kyoto restaurant

by Client 1

Submission date: 21-Apr-2020 09:53AM (UTC+0500)
Submission ID: 1303406756
File name: Kyoto_Restaurant_Analysis_-_updated.pdf (456.2K)
Word count: 6466
Character count: 34922

Kyoto Restaurant Analysis

Importing Libraries

In [363]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set() # setting seaborn default for plots
from sklearn.ensemble import *
from sklearn.svm import *
from prettytable import PrettyTable
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

Load Dataset

In [364]:
```
df =pd.read_csv('Kyoto_Restaurant_Info-1.csv')
df.head()                                    # .head() shows first 5 rows of dataset
```

Out[364]:

	Name	JapaneseName	Station	FirstCategory	SecondCategory	DinnerPrice	LunchPrice	TotalRating	DinnerRating	LunchR
0	Orudeidainingurajou	オールデイダイニング　ラジョウ	Kyoto	Buffet style	Cafe	￥4000～￥4999	￥2000～￥2999	3.39	3.20	
1	Steak Frites Gaspard zinzin	ステックフリットガスパールザンザン	Karasuma	Bistro	Steak	￥3000～￥3999	￥1000～￥1999	3.18	3.06	
2	KAZUMA	和馬	Sanjo	Izakaya (Tavern)	Japanese food (other)	￥3000～￥3999	NaN	3.28	3.28	
3	okonomiyakiteppanyakimiki	お好み焼き 鉄板焼き 三喜	Tambaguchi	Okonomiyaki	Izakaya (Tavern)	￥3000～￥3999	NaN	3.14	3.14	
4	Shaofeiyan	小肥羊 京都河原町店	Kyoto Shiyakusho Mae	Chinese hot pot / fire pot	Chinese	￥4000～￥4999	￥1000～￥1999	3.16	3.16	

- You may notice above TotalRating value is made by using both values of LunchRating and DinnerRating. So, we can drop these two columns and can work using TotalRating.

Data Preprocessing and Understanding

In [365]:
```
df.shape  # dimensions of dataset
```

Out[365]:
```
(895, 13)
```

- As, we can see There are 36 columns(features) and 141712 rows in this dataset

In [366]:
```
df.columns
```

Out[366]:
```
Index(['Name', 'JapaneseName', 'Station', 'FirstCategory', 'SecondCategory',
       'DinnerPrice', 'LunchPrice', 'TotalRating', 'DinnerRating',
       'LunchRating', 'ReviewNum', 'Lat', 'Long'],
      dtype='object')
```

- You can have a loook at all 13 features of dataset above.

In [367]:
```
df = df.applymap(lambda s:s.lower() if type(s) == str else s)  # convert all feature values to lower cas
```

In [368]:
```
df.dtypes  # check data type of each feature (column)
```

Out[368]:
```
Name              object
JapaneseName      object
Station           object
FirstCategory     object
SecondCategory    object
DinnerPrice       object
LunchPrice        object
TotalRating      float64
DinnerRating     float64
LunchRating      float64
ReviewNum          int64
Lat              float64
Long             float64
dtype: object
```

- In DinnerPrice, minimum and maximum price of dinner is given with approximation sign, lets extract these minimum and maximum values in a seperate column using which we will be able to calculate average price of a Dinner.

In [369]:
```
# removing extra information from feature values to make it in a useable form

df['mini_DinnerPrice'] = df['DinnerPrice'].apply(lambda x: str(x).split('～')[0] if x != np.nan else np.nan)
df['maxi_DinnerPrice'] = df['DinnerPrice'].apply(lambda x: str(x).split('～')[-1] if x != np.nan else np.nan)

df['mini_DinnerPrice'] = df['mini_DinnerPrice'].apply(lambda x: str(x).split('￥')[-1] if x != np.nan else np.nan)
df['maxi_DinnerPrice'] = df['maxi_DinnerPrice'].apply(lambda x: str(x).split('￥')[-1] if x != np.nan else np.nan)
```

In [370]:
```
df.head()
```

Out[370]:

	Name	JapaneseName	Station	FirstCategory	SecondCategory	DinnerPrice	LunchPrice	TotalRating	DinnerRating	LunchR
0	orudeidainingurajou	オールデイダイニング　ラジョウ	kyoto	buffet style	cafe	￥4000～￥4999	￥2000～￥2999	3.39	3.20	
1	steak frites gaspard zinzin	ステックフリット ガスパール ザンザン	karasuma	bistro	steak	￥3000～￥3999	￥1000～￥1999	3.18	3.06	
2	kazuma	和馬	sanjo	izakaya (tavern)	japanese food (other)	￥3000～￥3999	NaN	3.28	3.28	
3	okonomiyakiteppanyakimiki	お好み焼き 鉄板焼き 三喜	tambaguchi	okonomiyaki	izakaya (tavern)	￥3000～￥3999	NaN	3.14	3.14	
4	shaofeiyan	小肥羊 京都河原町店	kyoto shiyakusho mae	chinese hot pot / fire pot	chinese	￥4000～￥4999	￥1000～￥1999	3.16	3.16	

Note

- Have you noticed there are some empty strings in both maxi_DinnerPrice and mini_DinnerPrice features? if not let's see below

In [371]:
```
df[df['mini_DinnerPrice']==' ']
```

Out[371]:

	Name	JapaneseName	Station	FirstCategory	SecondCategory	DinnerPrice	LunchPrice	TotalRating	[illegible]
180	ikiikiudon	いきいきうどん 京都店	karasuma oike	udon	japanese food (other)	～￥999	～￥999	3.18	
484	resutoranrejina	レストラン レジーナ	kyoto	western food (other)	buffet style	～￥999	￥1000～￥1999	3.03	
611	zekuu	是空	imadegawa	ramen	boneless deep-fried chicken	～￥999	～￥999	3.34	
623	noukoutoripaitanramemmisora	濃厚鶏白湯ラーメン 美空	kawaramachi	ramen	tsukemen	～￥999	～￥999	3.09	
639	yebisu	担担麺 胡	keihan yamashina	dandan noodles	ramen	～￥999	NaN	3.07	
640	takayasuzekuu	高安 是空	karasuma	ramen	NaN	～￥999	～￥999	3.09	
642	ebisu	恵比朱	kyoto	ramen	chinese	～￥999	～￥999	3.51	
660	chuukasobatakayasu	中華そば 高安	ichijoji	ramen	boneless deep-fried chicken	～￥999	～￥999	3.59	
666	maribu	marib	shijo	dining bar	bar	～￥999	～￥999	3.07	
667	takoyakimihashiya	たこ焼き みはし屋	mototanaka	takoyaki	NaN	～￥999	～￥999	3.30	
701	karubidontosundoufusemmontenkandon	カルビ丼とスン豆腐専門店 韓丼 新堀川本店	fushimi	korean cuisine	ramen	～￥999	～￥999	3.33	
732	ramenhikosaku	らーめん　彦さく	imadegawa	ramen	dumplings	～￥999	～￥999	3.51	
750	marugameseimen	丸亀製麺 河原町三条店	kyoto shiyakusho mae	udon	NaN	～￥999	～￥999	3.07	
751	ramenfuji	ラーメン 藤 五条店	kiyomizu gojo	ramen	NaN	～￥999	～￥999	3.16	

