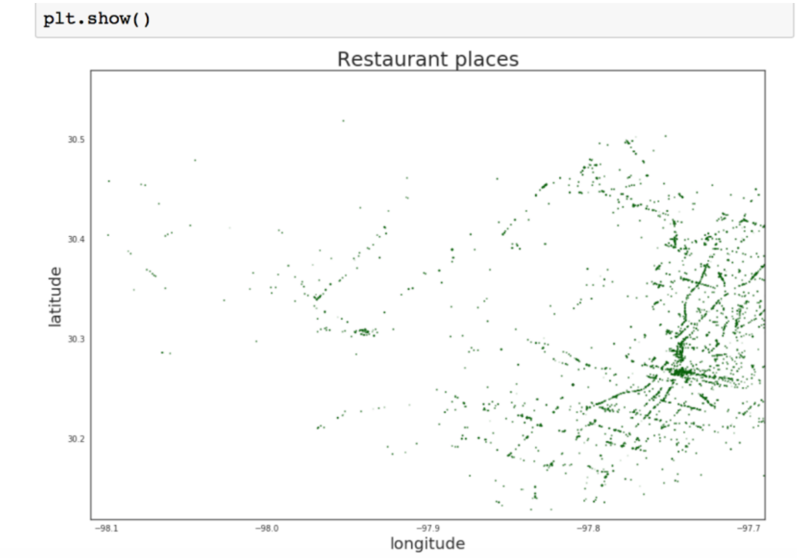


```
In [9]:
        %matplotlib inline

        import matplotlib.pyplot as plt
        # matplotlib.patches allows us create colored patches, we can use for
        import matplotlib.patches as mpatches
        # seaborn also builds on matplotlib and adds graphical features and r
        import seaborn as sns
        import pandas as pd

        inspection_pd = inspection_df[inspection_df['latitude'] != 0][['latit

        inspection_pd.columns = ['latitude', 'longitude', 'Score', 'Inspectic
```

5. Now to draw

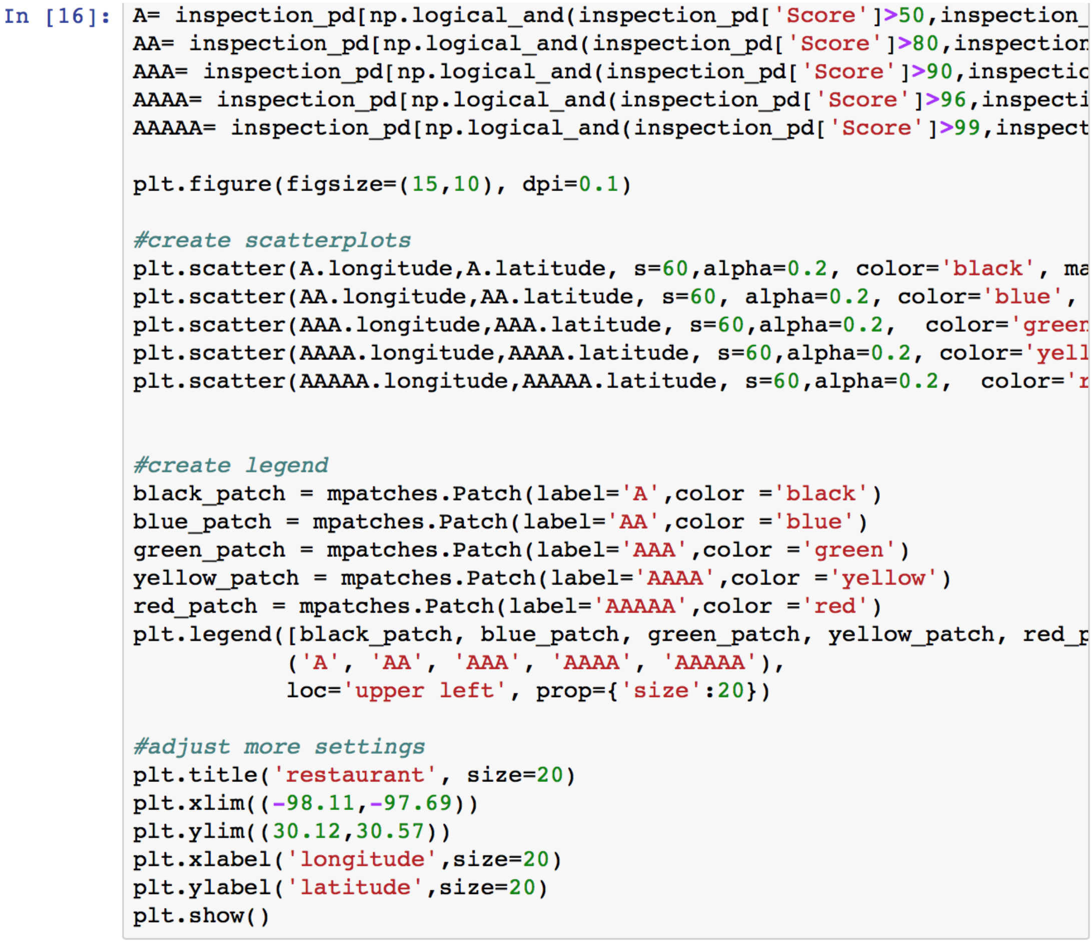


```
In [13]: #adjust settings
         sns.set_style("white")
         plt.figure(figsize=(15,10))

         #create scatterplots
         plt.scatter(inspection_pd.longitude, inspection_pd.latitude, alpha=0.

         #adjust more settings
         plt.title('Restaurant places', size=25)
         plt.xlim((-98.11,-97.69))
         plt.ylim((30.12,30.57))
         plt.xlabel('longitude',size=20)
         plt.ylabel('latitude',size=20)

         plt.show()
```

