Travel Packages Prediction

A Project Report

Submitted by:

Sonali Gupta (52255102716)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHONOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

at

MAHAVIR SWAMI INSTITUTE OF TECHNOLOGY

SONEPAT

(AFFILIATED TO GGSIPU DWARKA, NEW DELHI)

(2016-2020)

DECLARATION OF THE STUDENT

I, **Sonali Gupta** ROLLNO **52255102716** hereby declare that the project entitled **"Travel Prediction Packages"** submitted for the B. Tech. (CSE) degree is my original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.

Signature of the Student

Place:

Date:

Certificate of the Guide

This is to certify that the project titled "**Travel Packages Prediction**" is the bona fide work carried out by **Sonali Gupta**, a student of B Tech (CSE) of **Mahavir Swami Institute of Technology, Sonipat (Haryana)** affiliated to **Guru Gobind Singh Indraprastha University, Delhi(India)** during the academic year 2019-20, in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (Computer Science and Engineering) and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Signature of the Guide

Place:

Date:

Acknowledgment

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to our final year project manager and the head of the project, **MS SHRUTY AHUJA,** whose contribution in stimulating suggestions and encouragement, helped me to coordinate and whose have invested her full effort in guiding the team in achieving the goal. my project especially in writing this report.

Furthermore, I would also like to acknowledge with much appreciation the crucial role of the staff of **MS RUCHIKA PHARSWAN**, who gave the permission to use all required equipment and the necessary materials to complete the project "***TRAVEL PACKAGES PREDICTION***". I have to appreciate the guidance given by other supervisor as well as the panels especially in our project presentation that has improved our presentation skills thanks to their comment and advices.

Abstract

The purpose of this project is to show the application of a set of intelligent data analysis techniques to about 7 million of online travel reviews, with the aim of automatically extracting useful information. The reviews, collected from popular online tourism-related review platforms, are all those posted by reviewers across Europe. To carry out the study, the following methodology is applied: a preliminary statistical analysis is performed to acquire general knowledge about the datasets, such as the geographical distribution of reviewers, their activities, and a comparison among the time of visits and the average scores of the reviews.

Now a days there are many travel agencies providing different trip packages. In our project we will predict how many packages and what type of package are required based on travel review ratings of Europe. It will help other travel agencies how many packages and what type of package should they launched instead of launching n number of packages.

Our dataset is Classified, technique used is Unsupervised learning technique using concept of Clustering and the algorithm used is K-Means Algorithm or Hard Clustering or Exclusive Clustering. This data set is populated by capturing user ratings from Google reviews. Reviews on attractions from 24 categories across Europe are considered. Google user rating ranges from 1 to 5 and average user rating per category is calculated.

Then, Machine Learning techniques are applied to extract and compare travel-related review platforms. Finally, K- Means Clustering, Unsupervised Learning algorithm is applied, to extract preferred destinations for distinct groups of reviewers, by mining interesting associations among the countries of origin of the reviewers and the most frequent destinations visited. By elaborating the available data, it is possible to automatically disclose valuable information for consumers and providers. The information automatically extracted can be exploited, for example, to build a recommender system for customers or a market analysis tool for service providers.

List of Figures

Table of Contents

1. PROBLEM DEFINITION AND SCOPE OF PROJECT

1.1 Introduction:

Data science is changing the face of the travel industry. It helps travel and tourism businesses to provide unique travel experience and high satisfaction rates, preserving personal touch. In recent years data science has become one of the most promising technologies bringing changes to various industries. It has shifted the way we travel and our attitude toward traveling arrangements. With a vast variety of solutions provided by the application of data science and machine learning, travel business can learn their clients' needs and preferences to provide them with the best possible services and offers.

Tourism is one of the most rapidly growing global industries and tourism forecasting is becoming an increasingly important activity in planning and managing the industry.

1.2 Problem Definition:

Build a machine learning model To Predict number of packages to be launched by a travel agency based on review ratings.

1.3 Data Set Information:

Our dataset is Classified, technique used is Unsupervised learning technique using concept of Clustering and the algorithm used is K-Means Algorithm or Hard Clustering or Exclusive Clustering. This data set is populated by capturing user ratings from Google reviews. Reviews on attractions from 24 categories across Europe are considered. Google user rating ranges from 1 to 5 and average user rating per category is calculated.

1.4 Purpose:

The purpose of this project is to show the application of a set of intelligent data analysis techniques to about millions of online travel reviews, with the aim of automatically extracting useful information.

Now a days, there are many travel agencies providing different trip packages. In our project, we will predict how many packages and what kind of packages are required based on travel review ratings of Europe. It will help other travel agencies how many packages should they launch instead of launching 'N' number of packages.

1.5 Objective:

1. To manage all the travel requirements of the companies while providing ease and sort of convenience to the customers.
2. To add value to the travel sector of different companies
3. To offer the traveling services at very reasonable prices (Minimization of the travel cost).
4. Time management
5. Minimization of distance
6. Getting and Staying Profitable
    a. collaborative efficiency,
    b. optimized logistics,
    c. quality improvement and
    d. long-term stability with an overall outcome of creating a supply packages.
7. Productivity of People and Resources
8. Excellent Customer Service
9. Employee Attraction and Retention
10. Mission-driven Core Values
11. Sustainable Growth
12. Maintaining a Healthy Cash Flow
13. Dealing with Change
14. Reaching the Right Customers
15. Staying Ahead of the Competition

1.6 Project Scope:

"Smart Work is the passport to the future,
for tomorrow belongs to those
who prepare for it today."

The aim of the projects is the satisfaction of the end-user. Whether they meet the end-users expectations and accept the product, service or process. The end-users could be your customers or your internal team. It is all about the customer: what the customer wants and what they get. For customers, this includes pricing, value, and quality of products/services as well as availability, delivery and return policies. Generally, every customer wants a product or service that solves their problem, worth their money, and is delivered with amazing customer service. For employees, this includes the effectiveness and efficiency of new operational processes. Ultimately, our project scope is to give better outcomes to whoever our end users may be. Our priority is to control costs, satisfy customers better, satisfy employees better and become more transparent.

1.7 Technologies To be used(Front end & Back end):

1.7.1 Front end Technology:

1. OS: Windows 10
2. Programming language: Python
3. Software: MS Excel, Anaconda

1.7.2 Backend Technology:

1. Dataset: CSV format

2. LITERATURE SURVEY

2.1 Existing System

2.1.1 Description: Open Science has become a movement capable of greatly accelerating the production and dissemination of knowledge (Friesike, 2015). Thanks to this and other paradigms, researchers are publishing their datasets on the Web, thus encouraging their reuse and contributing to improved collaboration among members of the scientific community.

2.1.2 Dataset Description:

a) Researcher's Profile: Describes the basic data of a researcher, or a group of researchers associated with an institution, according to their profile available in Google Scholar. In this case the scholarly library (v. 0.2.4) was used to extract information from the authors of this site.

b) UCI Datasets: Describes the characteristics of each of the data sets available on the UCI Machine Learning Repository site. The python bs4 library (Lawson, 2015) is used to collect the metadata of each dataset.

c) Open Data in Tourisms: Tourism is by nature an industry in which marketing communications strongly depend on data exchange (Mack, Blose, & Pan, 2008). In today's rapidly changing world, various forms of data related to tourism activities and services are produced and utilized across a range of online applications (Buhalis & Law, 2008). This is primarily the outcome of the increasing ability to digitize growing volumes of data, and the development of open-sources and open data policies (Sabou et al., 2015, Soualah-Alila et al., 2016). For tourist destinations there are significant opportunities to use open data to develop cultural sights, transportation, marketing and the environment (Wiggins & Crowston, 2011). As people have increasingly focused on the quality of the tourist experience, the demand for open data in tourism and hospitality research has becomes intense (Wu et al., 2014). A growing amount of tourism-related open data is now available on the platform in XML, CSV, or JSON format (Wu et al., 2014).

d) Predictive Model: Open data analyses might support tourism managers in predicting tourists' judgements about a certain tourist attraction. To achieve this goal, it is necessary to introduce predictive models that support information selection within a huge amount of data.