	Name	JapaneseName	Station	FirstCategory	SecondCategory	DinnerPrice	LunchPrice	TotalRating	D
757	menshoutakamatsu	麺匠 たか松 四条店	karasuma	ramen	tsukemen	～￥999	～￥999	3.07	
761	kodawaritamagosemmontentamagoya	こだわり卵専門店 たまごや	arashiyama torokko station	cafe	japanese food (other)	～￥999	NaN	3.15	
770	itoukyuuemon	伊藤久右衛門 本店	uji	sweets	japan tea shoppe	～￥999	NaN	3.58	
775	hairaitoshiyokudou	ハイライト食堂 御池店	yamanouchi	table d'hôte	western cuisine	～￥999	～￥999	3.08	
786	honkedaiichiasahi	本家 第一旭 たかばし本店	kyoto	ramen	dumplings	～￥999	～￥999	3.60	
788	hisago	ひさご	gion shijo	oyako-don (chicken bowl)	soba	～￥999	～￥999	3.54	
793	ajinomeimon	味の名門 総本家	tambaguchi	ramen	dumplings	～￥999	～￥999	3.19	
847	maboroshinochuukasobakatouyashijounibojirou	幻の中華そば加藤屋 四条にぼ次朗	shijo	ramen	chinese	～￥999	～￥999	3.51	
877	karafuneyakohiten	からふね屋珈琲店 三条本店	kyoto shiyakusho mae	traditional café	cafe	～￥999	～￥999	3.52	
888	menshoutakamatsu	麺匠 たか松 本店	karasuma	tsukemen	ramen	～￥999	～￥999	3.58	

In [372]:
```
df[df['maxi_DinnerPrice']=='']
```

Out[372]:

	Name	JapaneseName	Station	FirstCategory	SecondCategory	DinnerPrice	LunchPrice	TotalRating	DinnerRating	LunchRating
804	shourenkanyoshinoya	昭恋館 よ志のや	kyotango	ryokan	crab	￥30000～	￥10000～￥14999	3.55	3.56	3.51

- if you have observe in a dataset most of the min and max values of DinnerPrice feature are close to each other. Like if minimum value is 2000

max value will be 2999. You can see below. Now let's try to fill the empty values using this technique.

In [373]:
```
df.head()
```

Out[373]:

	Name	JapaneseName	Station	FirstCategory	SecondCategory	DinnerPrice	LunchPrice	TotalRating	DinnerRating	LunchR
0	orudeidainingurajou	オールデイダイニング　ラジョウ	kyoto	buffet style	cafe	￥4000～￥4999	￥2000～￥2999	3.39	3.20	
1	steak frites gaspard zinzin	ステックフリット ガスパール ザンザン	karasuma	bistro	steak	￥3000～￥3999	￥1000～￥1999	3.18	3.06	
2	kazuma	和馬	sanjo	izakaya (tavern)	japanese food (other)	￥3000～￥3999	NaN	3.28	3.28	
3	okonomiyakiteppanyakimiki	お好み焼き 鉄板焼き 三喜	tambaguchi	okonomiyaki	izakaya (tavern)	￥3000～￥3999	NaN	3.14	3.14	
4	shaofeiyan	小肥羊 京都河原町店	kyoto shiyakusho mae	chinese hot pot / fire pot	chinese	￥4000～￥4999	￥1000～￥1999	3.16	3.16	

- In mini_DinnerPrice feature maximum value is **999** so lets fill the minimum value wirh **800**.
- In maxi_DinnerPrice feature minimum value is **30000** so lets fill the maximum value wirh **39999**.

In [374]:
```
df['mini_DinnerPrice']=df.mini_DinnerPrice.replace(" ", "800")
df['maxi_DinnerPrice']=df.maxi_DinnerPrice.replace("", "39999")
```

- Now, let's convert datatype of both features to numeric so that we will be able to calcualte average.

In [375]:
```
df['mini_DinnerPrice']= pd.to_numeric(df.mini_DinnerPrice)
df['maxi_DinnerPrice']= pd.to_numeric(df.maxi_DinnerPrice)
```

In [376]:
```
avg = (df['mini_DinnerPrice'] + df['maxi_DinnerPrice'])/2          # calculation of average
df['avgDinnerPrice']=np.ceil(avg)                                  # apply ceiling function on avg
df['avgDinnerPrice']=df['avgDinnerPrice'].astype(int)              # convert avg from float to int
```

Now we have average price of Dinner of each resturant

In [377]:
```
df.head()
```

Out[377]:

	Name	JapaneseName	Station	FirstCategory	SecondCategory	DinnerPrice	LunchPrice	TotalRating	DinnerRating	LunchR
0	orudeidainingurajou	オールデイダイニング　ラジョウ	kyoto	buffet style	cafe	￥4000～￥4999	￥2000～￥2999	3.39	3.20	
1	steak frites gaspard zinzin	ステックフリット ガスパール ザンザン	karasuma	bistro	steak	￥3000～￥3999	￥1000～￥1999	3.18	3.06	
2	kazuma	和馬	sanjo	izakaya (tavern)	japanese food (other)	￥3000～￥3999	NaN	3.28	3.28	
3	okonomiyakiteppanyakimiki	お好み焼き 鉄板焼き 三喜	tambaguchi	okonomiyaki	izakaya (tavern)	￥3000～￥3999	NaN	3.14	3.14	
4	shaofeiyan	小肥羊 京都河原町店	kyoto shiyakusho mae	chinese hot pot / fire pot	chinese	￥4000～￥4999	￥1000～￥1999	3.16	3.16	

In [378]:
```
df.describe() # description all numeric features(columns) in datse
```

Out[378]:

	TotalRating	DinnerRating	LunchRating	ReviewNum	Lat	Long	mini_DinnerPrice	maxi_DinnerPrice	avgDinnerPrice
count	895.000000	895.000000	493.000000	895.000000	895.000000	895.000000	895.000000	895.000000	895.000000
mean	3.212480	3.179218	3.213570	27.080447	35.005287	135.755774	3963.351955	5323.022346	4643.687151
std	0.215588	0.199673	0.213926	58.064708	0.062367	0.056629	2832.094336	3996.367691	3406.314349
min	3.000000	3.000000	3.000000	1.000000	34.715818	135.094284	800.000000	999.000000	900.000000
25%	3.050000	3.040000	3.050000	7.000000	34.995639	135.757501	3000.000000	3999.000000	3500.000000
50%	3.090000	3.070000	3.090000	15.000000	35.004454	135.762578	3000.000000	3999.000000	3500.000000
75%	3.420000	3.300000	3.390000	30.000000	35.008488	135.770655	4000.000000	4999.000000	4500.000000
max	4.190000	4.160000	4.110000	1319.000000	35.735303	135.831994	30000.000000	39999.000000	35000.000000

Insights:

As, you can observe

- count of each attribute showing above is 895 which is size of total rows in data there is no missing value in these features except LunchRating.
- minimum rating given to any resturant is 3 and minimum votes given to any resturant is 1 which means there are some resturants in the dataset which only get 1 vote(ReviewNum)
- maximum rating given to any resturant is 4.19 and maximum votes given to any resturant is 1319 which means there are some resturants in the dataset which have recieved 1319 votes(ReviewNum)
- 50 % of the resturants are those which only recieved reviewS by 15 people
- And 75 % of the resturants are those which only recieved reviewS by 30 people.

In [379]:
```
df.describe(include='object') # description all non-numeric features(columns)
```

Out[379]:

	Name	JapaneseName	Station	FirstCategory	SecondCategory	DinnerPrice	LunchPrice
count	895	895	895	895	867	895	484
unique	854	895	80	97	105	12	10
top	yakinikuyaruki	ピアキッチン肉バル ジカビヤ 京都河原町店	gion shijo	izakaya (tavern)	izakaya (tavern)	￥3000～￥3999	￥1000～￥1999
freq	6	1	112	228	77	315	171

Insights:

As, you can observe

- count: of all except 2 attributes is 895, means there is no missing value in that feature, whereas in **SecondCategory** count is 867 and LunchPrice count is 484 which means it contain empty(null) values.
- Unique values : shows each feature(column) unique values, like incident **Name** contains unique values **858** among **895** which means there are some resturant Names which are recorded multiple times in dataset.
- top : shows top values of each feature.
- freq : shows that how many times top value occured. Like in incident resturant Name **yakinikuyaruki** recorded **5** times in dataset and **station Gion Shijo** comes 112 times.