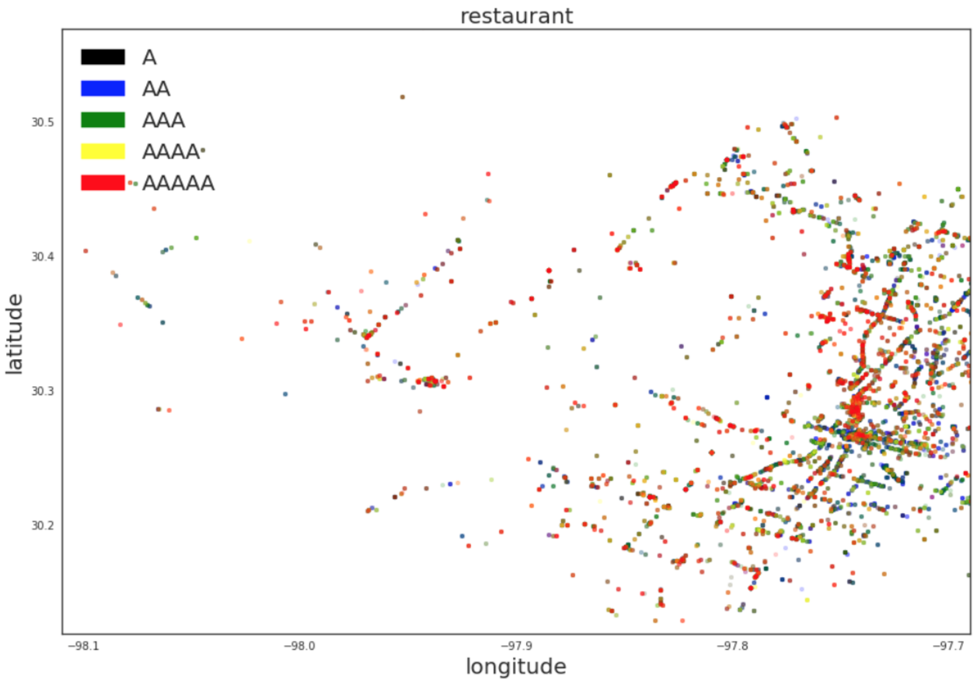


```
plt.show()
```

after we choose score as standard to draw an advanced map.



```
In [16]: A= inspection_pd[np.logical_and(inspection_pd['Score']>50,inspection_
         AA= inspection_pd[np.logical_and(inspection_pd['Score']>80,inspection
         AAA= inspection_pd[np.logical_and(inspection_pd['Score']>90,inspectic
         AAAA= inspection_pd[np.logical_and(inspection_pd['Score']>96,inspecti
         AAAAA= inspection_pd[np.logical_and(inspection_pd['Score']>99,inspect

         plt.figure(figsize=(15,10), dpi=0.1)

         #create scatterplots
         plt.scatter(A.longitude,A.latitude, s=60,alpha=0.2, color='black', ma
         plt.scatter(AA.longitude,AA.latitude, s=60, alpha=0.2, color='blue',
         plt.scatter(AAA.longitude,AAA.latitude, s=60,alpha=0.2,  color='greer
         plt.scatter(AAAA.longitude,AAAA.latitude, s=60,alpha=0.2, color='yell
         plt.scatter(AAAAA.longitude,AAAAA.latitude, s=60,alpha=0.2,  color='r


         #create legend
         black_patch = mpatches.Patch(label='A',color ='black')
         blue_patch = mpatches.Patch(label='AA',color ='blue')
         green_patch = mpatches.Patch(label='AAA',color ='green')
         yellow_patch = mpatches.Patch(label='AAAA',color ='yellow')
         red_patch = mpatches.Patch(label='AAAAA',color ='red')
         plt.legend([black_patch, blue_patch, green_patch, yellow_patch, red_p
                    ('A', 'AA', 'AAA', 'AAAA', 'AAAAA'),
                    loc='upper left', prop={'size':20})

         #adjust more settings
         plt.title('restaurant', size=20)
         plt.xlim((-98.11,-97.69))
         plt.ylim((30.12,30.57))
         plt.xlabel('longitude',size=20)
         plt.ylabel('latitude',size=20)
         plt.show()
```



Conclusion:

We are amazing by the speed of spark when it dealing with this much data. The speed is really fast and the API database control design is a really good way for programmers. It is really easy to use so we can quickly be familiar to it.