2.2 Proposed System:

The increasing amount of user-generated content spread via social networking services such as reviews, comments, and past experiences, has made a great deal of information available. Tourists can access this information to support their decision making process. This information is freely accessible online and generates so-called “open data or dataset”. While many studies have investigated the effect of online reviews on tourists' decisions, none have directly investigated the extent to which open data analyses might predict tourists' response to a certain destination.

The series of diverse analyses aim at providing fine-grained insights as well as improving the decision processes of consumers and providers. In fact, the analyses take into account: i) the reviewers’ activities, ii) the kind and frequencies of review terms, and iii) the preferred destinations of reviewers.
The aim of this study is to extract useful information that is originally implicit in review data, with the main purpose of supporting both providers and potential customers, helping the former to adapt their services and the latter to improve their decision processes.
To Predict number of packages to be launched by a travel agency based on review ratings which will help other travel agencies, how many packages should they launched instead of launching ‘N’ number of packages.

This data set, csv format named review_ratings.csv, is populated by capturing user ratings from Google reviews. Reviews on attractions from 24 categories across Europe are considered. Google user rating ranges from 1 to 5 and average user rating per category is calculated.

It has 5456 Instances (rows) and 25 Attributes (columns). Different techniques inherited from the wide field of Intelligent Data Analysis like Unsupervised learning technique using concept of Clustering and the algorithm used is K-Means Algorithm also known as Hard Clustering/Exclusive Clustering are exploited, which have been crawled by Jupyter Notebook (Anaconda). Overall, this dataset deals with the reviews of Churches, Resorts, Beaches, Parks, Theatres, Museums, Malls, Zoo, Restaurants, Pubs/Bars, Local services, Burger/Pizza shops, Hotels/Other lodgings, Juice bars, Art galleries, Dance clubs, Swimming pools, Gyms, Bakeries, Beauty & Spa, Cafes, View-points, Monuments, Gardens are collected and analysed.

2.2.1 Libraries Used:

a) **Pandas:** data manipulation and analysis
b) **Matplotlib:** Comprehensive 2D/3D plotting
c) **Numpy:** used for collection of high-level mathematical functions
d) **Seaborn:** data visualization library based on matplotlib
e) **Sckikit-Learn:** provides a range of supervised and unsupervised learning algorithms

To this end, our study contributes to the process of predicting tourists' future preferences via software that analyzes a large set of the dataset (i.e. review_ratings) that is freely available. This is devised by generating the classification function and the best model for predicting the destination tourists would potentially select.

2.3 Feasibility Study:

- Technical Feasibility
- Social and Operational Feasibility
- Time Feasibility

3. Methodology/Planning of Project

To carry out the study, the following methodology is applied: a preliminary statistical analysis is performed to acquire general knowledge about the datasets, such as the geographical distribution of reviewers, their activities, and a comparison among the time of visits and the average scores of the reviews. Then, Unsupervised Learning techniques and K- Means Clustering algorithm is applied, applied t**o** Predict number of packages to be launched by a travel agency based on review ratings by the most frequent destinations visited. By elaborating the available data, it is possible to automatically disclose valuable information for consumers and providers. The information automatically extracted can be exploited, for example, to build a recommender system for customers or a market analysis tool for service providers.

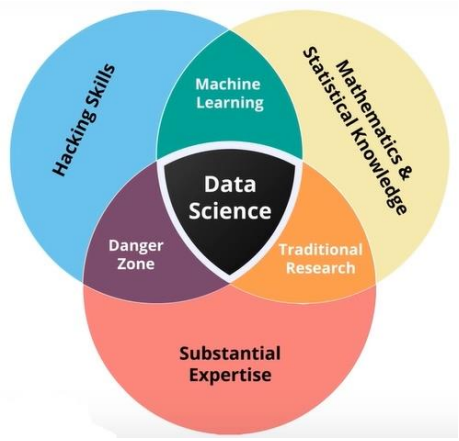


Fig: 1

3.1 Brief Description of Algorithms Used:

3.1.1 Data Science: is the area of study which involves extracting insights from various vast amounts of data by the use of various scientific methods, algorithms and processes helps to discover hidden pattern from the raw data. It is the process of using data to find solution to predict outcome for a problem statement. It is also known as Data-Driven Science.

3.1.2 Machine Learning: It's a class of algorithm which is data- driven, i.e. unlike "Normal" algorithm. It is the data that "Tells" what the "Good Answer" is. It's the application of Artificial Intelligence that provides systems, the ability to automatically learn and improve from experience without being implicitly programmed.

Getting computers to program themselves and also teaching them to make decision using data "where writing software is the bottleneck, let the data do the work instead."

3.1.3 Machine Learning Life Cycle:

Step-1: **Collecting Data:** Data is collected from various sources in a server.

Step-2: **Data Wrangling:** It is a process of cleaning and converting the raw data into a format that allows convenient consumption.

Step-3: **Analyse Data:** Data is analysed to select and filter data required to prepare the model. In this, we take data, use Machine Learning Algorithms to create a particular model.

Step-4: **Train Algorithm**: Here, we are training the model. Here, we use the dataset and Algorithm is trained on training dataset through which the Algorithm understand the pattern and rules which governs the particular data.

Step-5: **Test Algorithm:** Testing dataset determines the accuracy of the model and tells us the accuracy of the model.

Step-6: **Operation and Optimisation:** If the speed and accuracy of the model is acceptable, then that model should be deployed in the real system. The model i.e. used in the available data.

Models improve with the amount of available data is used to create the data. The result of the data needs to incorporated in the business strategy. After the model is deployed based upon the performance, the model is updated and improved. If there is a drip in the performance, the model is retrained.

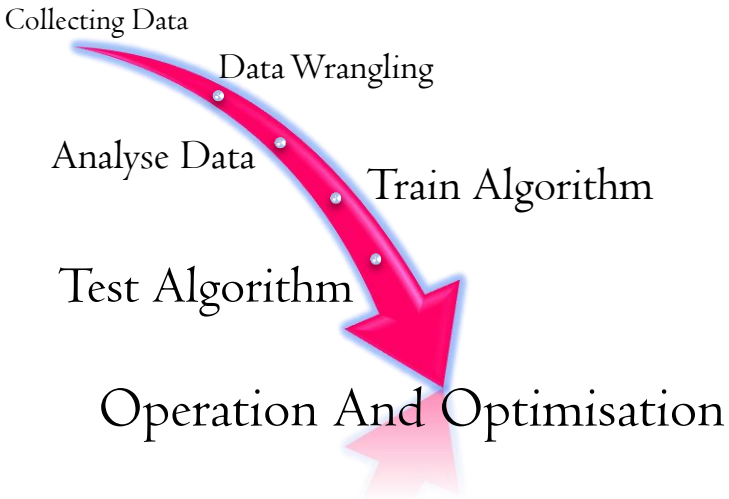


Fig: 2

3.1.5 Unsupervised learning: It is a type of machine learning algorithm used to draw influences from data sets consisting of input data without labelled response. Sometimes the data is unstructured and unlabelled, so it becomes difficult classify data into different categories. It is used to help this problem. This learning is used to cluster the data into different classes on the basis of different properties.

3.1.6 Clustering: It is the process of dividing the datasets into groups, consisting of similar data points. The goal of clustering is to maximize the similarity of observations within a cluster and maximize the dissimilarity between clusters.

3.1.7 K-Means Clustering: The process by which objects are classified into a predefined number of groups so that they are as much dissimilar as possible from one group to another group, but as much similar as possible within each group.