Missing value

In [380]:
```
# check missing values in dataset in each feature
df.isnull().sum().sort_values(ascending = False)
```

Out[380]:
```
LunchPrice          411
LunchRating         402
SecondCategory       28
avgDinnerPrice        0
maxi_DinnerPrice      0
mini_DinnerPrice      0
Long                  0
Lat                   0
ReviewNum             0
DinnerRating          0
TotalRating           0
DinnerPrice           0
FirstCategory         0
Station               0
JapaneseName          0
Name                  0
dtype: int64
```

Note

- As you can abserve almost half of the values of LunchPrice and LunchRating are unknown which is a large number so its better to drop these columns.
- Whereas, in SecondCategory only 28 values are unknown we will try to fill these values with the most suitable values below.

Handling Missing Values

There are many techniques using which we can fill misssing values. Let's fill missing value of SecondCategory with the most frquent value of the column, which is Izakaya (Tavern) which you can see below :

In [381]:
```
df['SecondCategory'].value_counts().head()
```

Out[381]:
```
izakaya (tavern)                77
seafood                         67
kyoto cuisine                   53
kaiseki (traditional japanese)  47
bar                             38
Name: SecondCategory, dtype: int64
```

In [382]:
```
df['SecondCategory']=df['SecondCategory'].fillna('Izakaya (Tavern)')
```

- Now let's drop those columns which have large number of missing values.

In [383]:
```
df.drop(['LunchPrice','LunchRating'], axis=1,inplace=True)
```

In [384]:
```
df.isnull().sum()
```

Out[384]:
```
Name              0
JapaneseName      0
Station           0
FirstCategory     0
SecondCategory    0
DinnerPrice       0
TotalRating       0
DinnerRating      0
ReviewNum         0
Lat               0
Long              0
mini_DinnerPrice  0
maxi_DinnerPrice  0
avgDinnerPrice    0
dtype: int64
```

- Now you can see there is no null value in a dataset

Resturant Name

In [385]:
```
len(df.Name.unique()) # Let's see resturant Names having unique values
```

Out[385]:
```
854
```

In [386]:
```
# total count of each Resturant Names
df.Name.value_counts()
```

Out[386]:
```
yakinikuyaruki                  6
nikubaruginjirou                4
ryuumon                         3
momojirou                       3
sumibikushiyakitsujiya          3
kaitoshirowainnobarukakimaru    3
kaiuntei                        3
kyuushuunecchuuya               3
sandaimetorimero                2
komefuku                        2
kyoutokimurayahonten            2
aburiya                         2
tenkanoyakinikudaishougun       2
gokyou                          2
suiba                           2
figerasu                        2
teppandainingugionshou          2
kankokusakuratei                2
ginzaraion                      2
ganko                           2
menshoutakamatsu                2
sute-kihausupondo               2
kushiyakimanten                 2
bejitejiya                      2
nikugaumaikafenikkusutokku      2
ichibakouji                     2
kampaiboupuruda                 2
jukuseiyakinikupondo            2
miho                            2
yakinikuyoshimi                 1
                               ..
koudaijihashiba                 1
shinshin                        1
itariamba-rukimuraya            1
fortune garden kyoto            1
in the soup.                    1
toriseetakoyakushiten           1
arashiyamamitate                1
```

```
kafeandopanke-kiguramu                  1
kyoukarasuma                            1
kyouzakanafukuya                        1
wainshokudouumiusagi                    1
the aglio garden                        1
nikudokoroikkou                         1
kabochanotane                           1
kirarazaka                              1
shunsaidaininguitokoya                  1
sumibirobataikoru                       1
kyoutosakagurakan                       1
waraiya                                 1
itariambarutoreotto                     1
aburaya                                 1
kyoutonanajouyakinikusakabayamadaruma   1
ajidokoroterasaki                       1
feliz-lana                              1
watanabe                                1
arufakafe                               1
miyakohanten                            1
kyoutohanabi                            1
gounotora                               1
karasumarugenkiichiba                   1
Name: Name, Length: 854, dtype: int64
```

- You can observe above there are some values in **Name** which occured more than 1 time.

In [387]:
```
df[df.Name=='yakinikuyaruki  '].head() # verify by looking at dataset
```

Out[387]:

	Name	JapaneseName	Station	FirstCategory	SecondCategory	DinnerPrice	TotalRating	DinnerRating	ReviewNum	Lat	Lo
415	yakinikuyaruki	焼肉やる気 西大路五条店	saiin	yakiniku (bbq beef)	izakaya (tavern)	￥2000～￥2999	3.00	3.00	30	34.997751	135.731
416	yakinikuyaruki	焼肉やる気 河原町店	sanjo	yakiniku (bbq beef)	izakaya (tavern)	￥2000～￥2999	3.04	3.04	19	35.007946	135.769
419	yakinikuyaruki	焼肉やる気 新堀川本店	fushimi	yakiniku (bbq beef)	izakaya (tavern)	￥2000～￥2999	3.02	3.02	21	34.944461	135.752
423	yakinikuyaruki	焼肉やる気 山科店	nagitsuji	yakiniku (bbq beef)	korean cuisine	￥2000～￥2999	3.01	3.01	24	34.971603	135.813
506	yakinikuyaruki	焼肉やる気 四条河原町店	gion shijo	yakiniku (bbq beef)	steak	￥2000～￥2999	3.07	3.08	29	35.003673	135.770

- All the restuants Name are same but have different Station values which means a resturant may have different branches in different locations.

Top Resturants using Rating Analysis

- As we have seen above TotalRating is made by using both DinnerRating and LunchRating, so here I will only use TotalRating here.

In [388]:
```
print(df[['Name','ReviewNum','TotalRating']])
```

```
                                  Name  ReviewNum  TotalRating
0                    orudeidainingurajou         56         3.39
1             steak frites gaspard zinzin         70         3.18
2                                 kazuma          7         3.28
3               okonomiyakiteppanyakimiki         16         3.14
4                              shaofeiyan         23         3.16
5                                okuta-va         24         3.08
6                  resort dining&bar hale         22         3.04
7                                 sumika         23         3.34
8                               gayagaya          3         3.03
9                     ishibekoujimamecha         92         3.56
10  nagoyako-chintokoshitsuizakayatorisai          4         3.04
11                              gionabesu        120         3.58
12          nishikimachinoakarishoutengai          2         3.00
13                            gionni-yongo         39         3.85
14                                 mumon         13         3.05
15               itariandainingutoreotto          2         3.04
16            sukiyakisunibiizakayahokuto         19         3.09
17                              sakaean         31         3.09
```

- Review number shows how many reviwers give reviews or rating to any resturant.
- We can observe there is a resturant which is rated by 1319 reviewers however, some resturants only got 1 rating(review) by only 1 reviewer.
- If we want to see popular resturants on the basis of rating, this rating is not reliable here because it does not take into consideration the popularity of a resturant. A resturant with a rating of `4.00` from `10` reviwers will be considered 'better' than a resturant with a rating of `3.00` from `100` reviwers.

Solution

- To solve this problem I am going to use weighted rated formula which you can see below.

Weighted rating formula calculation:

- The metric is the numeric quantity based on which we rank resturants. Using this metric a resturant is considered to be better than another resturant if it has a higher metric score than the other resturant.

- It is also a well-known fact that as the number of reviwers increase, the rating of a resturant normalizes and it approaches a value that is reflective of the resturant's quality and popularity with the general populace. A resturant with very few reviwers are not very reliable. A Resturant rated 10/10 by five users doesn't necessarily mean that it's a good resturant.