Reference code:

```
def set_hadoop_config(credentials):
    prefix = "fs.swift.service." + credentials['name']
    hconf = sc._jsc.hadoopConfiguration()
    hconf.set(prefix + ".auth.url", credentials['auth_url']+'/v3/auth/tokens')
    hconf.set(prefix + ".auth.endpoint.prefix", "endpoints")
    hconf.set(prefix + ".tenant", credentials['project_id'])
    hconf.set(prefix + ".username", credentials['user_id'])
    hconf.set(prefix + ".password", credentials['password'])
    hconf.setInt(prefix + ".http.port", 8080)
    hconf.set(prefix + ".region", credentials['region'])
    hconf.setBoolean(prefix + ".public", True)
```

```
credentials['name'] = 'keystone'
set_hadoop_config(credentials)
```

```
from __future__ import division
import numpy as np

from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

# adding the PySpark modul to SparkContext
sc.addPyFile("https://raw.githubusercontent.com/seahboonsiew/pyspark-csv/master/pyspark_csv.py")
import pyspark_csv as pycsv
```

```
inspection = sc.textFile("swift://" + credentials['container'] + "." + credentials['name'] + "/Restaurant_Inspection_Scores3.csv")

def skip_header(idx, iterator):
    if (idx == 0):
        next(iterator)
    return iterator

inspection_header = inspection.first()



inspection_header_list = inspection_header.split(",")
inspection_body = inspection.mapPartitionsWithIndex(skip_header)

# filter not valid rows
inspection_body = inspection_body.filter(lambda line : len(line.split(","))>7)

# create Spark DataFrame using pyspark-csv
inspection_df = pycsv.csvToDataFrame(sqlContext, inspection_body, sep=",", columns=inspection_header_list)
inspection_df.cache()
```

```
# Python expressions in a code cell will be outputted after computation
inspection_df.printSchema()
```

```
!pip install --user seaborn
```

```
%matplotlib inline

import matplotlib.pyplot as plt
# matplotlib.patches allows us create colored patches, we can use for legends in plots
import matplotlib.patches as mpatches
# seaborn also builds on matplotlib and adds graphical features and new plot types
import seaborn as sns
import pandas as pd
```

```
inspection_pd = inspection_df[inspection_df['latitude'] != 0][['latitude', 'longitude', 'Score', 'Inspection Date']].toPandas()

inspection_pd.columns = ['latitude', 'longitude', 'Score', 'Inspection Date']
```

```
#adjust settings
sns.set_style("white")
plt.figure(figsize=(15,10))

#create scatterplots
plt.scatter(inspection_pd.longitude, inspection_pd.latitude, alpha=0.15, s=4, color='darkgreen')

#adjust more settings
plt.title('Restaurant places', size=25)
plt.xlim((-98.11,-97.69))
plt.ylim((30.12,30.57))
plt.xlabel('longitude',size=20)
plt.ylabel('latitude',size=20)

plt.show()
```

```
A= inspection_pd[np.logical_and(inspection_pd['Score']>50,inspection_pd['Score']<81)]
AA= inspection_pd[np.logical_and(inspection_pd['Score']>80,inspection_pd['Score']<91)]
AAA= inspection_pd[np.logical_and(inspection_pd['Score']>90,inspection_pd['Score']<97)]
AAAA= inspection_pd[np.logical_and(inspection_pd['Score']>96,inspection_pd['Score']<100)]
AAAAA= inspection_pd[np.logical_and(inspection_pd['Score']>99,inspection_pd['Score']<101)]

plt.figure(figsize=(15,10), dpi=0.1)

#create scatterplots
plt.scatter(A.longitude,A.latitude, s=60,alpha=0.2, color='black', marker ='.')
plt.scatter(AA.longitude,AA.latitude, s=60, alpha=0.2, color='blue', marker ='.')
plt.scatter(AAA.longitude,AAA.latitude, s=60,alpha=0.2, color='green', marker ='.')
plt.scatter(AAAA.longitude,AAAA.latitude, s=60,alpha=0.2, color='yellow', marker
='.')
plt.scatter(AAAAA.longitude,AAAAA.latitude, s=60,alpha=0.2, color='red',
marker ='.')


#create legend
black_patch = mpatches.Patch(label='A',color ='black')
blue_patch = mpatches.Patch(label='AA',color ='blue')
green_patch = mpatches.Patch(label='AAA',color ='green')
yellow_patch = mpatches.Patch(label='AAAA',color ='yellow')
red_patch = mpatches.Patch(label='AAAAA',color ='red')
plt.legend([black_patch, blue_patch, green_patch, yellow_patch, red_patch],
           ('A', 'AA', 'AAA', 'AAAA', 'AAAAA'),
           loc='upper left', prop={'size':20})

#adjust more settings
plt.title('restaurant', size=20)
plt.xlim((-98.11,-97.69))
plt.ylim((30.12,30.57))
plt.xlabel('longitude',size=20)
plt.ylabel('latitude',size=20)
plt.show()
```