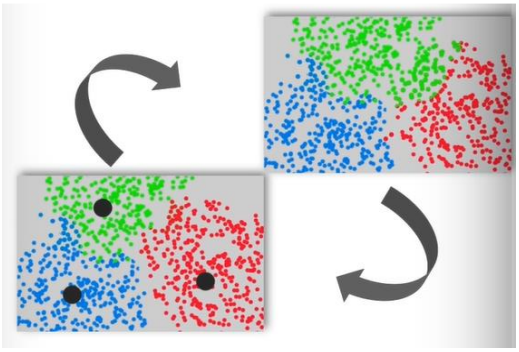


Fig: 3

3.1.8 K-Means Algorithm Steps:

Step-1: First, we need to decide the number of clusters to be made by guessing.
Step-2: Then, we provide centroid of all the clusters by guessing.
Step-3: The algorithm calculates Euclidean distance of the point from each centroid and assigns the point to the closest cluster.
Step-4: Next, the centroids are calculated again, when we have our new cluster.
Step-5: The distance of the points from the centre of the clusters are calculated again and points are assigned to the closest cluster.
Step-6: And then, again the new centroid for the cluster is calculated.
Step-7: These steps are repeated until we have a repetition in centroid or new centroids are very close to the previous ones.

Source Code

In [1]:
```
#  Import libraries

import pandas as pd                                    # Pandas for reading and writing spreadsheets
import numpy as np                                     # Numpy for carrying out efficient computations
from matplotlib import pyplot as plt                   # Matplotlib for visualization of data
import seaborn as sns                                  # Sklearn for clustering
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
```

In [2]:
```
# import csv files
read_df=pd.read_csv('review_ratings.csv')
```

In [3]:
```
# print the data set with number of rows and columns
read_df
```

Out[3]:

	User	Category 1	Category 2	Category 3	Category 4	Category 5	Category 6	Category 7	Category 8	Category 9	...	Category 15
0	User 1	0.00	0.00	3.63	3.65	5.00	2.92	5.00	2.35	2.33	...	1.74
1	User 2	0.00	0.00	3.63	3.65	5.00	2.92	5.00	2.64	2.33	...	1.74
2	User 3	0.00	0.00	3.63	3.63	5.00	2.92	5.00	2.64	2.33	...	1.74
3	User 4	0.00	0.50	3.63	3.63	5.00	2.92	5.00	2.35	2.33	...	1.74
4	User 5	0.00	0.00	3.63	3.63	5.00	2.92	5.00	2.64	2.33	...	1.74
5	User 6	0.00	0.00	3.63	3.63	5.00	2.92	5.00	2.63	2.33	...	1.74
6	User	0.00	5.00	3.63	3.63	5.00	2.92	3.03	2.35	2.33		1.75

In [4]:
```
# head() defines read first 5 data
read_df.head()
```

Out[4]:

	User	Category 1	Category 2	Category 3	Category 4	Category 5	Category 6	Category 7	Category 8	Category 9	...	Category 15	Catego
0	User 1	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.35	2.33	...	1.74	0.
1	User 2	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.64	2.33	...	1.74	0.
2	User 3	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	...	1.74	0.
3	User 4	0.0	0.5	3.63	3.63	5.0	2.92	5.0	2.35	2.33	...	1.74	0.
4	User 5	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	...	1.74	0.

5 rows × 25 columns

In [5]:
```
# tail() defines read last 5 data
read_df.tail()
```

Out[5]:

	User	Category 1	Category 2	Category 3	Category 4	Category 5	Category 6	Category 7	Category 8	Category 9	...	Category 15	Cat
5451	User 5452	0.91	5.00	4.00	2.79	2.77	2.57	2.43	1.09	1.77	...	5.00	
5452	User 5453	0.93	5.00	4.02	2.79	2.78	2.57	1.77	1.07	1.76	...	0.89	
5453	User 5454	0.94	5.00	4.03	2.80	2.78	2.57	1.75	1.05	1.75	...	0.87	
5454	User 5455	0.95	4.05	4.05	2.81	2.79	2.44	1.76	1.03	1.74	...	5.00	
5455	User 5456	0.95	4.07	5.00	2.82	2.80	2.57	2.42	1.02	1.74	...	0.85	

5 rows × 25 columns

In [6]:
```
# changing column name
column_names=['U_Id','Churches','Resorts','Beaches','Parks','Theatres','Museums','Malls','Zoo','Resta
read_df.columns=column_names
```

In [7]:
```
# print columns
read_df
```

Out[7]:

	U_Id	Churches	Resorts	Beaches	Parks	Theatres	Museums	Malls	Zoo	Restaurants	...	Art_Galleries	Dance_
0	User 1	0.00	0.00	3.63	3.65	5.00	2.92	5.00	2.35	2.33	...	1.74	
1	User 2	0.00	0.00	3.63	3.65	5.00	2.92	5.00	2.64	2.33	...	1.74	
2	User 3	0.00	0.00	3.63	3.63	5.00	2.92	5.00	2.64	2.33	...	1.74	
3	User 4	0.00	0.50	3.63	3.63	5.00	2.92	5.00	2.35	2.33	...	1.74	
4	User 5	0.00	0.00	3.63	3.63	5.00	2.92	5.00	2.64	2.33	...	1.74	
5	User 6	0.00	0.00	3.63	3.63	5.00	2.92	5.00	2.63	2.33	...	1.74	
6	User 7	0.00	5.00	3.63	3.63	5.00	2.92	3.03	2.35	2.33	...	1.75	

In [8]:
```
# shape define number of rows and columns
read_df.shape
```

Out[8]:
```
(5456, 25)
```

In [9]:
```
# describe shows mean, std, min,max
read_df.describe()
```

Out[9]:

	Churches	Resorts	Beaches	Parks	Theatres	Museums	Malls	Zoo	Restaurants
count	5456.000000	5456.000000	5456.000000	5456.000000	5456.000000	5456.00000	5456.000000	5456.000000	5456.000000
mean	1.455720	2.319707	2.489331	2.796886	2.958941	2.89349	3.351395	2.540795	3.126019
std	0.827604	1.421438	1.247815	1.309159	1.339056	1.28240	1.413492	1.111391	1.356802
min	0.000000	0.000000	0.000000	0.830000	1.120000	1.11000	1.120000	0.860000	0.840000
25%	0.920000	1.360000	1.540000	1.730000	1.770000	1.79000	1.930000	1.620000	1.800000
50%	1.340000	1.905000	2.060000	2.460000	2.670000	2.68000	3.230000	2.170000	2.800000
75%	1.810000	2.682500	2.740000	4.092500	4.312500	3.84000	5.000000	3.190000	5.000000
max	5.000000	5.000000	5.000000	5.000000	5.000000	5.00000	5.000000	5.000000	5.000000

8 rows × 23 columns

In [10]:
```
# info shows detail of all columns
read_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5456 entries, 0 to 5455
Data columns (total 25 columns):
U_Id                  5456 non-null object
Churches              5456 non-null float64
Resorts               5456 non-null float64
Beaches               5456 non-null float64
Parks                 5456 non-null float64
Theatres              5456 non-null float64
Museums               5456 non-null float64
Malls                 5456 non-null float64
Zoo                   5456 non-null float64
Restaurants           5456 non-null float64
Pubs_Bars             5456 non-null float64
Local_Services        5456 non-null object
Burger_Pizza_Shops    5455 non-null float64
Hotels                5456 non-null float64
Juice_Bars            5456 non-null float64
Art_Galleries         5456 non-null float64
Dance_Clubs           5456 non-null float64
Swimming_Pool         5456 non-null float64
Gyms                  5456 non-null float64
Bakeries              5456 non-null float64
Beauty_Spa            5456 non-null float64
Cafes                 5456 non-null float64
View_Points           5456 non-null float64
Monuments             5456 non-null float64
Gardens               5455 non-null float64
dtypes: float64(23), object(2)
memory usage: 1.0+ MB
```

In [11]:
```
# dtype show each data types
read_df.dtypes
```

Out[11]:
```
U_Id                 object
Churches            float64
Resorts             float64
Beaches             float64
Parks               float64
Theatres            float64
Museums             float64
Malls               float64
Zoo                 float64
Restaurants         float64
Pubs_Bars           float64
Local_Services       object
Burger_Pizza_Shops  float64
Hotels              float64
Juice_Bars          float64
Art_Galleries       float64
Dance_Clubs         float64
Swimming_Pool       float64
Gyms                float64
Bakeries            float64
Beauty_Spa          float64
Cafes               float64
View_Points         float64
Monuments           float64
Gardens             float64
dtype: object
```