Weighted Rating (WR) = (v/v+m.R)+(m/v+m.C)

where,

- **v** is the number of reviewers of the resturant;
- **m** is the minimum votes required to be listed;
- **R** is the TotalRating of the resturant; And
- **C** is the mean vote across the whole dataset

- In the dataset we already have the values of R(TotalRating) and v(ReviewNum) of every resturant. It's possible to calculate value C directly using this dataset.

In [389]:
```
# Calculate C
C = df['TotalRating'].mean()
C
```

Out[389]:
```
3.212480446927375
```

- Here what is need to do is, to determine the appropriate value of minimum votes m which are needed to be listed. There is not any fix value of m. In this way you can filter out those resturants whose reviewers are less than the minimum required reviwers. The selection of this filter is up to you.
- As a cutoff in this case I am going to use 90th Percentile which means a resturant must have reviwers more than the 90 percent of the resturants which are in list. it means you need to consider those resturants which are top 10% which have gained votes.

In [390]:
```
# Calculate the minimum number of reviwers required to be listed, m
m = df['ReviewNum'].quantile(0.90)
print(m)
```

```
56.0
```

- It means only 10% of the resturants have gained 56 reviews. So, our value of m will be 56.

In [391]:
```
# Filter out all qualified resturants into a new DataFrame
popular_Resturants = df.copy().loc[df['ReviewNum'] >= m]
popular_Resturants.shape
```

Out[391]: (93, 14)

Calculate the score for every resturant:

In [392]:
```
# Function that computes the weighted rating of each resturant
def weighted_rating(x, m=m, C=C):
    v = x['ReviewNum']
    R = x['TotalRating']
    # Calculation based on the IMDB formula
    return (v/(v+m)   * R) + (m/(m+v) * C)
```

- There is just one step left. We now need to sort our DataFrame on the basis of the score we just computed and output the list of top resturants:

In [393]:
```
# Define a new feature 'score' and calculate its value with `weighted_rating()`
popular_Resturants['score'] = popular_Resturants.apply(weighted_rating, axis=1)
```

In [394]:
```
#Sort resturants based on score calculated above
popular_Resturants = popular_Resturants.sort_values('score', ascending=False)
popular_Resturants.head()
```

Out[394]:

	Name	JapaneseName	Station	FirstCategory	SecondCategory	DinnerPrice	TotalRating	DinnerRating	ReviewNum
787	kichisen	京懐石 吉泉	demachiyanagi	kaiseki (traditional japanese)	Izakaya (Tavern)	￥20000～￥29999	4.19	4.16	7
504	orudohonkonresutorankyouto	老香港酒家京都	shijo	chinese	cantonese cuisine	￥10000～￥14999	3.88	4.04	14
67	kyoutogiontempurayasakaendou	京都祇園 天ぷら八坂圓堂	gion shijo	tempura	kaiseki (traditional japanese)	￥10000～￥14999	3.68	3.64	25
786	honkedaiichiasahi	本家 第一旭 たかばし本店	kyoto	ramen	dumplings	～￥999	3.60	3.59	131
657	nihonryourisakuragawa	日本料理 櫻川	kyoto shiyakusho mae	kaiseki (traditional japanese)	kappo (traditional japanese)	￥20000～￥29999	3.89	3.71	6

Output Top Resturants

In [395]:
```
popular_Resturants[['Name', 'TotalRating', 'ReviewNum', 'score']].head(10)
```

Out[395]:

	Name	TotalRating	ReviewNum	score
787	kichisen	4.19	79	3.784510
504	orudohonkonresutorankyouto	3.88	148	3.696759
67	kyoutogiontempurayasakaendou	3.68	252	3.594996
786	honkedaiichiasahi	3.60	1319	3.584217
657	nihonryourisakuragawa	3.89	61	3.565717
660	chuukasobatakayasu	3.59	713	3.562508
38	zasoudouhigashiyamakyouto	3.61	272	3.542131
770	itoukyuuemon	3.58	380	3.532796
638	ristorante t.v.b	3.67	121	3.525248
888	menshoutakamatsu	3.58	203	3.500536

We can see that `Kichisen` resturtant figures at the top of the list. We can also see that it has a noticeably smaller number of votes than the some other Top resturants in the list. This strongly suggests that we should probably explore a higher value of **m**. But our dataset size is small here, you can experiment with different **m** values in larger dataset to get better results.

Popular Areas of Resturants

In [396]:
```
y=df.Station.value_counts().head(10)
y
```

Out[396]:
```
gion shijo              112
kyoto                   111
karasuma                 79
kawaramachi              73
sanjo                    59
shijo                    54
kyoto shiyakusho mae     54
karasuma oike            43
marutamachi              23
saiin                    18
Name: Station, dtype: int64
```

- We can observe above that there are 112 resturants near 'Gion Shijo' station you can visualize it below for better understanding.

In [397]:
```
ax = y.plot(kind='barh', figsize=(10,7),  color="pink", fontsize=13);
ax.set_alpha(0.8)
ax.set_title("Top 10 areas", fontsize=18)
ax.set_xticks([20,40,60,80,100,120])

# create a list to collect the plt.patches data
totals = []

# find the values and append to list
for i in ax.patches:
    totals.append(i.get_width())

# set individual bar lables using above list
total = sum(totals)

# set individual bar lables using above list
for i in ax.patches:
    # get_width pulls left or right; get_y pushes up or down
    ax.text(i.get_width()+.3, i.get_y()+.38, \
            str(round((i.get_width()/total)*100, 2))+'%', fontsize=10, color='dimgrey')

# invert for largest on top
ax.invert_yaxis()
```