In [12]:
```
# checking type as float
read_df.Local_Services=read_df.Local_Services.astype('float')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-12-3db2c959803f> in <module>
      1 # checking type as float
----> 2 read_df.Local_Services=read_df.Local_Services.astype('float')

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in astype(self, dtype, copy, errors, **kwargs)
   5689             # else, only a single dtype is given
   5690             new_data = self._data.astype(dtype=dtype, copy=copy, errors=errors,
-> 5691                                          **kwargs)
   5692             return self._constructor(new_data).__finalize__(self)
   5693

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\internals\managers.py in astype(self, dtype, **kwargs)
    529
    530     def astype(self, dtype, **kwargs):
--> 531         return self.apply('astype', dtype=dtype, **kwargs)
```

In [13]:
```
Local_Services_mean=read_df['Local_Services'][read_df['Local_Services']!='2\t2.']
read_df['Local_Services'][read_df['Local_Services']=='2\t2.']=np.mean(Local_Services_mean.astype('flo
read_df['Local_Services']=read_df['Local_Services'].astype('float')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
```

In [14]:
```
read_df.dtypes
```

Out[14]:
```
U_Id                  object
Churches             float64
Resorts              float64
Beaches              float64
Parks                float64
Theatres             float64
Museums              float64
Malls                float64
Zoo                  float64
Restaurants          float64
Pubs_Bars            float64
Local_Services       float64
Burger_Pizza_Shops   float64
Hotels               float64
Juice_Bars           float64
Art_Galleries        float64
Dance_Clubs          float64
Swimming_Pool        float64
Gyms                 float64
Bakeries             float64
Beauty_Spa           float64
Cafes                float64
View_Points          float64
Monuments            float64
Gardens              float64
dtype: object
```

In [15]:
```
# checking the columns if it contains null values(NA)[1] or not[0]
# isnull() defines checks null values
# sum() defines no. of null values
read_df[column_names].isnull().sum()
```

Out[15]:
```
U_Id                 0
Churches             0
Resorts              0
Beaches              0
Parks                0
Theatres             0
Museums              0
Malls                0
Zoo                  0
Restaurants          0
Pubs_Bars            0
Local_Services       0
Burger_Pizza_Shops   1
Hotels               0
Juice_Bars           0
Art_Galleries        0
Dance_Clubs          0
Swimming_Pool        0
Gyms                 0
Bakeries             0
Beauty_Spa           0
Cafes                0
View_Points          0
Monuments            0
Gardens              1
dtype: int64
```

In [16]:
```
# remove rows having null values
read_df=read_df.dropna()
```

In [17]:
```
# checking the columns with new variable
read_df.isnull().sum()
```

Out[17]:
```
U_Id                 0
Churches             0
Resorts              0
Beaches              0
Parks                0
Theatres             0
Museums              0
Malls                0
Zoo                  0
Restaurants          0
Pubs_Bars            0
Local_Services       0
Burger_Pizza_Shops   0
Hotels               0
Juice_Bars           0
Art_Galleries        0
Dance_Clubs          0
Swimming_Pool        0
Gyms                 0
Bakeries             0
Beauty_Spa           0
Cafes                0
View_Points          0
Monuments            0
Gardens              0
dtype: int64
```

In [18]:
```
# check data type of every column
read_df.dtypes
```

Out[18]:
```
U_Id                  object
Churches             float64
Resorts              float64
Beaches              float64
Parks                float64
Theatres             float64
Museums              float64
Malls                float64
Zoo                  float64
Restaurants          float64
Pubs_Bars            float64
Local_Services       float64
Burger_Pizza_Shops   float64
Hotels               float64
Juice_Bars           float64
Art_Galleries        float64
Dance_Clubs          float64
Swimming_Pool        float64
Gyms                 float64
Bakeries             float64
Beauty_Spa           float64
Cafes                float64
View_Points          float64
Monuments            float64
Gardens              float64
dtype: object
```

In [19]:
```
# describe first 12 columns
read_df[column_names[:12]].describe()
```

Out[19]:

	Churches	Resorts	Beaches	Parks	Theatres	Museums	Malls	Zoo	Restaurant
count	5454.000000	5454.000000	5454.000000	5454.000000	5454.000000	5454.000000	5454.000000	5454.000000	5454.00000
mean	1.455746	2.320048	2.489059	2.797103	2.958904	2.893423	3.351476	2.541177	3.12654
std	0.827732	1.421576	1.247503	1.309188	1.338785	1.282101	1.413291	1.111398	1.35677
min	0.000000	0.000000	0.000000	0.830000	1.120000	1.110000	1.120000	0.860000	0.84000
25%	0.920000	1.360000	1.540000	1.730000	1.770000	1.790000	1.930000	1.620000	1.80000
50%	1.340000	1.910000	2.060000	2.460000	2.670000	2.680000	3.230000	2.170000	2.80000
75%	1.810000	2.687500	2.740000	4.097500	4.310000	3.837500	5.000000	3.190000	5.00000
max	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.00000

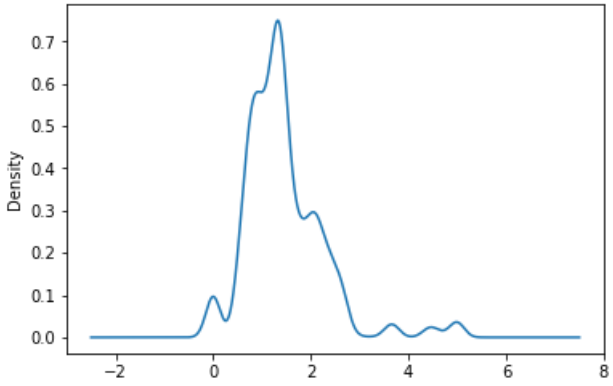
In [20]:
```
# describe after 12th column till last
read_df[column_names[12:]].describe()
```

Out[20]:

	Burger_Pizza_Shops	Hotels	Juice_Bars	Art_Galleries	Dance_Clubs	Swimming_Pool	Gyms	Bakerie
count	5454.000000	5454.000000	5454.000000	5454.000000	5454.000000	5454.000000	5454.000000	5454.00000
mean	2.078401	2.125820	2.190429	2.206140	1.192710	0.949349	0.822525	0.96925
std	1.249315	1.406682	1.576505	1.715848	1.107176	0.973628	0.948015	1.20288
min	0.780000	0.770000	0.760000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	1.290000	1.190000	1.030000	0.860000	0.690000	0.580000	0.530000	0.52000
50%	1.690000	1.610000	1.490000	1.330000	0.800000	0.740000	0.690000	0.69000
75%	2.287500	2.360000	2.740000	4.440000	1.160000	0.910000	0.840000	0.86000
max	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.00000

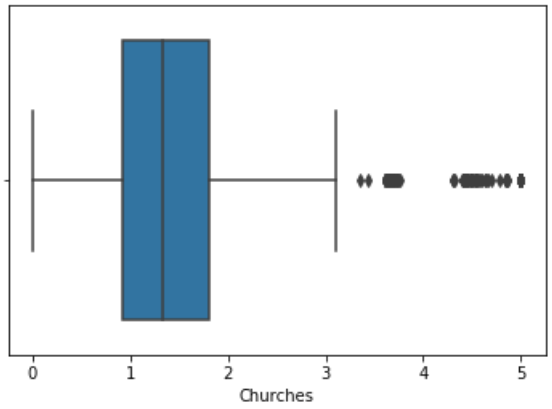
In [21]:
```
# plot density curve of Churches after cleaning null values
read_df.Churches.plot.density()
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa61349b0>
```



In [22]:
```
# plotting box plot of Churches
sns.boxplot(read_df['Churches'])
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa67e5240>
```



In [23]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Churches.describe()
```

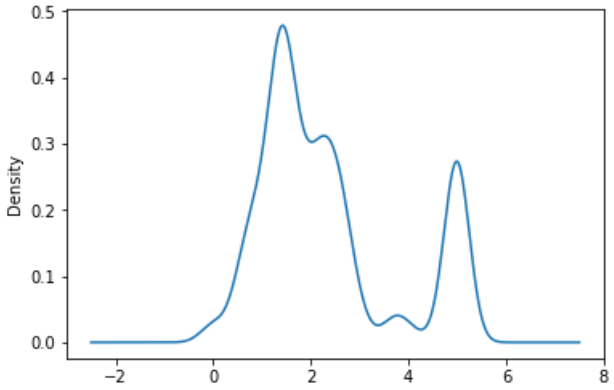
Out[23]:
```
count    5454.000000
mean        1.455746
std         0.827732
min         0.000000
25%         0.920000
50%         1.340000
75%         1.810000
max         5.000000
Name: Churches, dtype: float64
```

In [24]:
```
# displaying mode
read_df.Churches.mode()
```

Out[24]:
```
0    0.0
dtype: float64
```

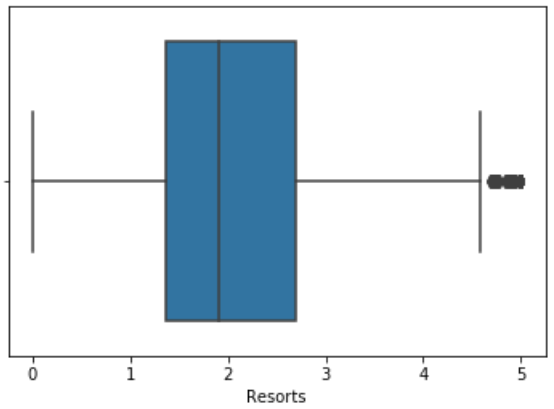
In [25]:
```
# plot density curve of Resorts after cleaning null values
read_df.Resorts.plot.density()
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa6853320>
```



In [27]:
```
# plotting box plot of Resorts
sns.boxplot(read_df['Resorts'])
```

Out[27]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa68efda0>
```



In [28]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Resorts.describe()
```

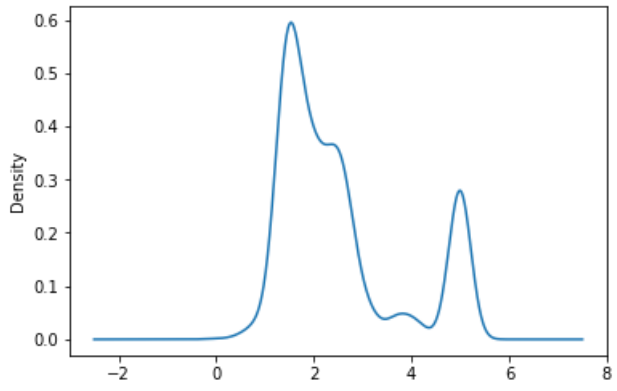
Out[28]:
```
count    5454.000000
mean        2.320048
std         1.421576
min         0.000000
25%         1.360000
50%         1.910000
75%         2.687500
max         5.000000
Name: Resorts, dtype: float64
```

In [30]:
```
# displaying mode
read_df.Resorts.mode()
```

Out[30]:
```
0    5.0
dtype: float64
```

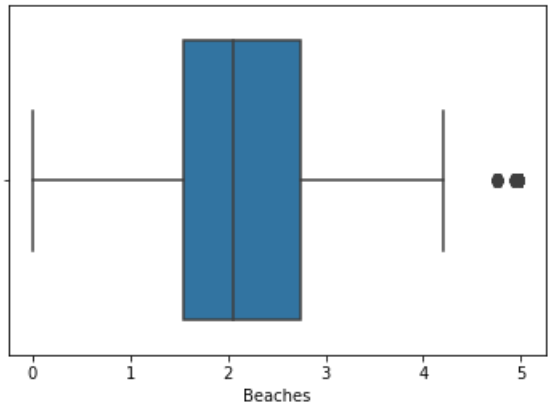
In [31]:
```
# plot density curve of Beaches after cleaning null values
read_df.Beaches.plot.density()
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa69438d0>
```



In [32]:
```
# plotting box plot of Beaches
sns.boxplot(read_df['Beaches'])
```

Out[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa69ad550>
```



In [33]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Beaches.describe()
```

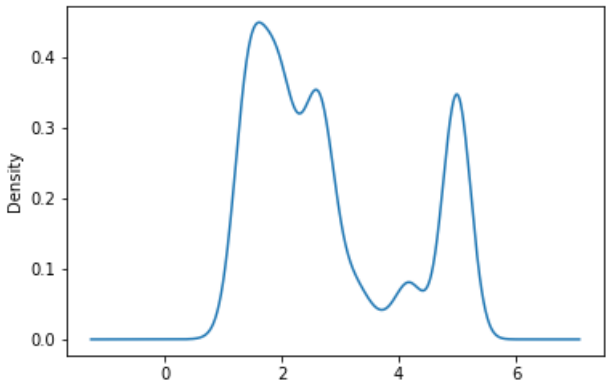
Out[33]:
```
count    5454.000000
mean        2.489059
std         1.247503
min         0.000000
25%         1.540000
50%         2.060000
75%         2.740000
max         5.000000
Name: Beaches, dtype: float64
```

In [34]:
```
# displaying mode
read_df.Beaches.mode()
```

Out[34]:
```
0    5.0
dtype: float64
```

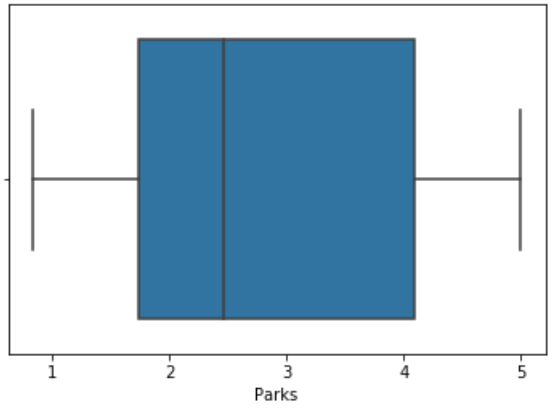
In [35]:
```
# plot density curve of Parks after cleaning null values
read_df.Parks.plot.density()
```

Out[35]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa6a1a128>
```



In [36]:
```
# plotting box plot of Parks
sns.boxplot(read_df['Parks'])
```

Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa6a625c0>
```



In [37]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Parks.describe()
```

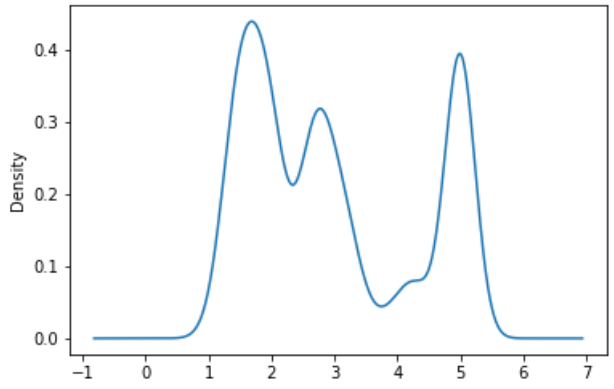
Out[37]:
```
count    5454.000000
mean        2.797103
std         1.309188
min         0.830000
25%         1.730000
50%         2.460000
75%         4.097500
max         5.000000
Name: Parks, dtype: float64
```

In [38]:
```
# displaying mode
read_df.Parks.mode()
```

Out[38]:
```
0    5.0
dtype: float64
```

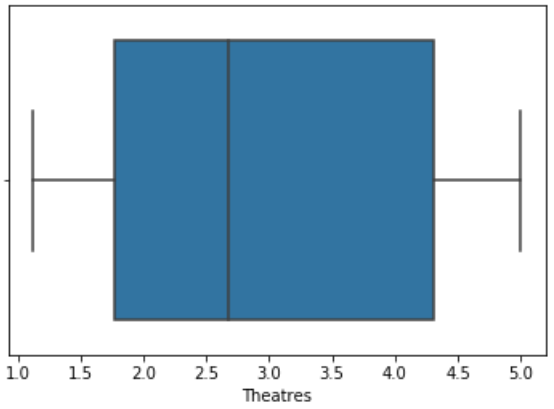
In [39]:
```
# plot density curve of Theatres after cleaning null values
read_df.Theatres.plot.density()
```