Top 10 areas

Top Food Category

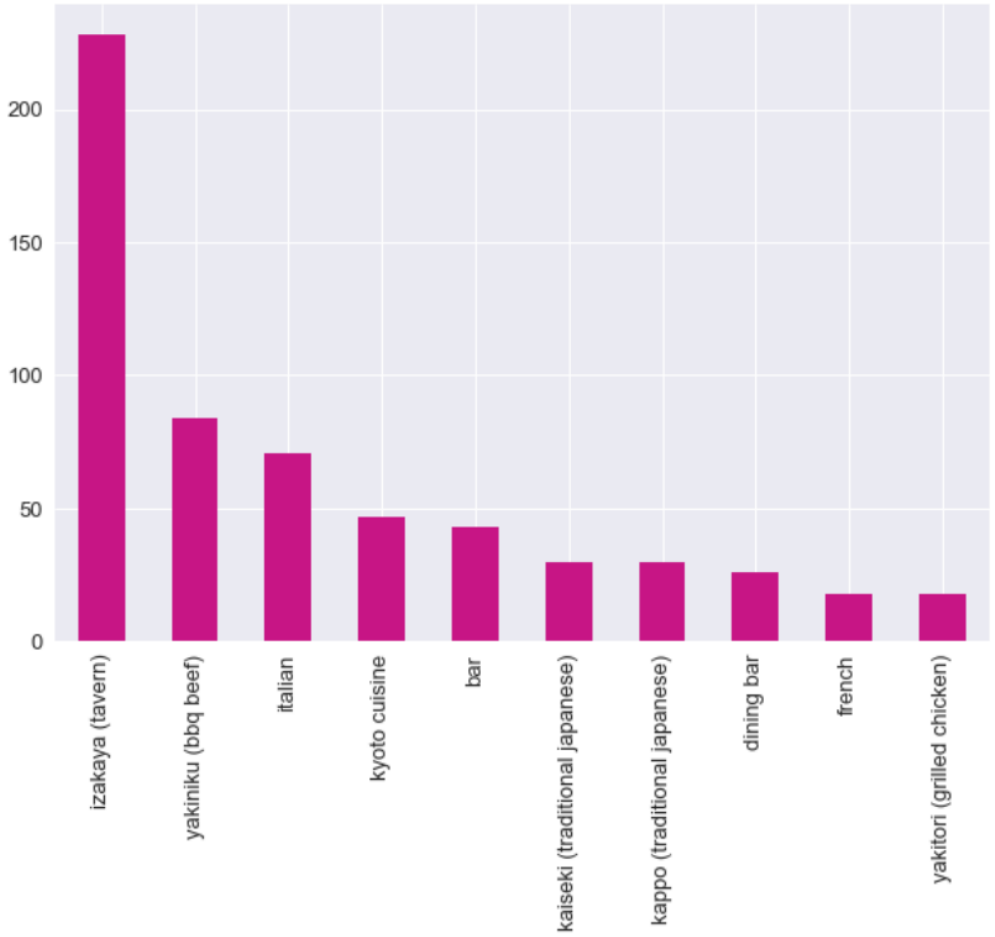
FirstCategory

In [398]:
```
FirstCategory=df.FirstCategory.value_counts().head(10)
FirstCategory
```

Out[398]:
```
izakaya (tavern)                228
yakiniku (bbq beef)              84
italian                          71
kyoto cuisine                    47
bar                              43
kaiseki (traditional japanese)   30
kappo (traditional japanese)     30
dining bar                       26
french                           18
yakitori (grilled chicken)       18
Name: FirstCategory, dtype: int64
```

- As you can see above `Izakaya (Tavern)` is the top FirstCategory served in 228 resturants and `Yakiniku (BBQ Beef)` served in 84 resturants. You can visualize it below:

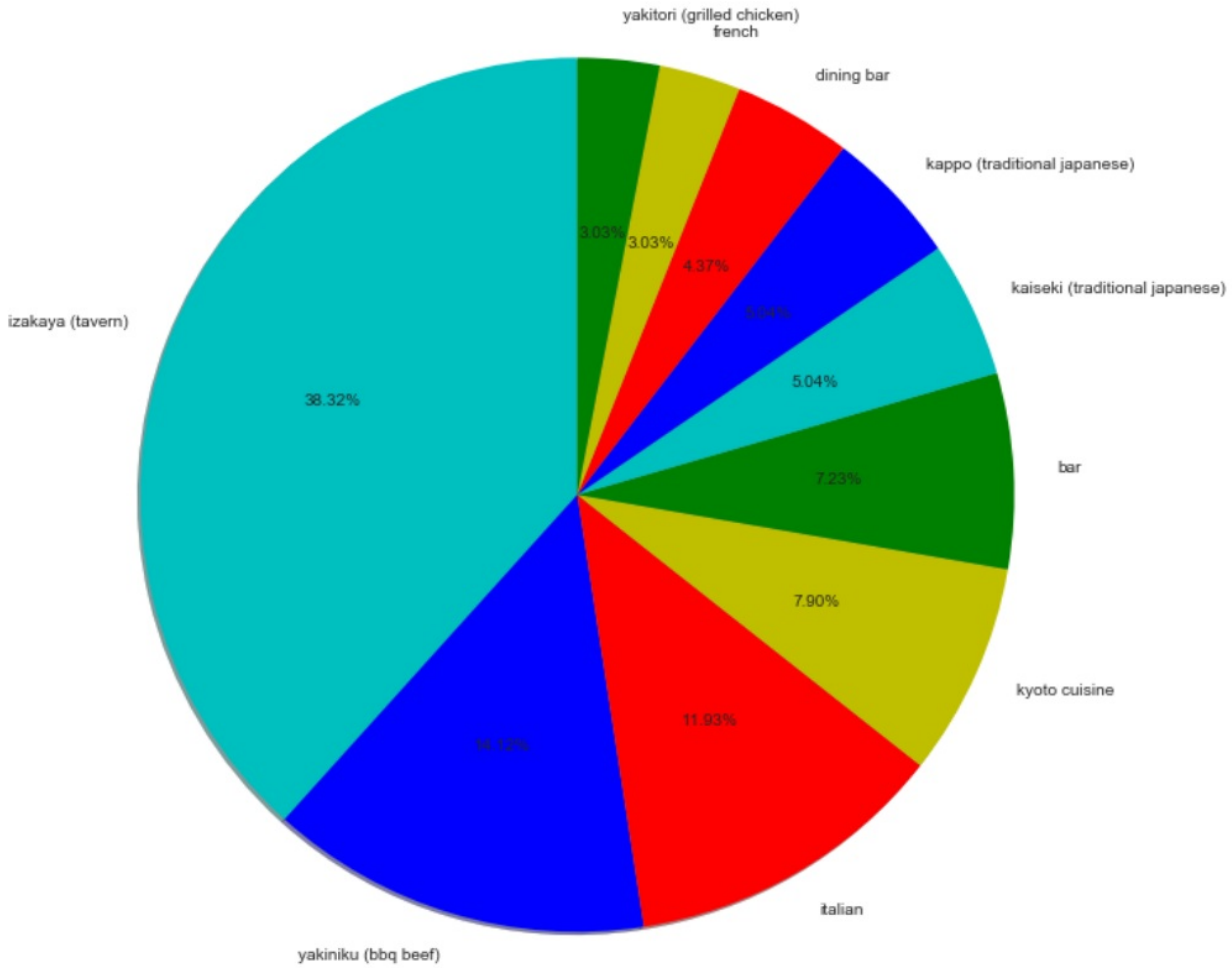
In [399]:
```
ax = FirstCategory.plot(kind='bar', figsize=(10,7),  color='mediumvioletred' , fontsize=13);
```



- For better understanding you can have a look at pie chart below :

In [400]:
```
colors = ["c", 'b','r','y','g'] # Color of each section
textprops = {"fontsize":10} # Font size of text in pie chart

FirstCategory.plot.pie( # Values
        colors =colors, # Color of each section
        autopct = "%0.2f%%", # Show data in persentage for with 2 decimal point
        shadow = True, # Showing shadow of pie chart
        radius = 3, # Radius to increase or decrease the size of pie chart
        startangle = 90, # Start angle of first section
        textprops =textprops)
plt.gca().set_aspect('equal')
plt.show() # To show pie chart only
```