Out[39]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa6ab6ef0>
```



In [40]:
```
# plotting box plot of Theatres
sns.boxplot(read_df['Theatres'])
```

Out[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa6b1d7f0>
```



In [41]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Theatres.describe()
```

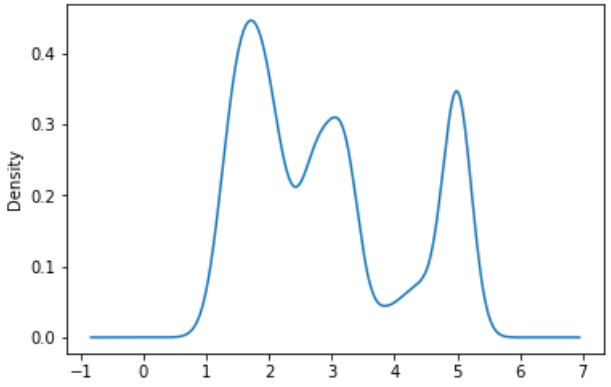
Out[41]:
```
count    5454.000000
mean        2.958904
std         1.338785
min         1.120000
25%         1.770000
50%         2.670000
75%         4.310000
max         5.000000
Name: Theatres, dtype: float64
```

In [42]:
```
# displaying mode
read_df.Theatres.mode()
```

Out[42]:
```
0    5.0
dtype: float64
```

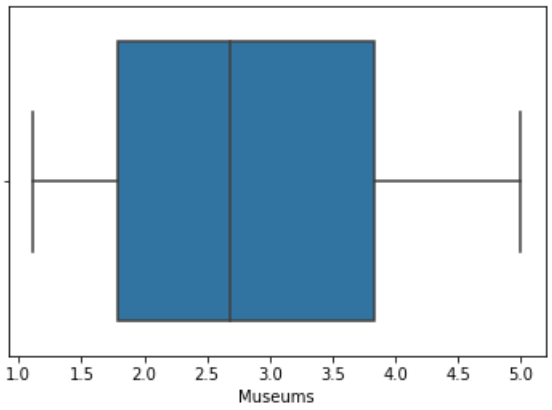
In [43]:
```
# plot density curve of Museums after cleaning null values
read_df.Museums.plot.density()
```

Out[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa6b72fd0>
```



In [45]:
```
# plotting box plot of Museums
sns.boxplot(read_df['Museums'])
```

Out[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa7c040f0>
```



In [46]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Museums.describe()
```

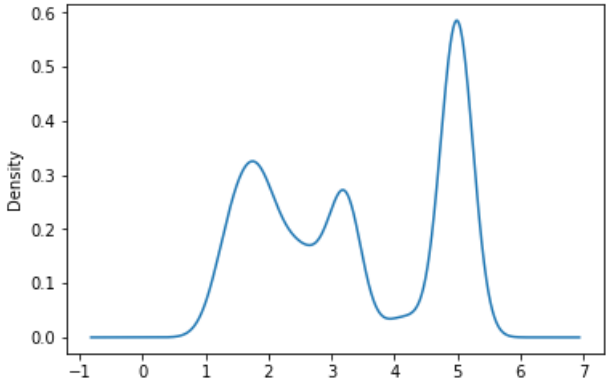
Out[46]:
```
count    5454.000000
mean        2.893423
std         1.282101
min         1.110000
25%         1.790000
50%         2.680000
75%         3.837500
max         5.000000
Name: Museums, dtype: float64
```

In [47]:
```
# displaying mode
read_df.Museums.mode()
```

Out[47]:
```
0    5.0
dtype: float64
```

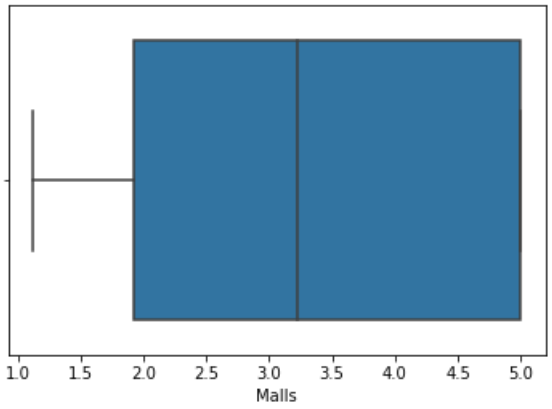
In [48]:
```
# plot density curve of Malls after cleaning null values
read_df.Malls.plot.density()
```

Out[48]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa7c6a208>
```



In [49]:
```
# plotting box plot of Malls
sns.boxplot(read_df['Malls'])
```

Out[49]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa7cc7438>
```



In [50]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Malls.describe()
```

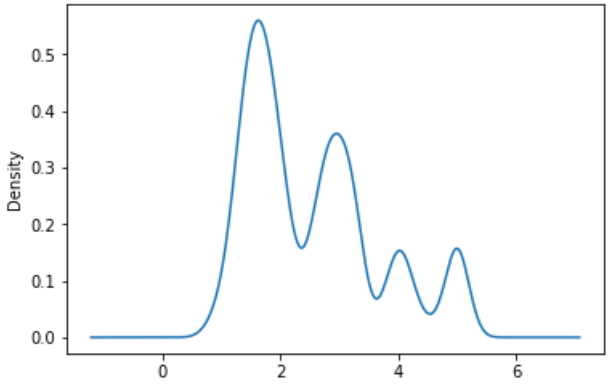
Out[50]:
```
count    5454.000000
mean        3.351476
std         1.413291
min         1.120000
25%         1.930000
50%         3.230000
75%         5.000000
max         5.000000
Name: Malls, dtype: float64
```

In [51]:
```
# displaying mode
read_df.Malls.mode()
```

Out[51]:
```
0    5.0
dtype: float64
```

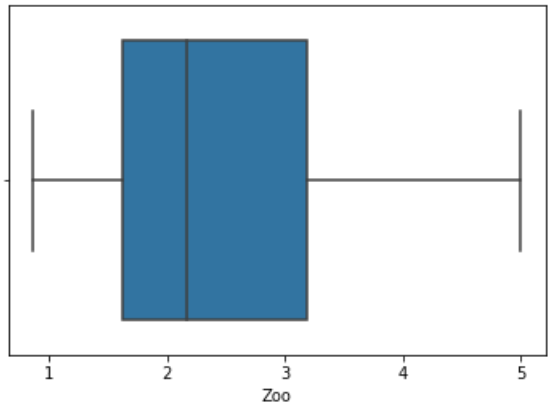
In [52]:
```
# plot density curve of Zoo after cleaning null values
read_df.Zoo.plot.density()
```

Out[52]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa7d2ca58>
```



In [53]:
```
# plotting box plot of Zoo
sns.boxplot(read_df['Zoo'])
```

Out[53]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa7d86588>
```



In [54]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Zoo.describe()
```

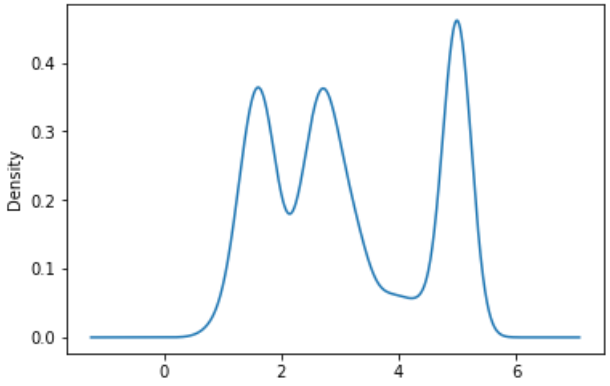
Out[54]:
```
count    5454.000000
mean        2.541177
std         1.111398
min         0.860000
25%         1.620000
50%         2.170000
75%         3.190000
max         5.000000
Name: Zoo, dtype: float64
```

In [55]:
```
# displaying mode
read_df.Zoo.mode()
```

Out[55]:
```
0    5.0
dtype: float64
```

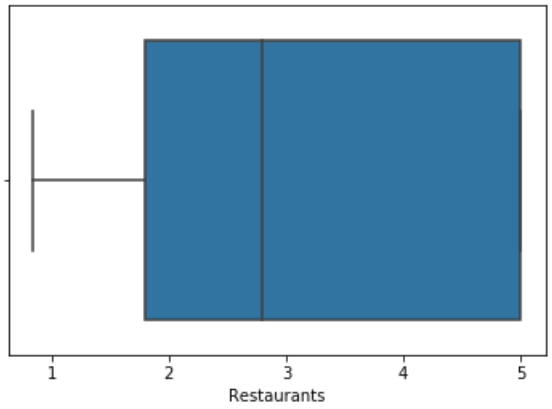
In [56]:
```
# plot density curve of Restaurants after cleaning null values
read_df.Restaurants.plot.density()
```

Out[56]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa7dddac8>
```



In [57]:
```
# plotting box plot of Restaurants
sns.boxplot(read_df['Restaurants'])
```

Out[57]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa7e2a550>
```



In [58]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Restaurants.describe()
```

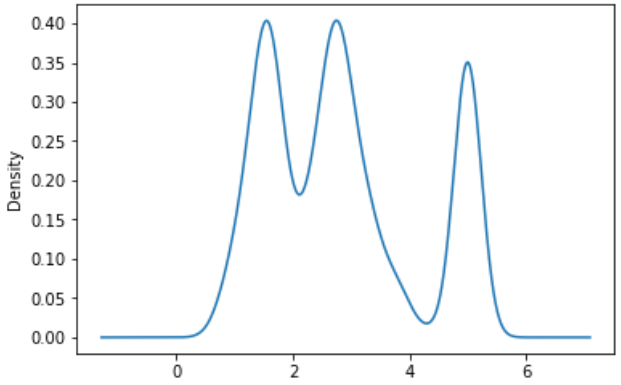
Out[58]:
```
count    5454.000000
mean        3.126542
std         1.356774
min         0.840000
25%         1.800000
50%         2.800000
75%         5.000000
max         5.000000
Name: Restaurants, dtype: float64
```

In [59]:
```
# displaying mode
read_df.Restaurants.mode()
```

Out[59]:
```
0    5.0
dtype: float64
```

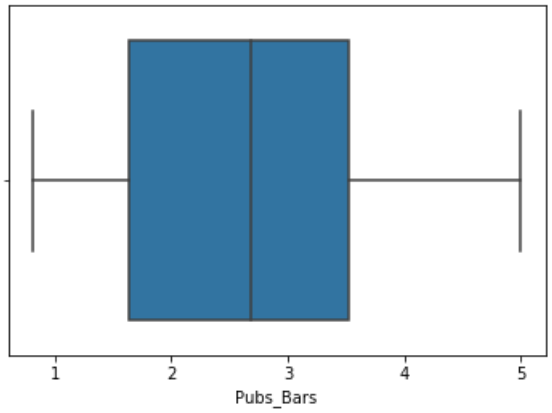
In [60]:
```
# plot density curve of Pubs_Bars after cleaning null values
read_df.Pubs_Bars.plot.density()
```

Out[60]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa7e67828>
```



In [61]:
```
# plotting box plot of Pubs_Bars
sns.boxplot(read_df['Pubs_Bars'])
```

Out[61]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa7e84b00>
```



In [62]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Pubs_Bars.describe()
```

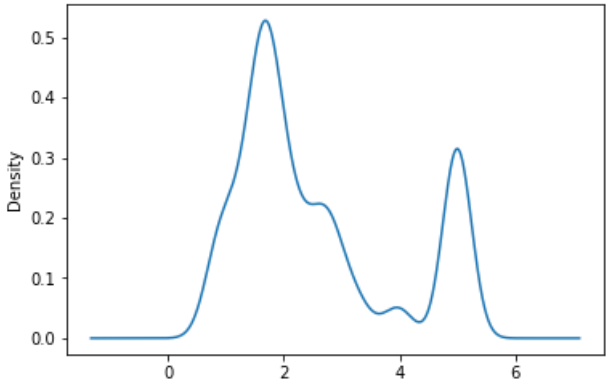
Out[62]:
```
count    5454.000000
mean        2.832695
std         1.307299
min         0.810000
25%         1.640000
50%         2.680000
75%         3.527500
max         5.000000
Name: Pubs_Bars, dtype: float64
```

In [63]:
```
# displaying mode
read_df.Pubs_Bars.mode()
```

Out[63]:
```
0    5.0
dtype: float64
```

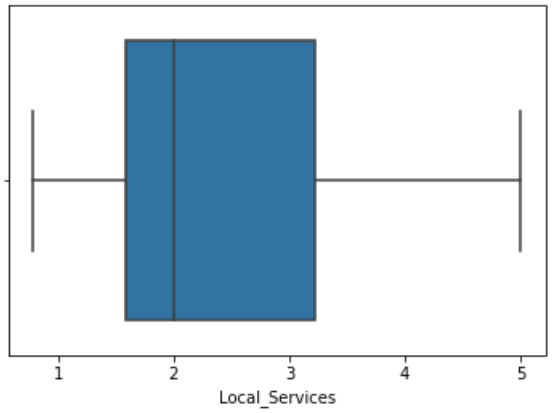
In [64]:
```
# plot density curve of Local_Services after cleaning null values
read_df.Local_Services.plot.density()
```

Out[64]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa7f29f98>
```



In [65]:
```
# plotting box plot of Local_Services
sns.boxplot(read_df['Local_Services'])
```

Out[65]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa7f8bcc0>
```



In [66]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Local_Services.describe()
```

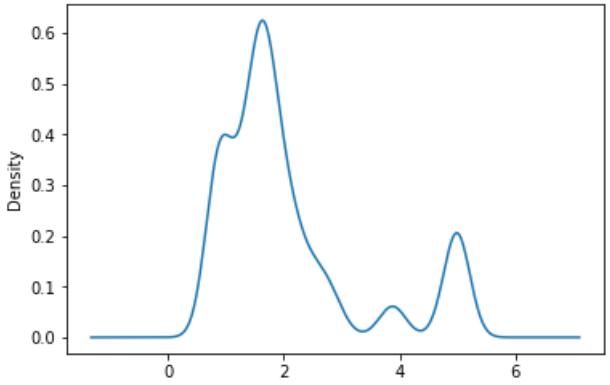
Out[66]:
```
count    5454.000000
mean        2.549622
std         1.381498
min         0.780000
25%         1.580000
50%         2.000000
75%         3.217500
max         5.000000
Name: Local_Services, dtype: float64
```

In [67]:
```
# displaying mode
read_df.Local_Services.mode()
```

Out[67]:
```
0    5.0
dtype: float64
```

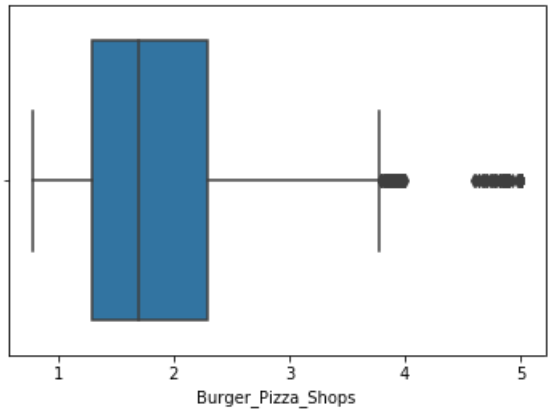
In [68]:
```
# plot density curve of Burger_Pizza_Shops after cleaning null values
read_df.Burger_Pizza_Shops.plot.density()
```

Out[68]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa7fe08d0>
```



In [69]:
```
# plotting box plot of Burger_Pizza_Shops
sns.boxplot(read_df['Burger_Pizza_Shops'])
```