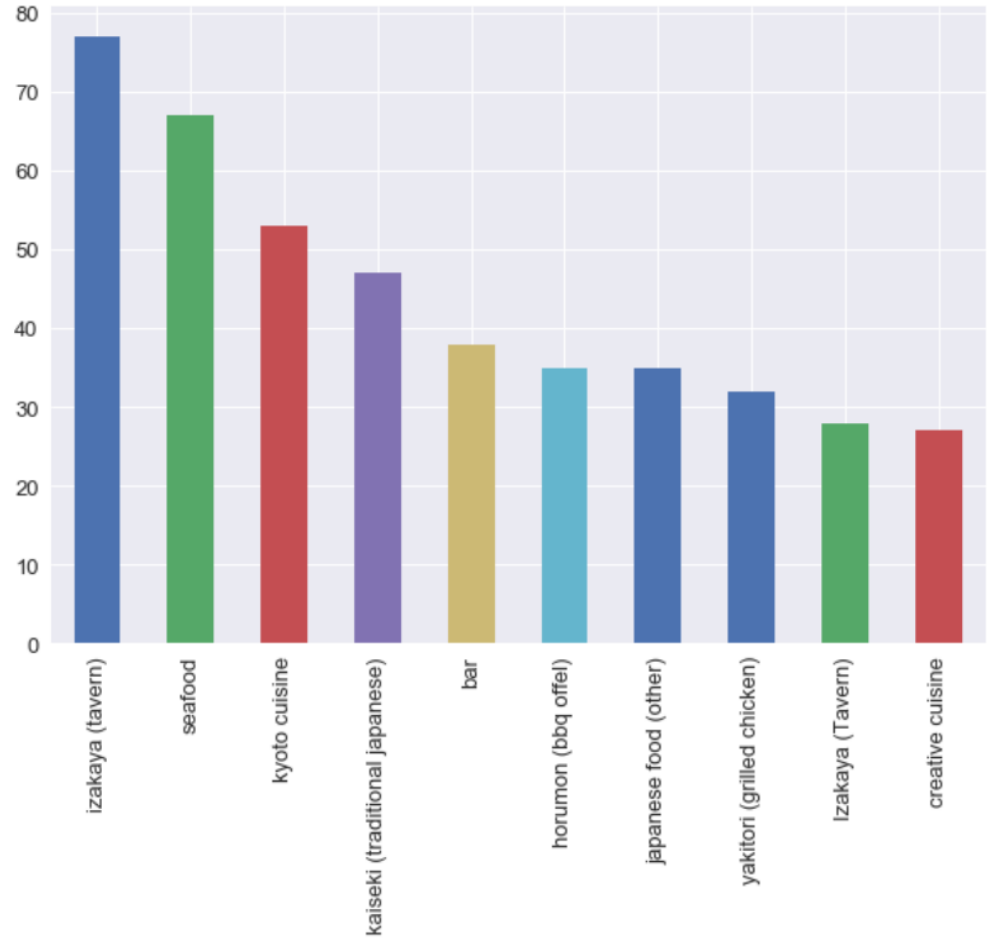
SecondCategory

In [401]:
```
SecondCategory=df.SecondCategory.value_counts().head(10)
SecondCategory
```

Out[401]:
```
izakaya (tavern)                77
seafood                         67
kyoto cuisine                   53
kaiseki (traditional japanese)  47
bar                             38
horumon (bbq offel)             35
japanese food (other)           35
yakitori (grilled chicken)      32
Izakaya (Tavern)                28
creative cuisine                27
Name: SecondCategory, dtype: int64
```

- We can see that 105 resturants have `Izakaya (Tavern)` as its SecondCategory and so on

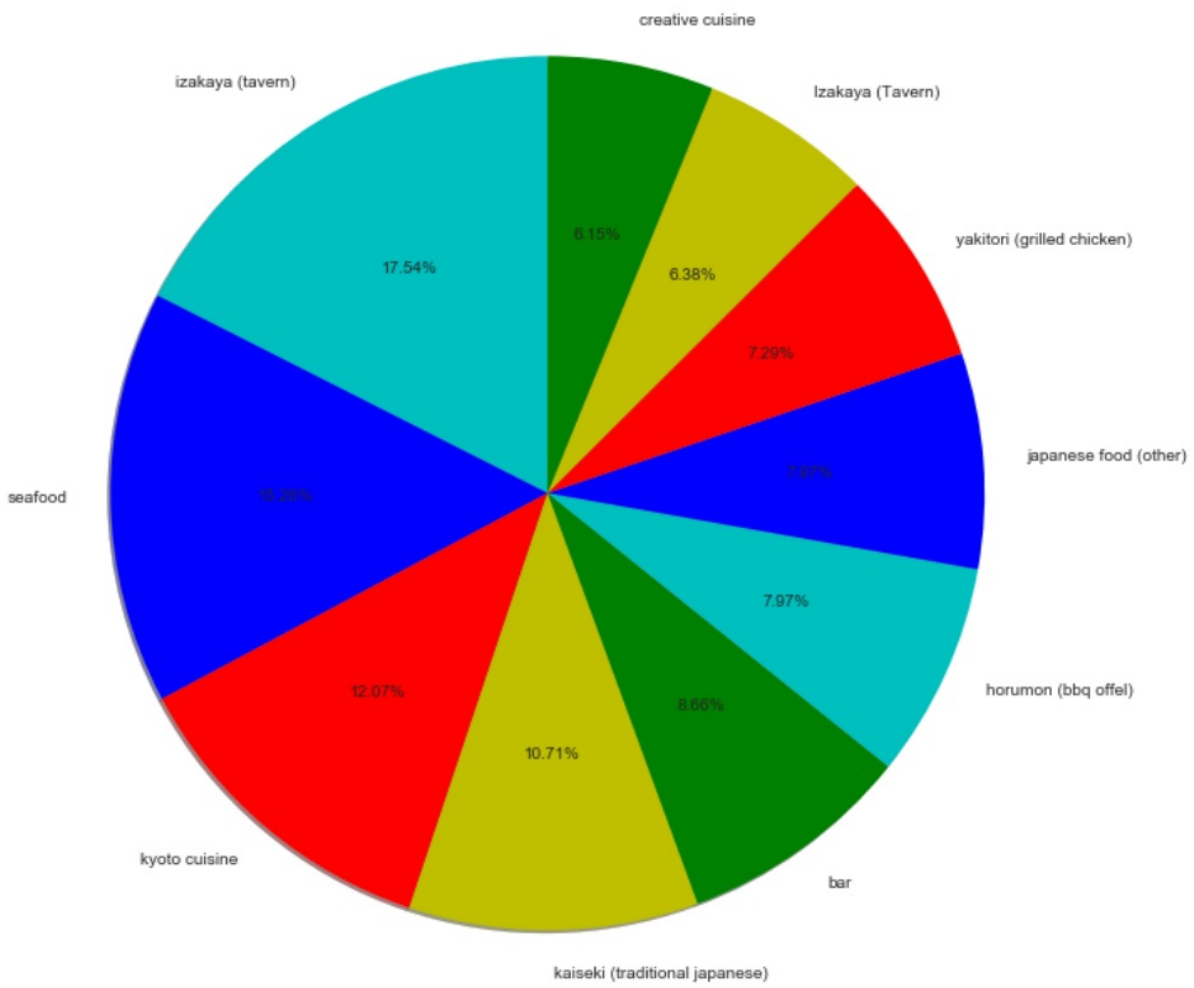
In [402]:
```
ax = SecondCategory.plot(kind='bar', figsize=(10,7), fontsize=13);
```



- For better understanding you can have a look at pie chart below :

In [403]:
```
colors = ["c", 'b','r','y','g'] # Color of each section
textprops = {"fontsize":10} # Font size of text in pie chart

SecondCategory.plot.pie( # Values
        colors =colors, # Color of each section
        autopct = "%0.2f%%", # Show data in persentage for with 2 decimal point
        shadow = True, # Showing shadow of pie chart
        radius = 3, # Radius to increase or decrease the size of pie chart
        startangle = 90, # Start angle of first section
        textprops =textprops)
plt.gca().set_aspect('equal')
plt.show() # To show pie chart only
```



Relation between FirstCategory and SecondCategory

In [404]:
```
df.groupby('FirstCategory').SecondCategory.value_counts()
```

Out[404]:

FirstCategory	SecondCategory	
bar	italian	7
	izakaya (tavern)	5
	steak	5
	wine bar	4
	bar	2
	dining bar	2
	lounge	2
	nihonshu (japanese sake)	2
	pub	2
	spain	2
	Izakaya (Tavern)	1
	bistro	1
	cafe	1
	chinese	1
	french	1
	motsu nabe (offel hot pot)	1
	seafood	1
	sumibiyaki	1
	western (others)	1
	yakiniku (bbq beef)	1
beef dishes	french	1
	wine bar	1
beer	beer restaurant	1
beer bar	bar	1
beer garden	bbq	1
	beer	1
bistro	bar	3
	seafood	3
	italian	2
	steak	2
		..
western (others)	french	1
western cuisine	french	2
	buffet style	1
western food (other)	buffet style	1
wine bar	bar	1
	italian	1

```
                           izakaya (tavern)                    1
yakiniku (bbq beef)        horumon (bbq offel)                33
                           korean cuisine                     16
                           izakaya (tavern)                   14
                           Izakaya (Tavern)                    3
                           bar                                 3
                           japanese food (other)               2
                           shabu shabu (japanese steamboat)    2
                           steak                               2
                           sukiyaki                            2
                           chiritori nabe (tripe)              1
                           creative cuisine                    1
                           gyutan (beef tongue)                1
                           kaiseki (traditional japanese)      1
                           kyoto cuisine                       1
                           meat dishes                         1
                           seafood                             1
yakitori (grilled chicken) izakaya (tavern)                    9
                           fowl                                3
                           kushi-age (fried skewer)            2
                           kyoto cuisine                       1
                           motsu nabe (offel hot pot)          1
                           shochu (japanese spirits)           1
                           sukiyaki                            1
Name: SecondCategory, Length: 380, dtype: int64
```

Rating vs. Average Price

In [405]:
```
df.groupby('DinnerRating').avgDinnerPrice.value_counts()
```

Out[405]:

DinnerRating	avgDinnerPrice	
3.00	3500	35
	2500	21
	4500	13
	5500	6
	1500	4
	7000	2
	900	1
	9000	1
3.01	2500	5
	3500	3
	5500	1
	7000	1
3.02	3500	13
	2500	7
	1500	2
	4500	2
	5500	2
	7000	1
	9000	1
3.03	3500	22
	4500	8
	2500	7
	1500	2
	5500	2
	900	1
	7000	1
	9000	1
3.04	3500	37
	2500	13
	4500	13
		..
3.58	4500	1
	7000	1
	9000	1
	25000	1
3.59	12500	2
	900	1

```
                9000              1
                17500             1
3.60            1500              1
                5500              1
                9000              1
3.61            12500             2
                7000              1
3.62            9000              1
3.63            12500             1
3.64            12500             1
                17500             1
3.65            12500             1
3.66            9000              1
3.68            25000             1
3.69            9000              1
3.71            25000             1
3.72            12500             1
3.74            5500              1
3.80            25000             1
3.83            12500             1
3.91            12500             1
3.97            9000              1
4.04            12500             1
4.16            25000             1
Name: avgDinnerPrice, Length: 299, dtype: int64
```

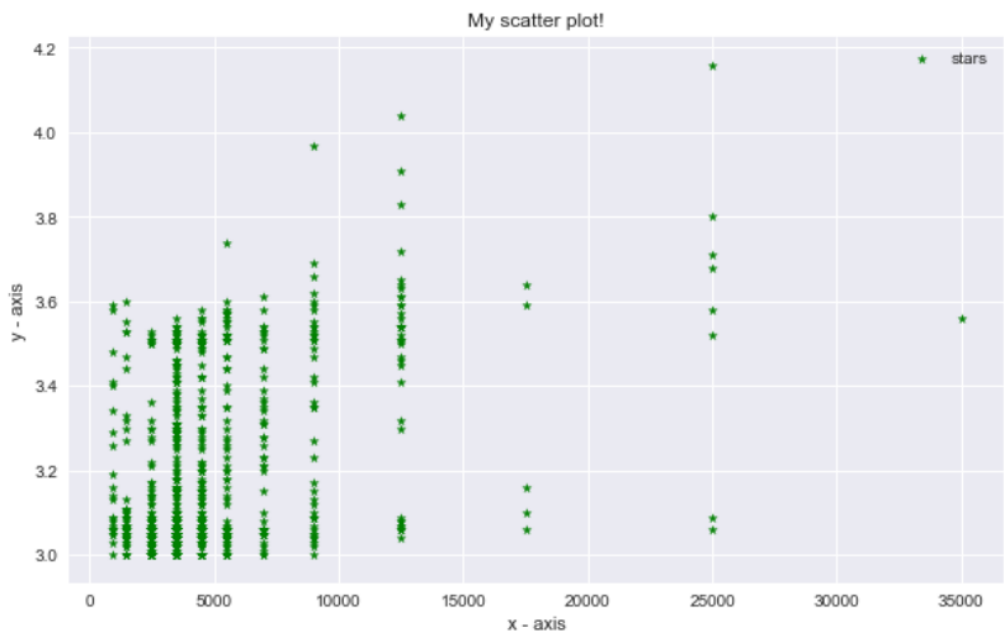
Insights:

- You can observe if you pay more you can get higher quality of food. You can visuilze it below:

In [406]:
```
import matplotlib.pyplot as plt
# x-axis values
x = df['avgDinnerPrice']
# y-axis values
y = df['DinnerRating']
fig, ax = plt.subplots(figsize=(10, 6))
# plotting points as a scatter plot
ax.scatter(x, y, label= "stars", color= "green", marker= "*", s=30)

# x-axis label
plt.xlabel('x - axis')
# frequency label
plt.ylabel('y - axis')
# plot title
plt.title('My scatter plot!')
# showing legend
plt.legend()

# function to show the plot
plt.show()
```



- Top 10 FirstCategory of resturants are given above.

List of Resturant in Each Station

- It is Treemap charts, visualize hierarchical data using nested rectangles. Click on one sector to zoom in/out, which also displays a pathbar in the upper-left corner of your treemap. To zoom out you can use the path bar as well. You can save the chart by clicking on camera icon showing in upper-left corner.

In [407]:
```
import plotly.express as px

fig = px.treemap(df, path=['Station','Name'], values='avgDinnerPrice', hover_data=['Station'],color='Station')
fig.show()
```

Insights:

- Area names are based on restaurants' closest stations. All resturants available in a each station are given above you can select any station to see which resturant is present in it. Size of boxes are bigger according to average price of a resturant.
- Like if you will click `Gion Shijo` you can see list of all resturants in it you can verify it below:

In [408]:
```
df[df['Station']=='gion shijo']
```

Out[408]:

	Name	JapaneseName	Station	FirstCategory	SecondCategory	DinnerPrice	TotalRating	DinnerRating	ReviewNum	
9	ishibekoujimamecha	石塀小路 豆ちゃ 京都	gion shijo	izakaya (tavern)	kyoto cuisine	￥4000～￥4999	3.56	3.56	92	35
11	gionabesu	祇園 abbesses	gion shijo	french	italian	￥8000～￥9999	3.58	3.54	120	35
32	teppandainingukyoushikian	鉄板ダイニング 京四季庵	gion shijo	teppanyaki	steak	￥5000～￥5999	3.06	3.06	16	35
34	in the soup.	in the soup.	gion shijo	bar	bistro	￥3000～￥3999	3.02	3.03	5	35
35	gionyakinikukokoro	祇園焼肉 志	gion shijo	yakiniku (bbq beef)	horumon (bbq offel)	￥5000～￥5999	3.06	3.06	5	35
36	daininguba-daburyu-	dining bar w	gion shijo	kappo (traditional japanese)	steak	￥4000～￥4999	3.25	3.25	17	35

Prediction of Rating

Feature Selection

- Following are the available features present in dataset.

In [409]:
```
df.columns
```

Out[409]:
```
Index(['Name', 'JapaneseName', 'Station', 'FirstCategory', 'SecondCategory',
       'DinnerPrice', 'TotalRating', 'DinnerRating', 'ReviewNum', 'Lat',
       'Long', 'mini_DinnerPrice', 'maxi_DinnerPrice', 'avgDinnerPrice'],
      dtype='object')
```

In [410]:
```
selected_df= df.copy()
```

I drop unnecessary columns/features and keep only the useful ones for prediction task.

In [411]:
```
features_drop = [ 'Name','JapaneseName',  'FirstCategory', 'SecondCategory',
       'DinnerPrice','DinnerRating', 'ReviewNum', 'Lat',
       'Long', 'mini_DinnerPrice', 'maxi_DinnerPrice']

selected_df = selected_df.drop(features_drop, axis=1)
```

Covert all features into numeric

- For applying machine learning algorithms all feature values must be in numeric form, So now lets encode all the `categorical` features into numeric using Label encoding.

In [412]:
```
selected_df.head()                # Dataset Before Encoding
```

Out[412]:

	Station	TotalRating	avgDinnerPrice
0	kyoto	3.39	4500
1	karasuma	3.18	3500
2	sanjo	3.28	3500
3	tambaguchi	3.14	3500
4	kyoto shiyakusho mae	3.16	4500

In [413]:
```
selected_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 895 entries, 0 to 894
Data columns (total 3 columns):
Station           895 non-null object
TotalRating       895 non-null float64
avgDinnerPrice    895 non-null int32
dtypes: float64(1), int32(1), object(1)
memory usage: 17.6+ KB
```

Label Encoding

In [414]:
```
labelEncoder_Station = LabelEncoder()
labelEncoder_Station.fit(selected_df.Station)
selected_df['Station']=labelEncoder_Station.transform(selected_df.Station)
```

In [415]:
```
selected_df.head()                    # Dataset After Encoding
```

Out[415]:

	Station	TotalRating	avgDinnerPrice
0	41	3.39	4500
1	24	3.18	3500
2	66	3.28	3500
3	74	3.14	3500
4	42	3.16	4500

- We are done with Label Encoding, now all of our features are in numeric. Each `Catagorical` value is encoded with a unique number. Like here `41` represent `Kyoto` , `24` represent `Karasuma` and so on. You can check dataset before conding above too.

Define training and testing set

In [416]:
```
X= selected_df.drop('TotalRating', axis=1)     # X contain features using which we will train our model
y= selected_df['TotalRating']                  # Y contain "output feature" which we want to predict
```

In [417]:
```
# split data into training and testing  data
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state = 0)
```

Preparing the Models.

In [418]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [419]:
```
def create_models(models):
    all_models = []

    for model in models:
        if model == "LinearSVR":
            all_models.append(('LinearSVR', LinearSVR()))
        elif model == "RandomForestRegressor":
            all_models.append(('RandomForestRegressor', RandomForestRegressor()))
        elif model == "GradientBoostingRegressor":
            all_models.append(('GradientBoostingRegressor', GradientBoostingRegressor()))


    return all_models
```

Training the Models.

In [420]:
```
def fit_models(all_models, train_X, test_X, train_y, test_y):
    t = PrettyTable(['Model','error'])
    print('')
    #Iterating all models one by one
    for name, model in all_models:
        print('Fitting:',name)
        trained_model = model.fit(train_X,train_y.values)
        prediction = trained_model.predict(test_X)
        error = mean_absolute_error(test_y, prediction)

        t.add_row([name, round(error,3)])
    print('\n\nDetailed Performance Of All Models.')
    print("====================================")
    print(t)
```

Main

In [421]:
```
models = ["LinearSVR","RandomForestRegressor","GradientBoostingRegressor"]

# Create Models
all_models = create_models(models)

# Model Evaluation
fit_models(all_models, train_X, test_X, train_y, test_y)
```

```
Fitting: LinearSVR
Fitting: RandomForestRegressor
Fitting: GradientBoostingRegressor


Detailed Performance Of All Models.
=====================================
+--------------------------+-------+
|           Model          | error |
+--------------------------+-------+
|         LinearSVR        | 1.379 |
|   RandomForestRegressor  | 0.173 |
| GradientBoostingRegressor | 0.172 |
+--------------------------+-------+
```

As you can see in above comparison `GradientBoostingRegressor` have minimum error. I am going to select it as best model.

In [422]:
```
BestModel=GradientBoostingRegressor()
```

Train Best Model on All Data

In [423]:
```
x_alldata= selected_df.drop('TotalRating', axis=1)
y_alldata= selected_df['TotalRating']
```

In [424]:
```
BestModel.fit(x_alldata, y_alldata)
```

Out[424]:
```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.1, loss='ls', max_depth=3, max_features=None,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=1,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=100, presort='auto', random_state=None,
             subsample=1.0, verbose=0, warm_start=False)
```

Take Input from User

In [425]:
```
Station=input("Please Enter Station here: ")
avgDinnerPrice=int(input("Please Enter your avg Dinner Price: "))
```

```
Please Enter Station here: kyoto
Please Enter your avg Dinner Price: 5600
```

In [426]:
```
dictt={'Station':[Station],'avgDinnerPrice':[avgDinnerPrice]}
```

Convert User Input into Feature Vector

In [427]:
```
input1 = pd.DataFrame(dictt)
print("User Input in Actual Dataframe for : ")
input1
```

```
User Input in Actual Dataframe for :
```

Out[427]:

	Station	avgDinnerPrice
0	kyoto	5600

In [428]:
```
labelEncoder_Station.transform(input1.Station)
```

Out[428]:
```
array([41], dtype=int64)
```

In [429]:
```
input1['Station'] =labelEncoder_Station.transform(input1.Station)
```

In [430]:
```
print("User Input in Encoded Dataframe form : ")
input1
```

```
User Input in Encoded Dataframe form :
```

Out[430]:

	Station	avgDinnerPrice
0	41	5600

Apply Trained Model on Feature Vector of Unseen Data

In [431]:
```
y_predict = BestModel.predict(input1)
```

Output Prediction to User

In [433]:
```
print("Prediction : {} is a Predicted Rating for given Station. ".format((y_predict)))
```

```
Prediction : [3.25306596] is a Predicted Rating for given Station.
```

kyoto restaurant

ORIGINALITY REPORT

20%	12%	3%	15%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Monash University Student Paper	4%
2	tutorialedge.net Internet Source	3%
3	Submitted to Parkland College Student Paper	3%
4	robertmitchellv.com Internet Source	2%
5	www.datacamp.com Internet Source	1%
6	es.scribd.com Internet Source	1%

7	www.geeksforgeeks.org Internet Source	1%
8	xohkm.koloroj.eu Internet Source	<1%
9	Submitted to Cardiff University Student Paper	<1%
10	Rui Sarmento, Vera Costa. "chapter 2 Introduction to Programming R and Python Languages", IGI Global, 2017 Publication	<1%
11	Submitted to RMIT University Student Paper	<1%
12	Manohar Swamynathan. "Mastering Machine Learning with Python in Six Steps", Springer Science and Business Media LLC, 2017 Publication	<1%
13	Ekaba Bisong. "Building Machine Learning and Deep Learning Models on Google Cloud Platform", Springer Science and Business Media LLC, 2019 Publication	<1%

14	Submitted to Birkbeck College Student Paper	<1%
15	www.awesomestats.in Internet Source	<1%
16	Submitted to University of Technology, Sydney Student Paper	<1%
17	Submitted to University College London Student Paper	<1%
18	sakuravillage.jp Internet Source	<1%
19	Submitted to Institute of Technology Blanchardstown Student Paper	<1%
20	search.aol.com Internet Source	<1%
21	nihondebaito.com Internet Source	<1%
22	Submitted to Rochester Institute of Technology Student Paper	<1%

23	www.tripadvisor.com Internet Source	<1%
24	travel4apurpose.blogspot.com Internet Source	<1%
25	Submitted to Nanyang Technological University Student Paper	<1%
26	Puneet Mathur. "Machine Learning Applications Using Python", Springer Science and Business Media LLC, 2019 Publication	<1%

Exclude quotes	Off	Exclude matches	< 4 words
Exclude bibliography	On		