Out[69]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa803ac88>
```



In [70]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Burger_Pizza_Shops.describe()
```

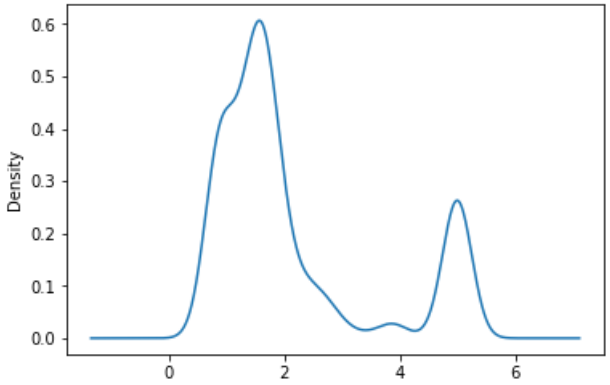
Out[70]:
```
count    5454.000000
mean        2.078401
std         1.249315
min         0.780000
25%         1.290000
50%         1.690000
75%         2.287500
max         5.000000
Name: Burger_Pizza_Shops, dtype: float64
```

In [71]:
```
# displaying mode
read_df.Burger_Pizza_Shops.mode()
```

Out[71]:
```
0    5.0
dtype: float64
```

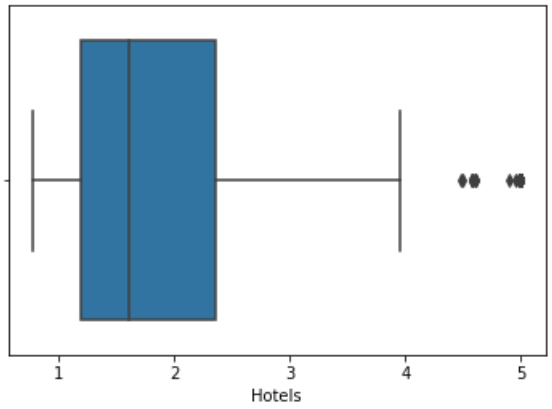
In [72]:
```
# plot density curve of Hotels after cleaning null values
read_df.Hotels.plot.density()
```

Out[72]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa807b390>
```



In [73]:
```
# plotting box plot of Hotels
sns.boxplot(read_df['Hotels'])
```

Out[73]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa80eab70>
```



In [74]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Hotels.describe()
```

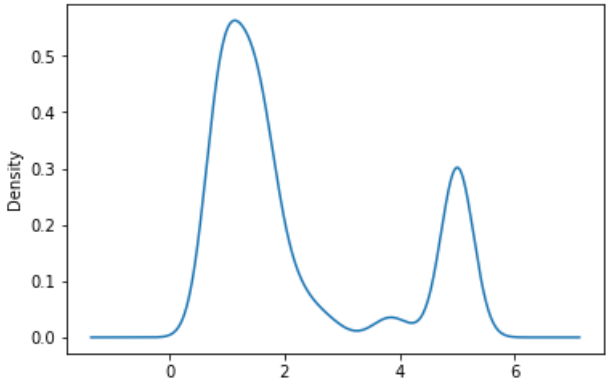
Out[74]:
```
count    5454.000000
mean        2.125820
std         1.406682
min         0.770000
25%         1.190000
50%         1.610000
75%         2.360000
max         5.000000
Name: Hotels, dtype: float64
```

In [75]:
```
# displaying mode
read_df.Hotels.mode()
```

Out[75]:
```
0    5.0
dtype: float64
```

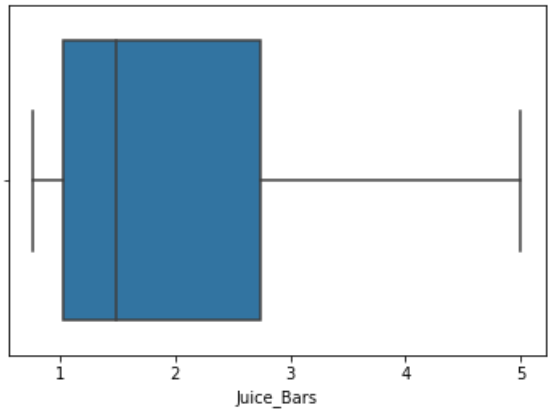
In [76]:
```
# plot density curve of Juice_Bars after cleaning null values
read_df.Juice_Bars.plot.density()
```

Out[76]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa8134278>
```



In [77]:
```
# plotting box plot of Juice_Bars
sns.boxplot(read_df['Juice_Bars'])
```

Out[77]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa8185ba8>
```



In [78]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Juice_Bars.describe()
```

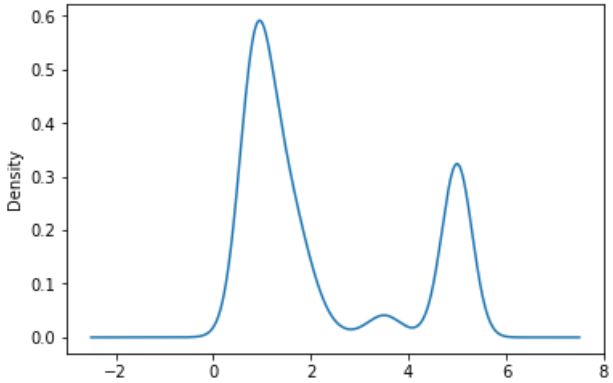
Out[78]:
```
count    5454.000000
mean        2.190429
std         1.576505
min         0.760000
25%         1.030000
50%         1.490000
75%         2.740000
max         5.000000
Name: Juice_Bars, dtype: float64
```

In [79]:
```
# displaying mode
read_df.Juice_Bars.mode()
```

Out[79]:
```
0    5.0
dtype: float64
```

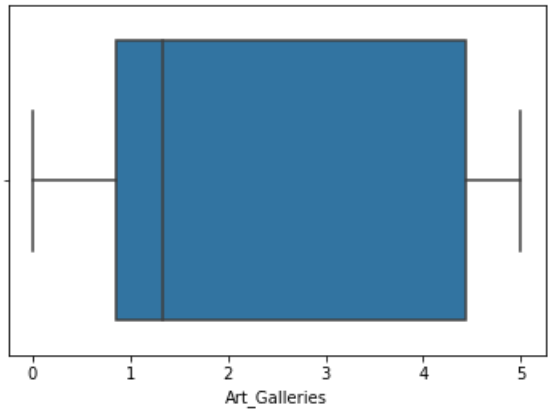
In [80]:
```
# plot density curve of Art_Galleries after cleaning null values
read_df.Art_Galleries.plot.density()
```

Out[80]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa81d5048>
```



In [81]:
```
# plotting box plot of Art_Galleries
sns.boxplot(read_df['Art_Galleries'])
```

Out[81]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa8239550>
```



In [82]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Art_Galleries.describe()
```

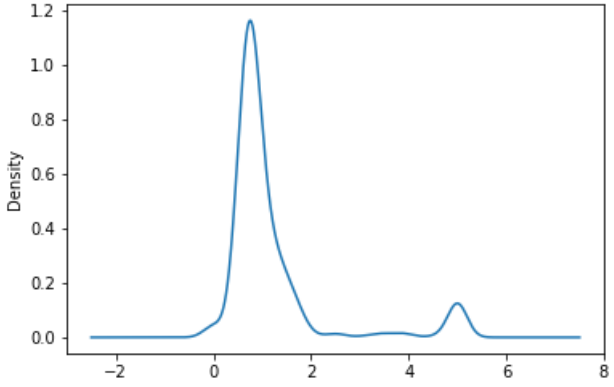
Out[82]:
```
count    5454.000000
mean        2.206140
std         1.715848
min         0.000000
25%         0.860000
50%         1.330000
75%         4.440000
max         5.000000
Name: Art_Galleries, dtype: float64
```

In [83]:
```
# displaying mode
read_df.Art_Galleries.mode()
```

Out[83]:
```
0    5.0
dtype: float64
```

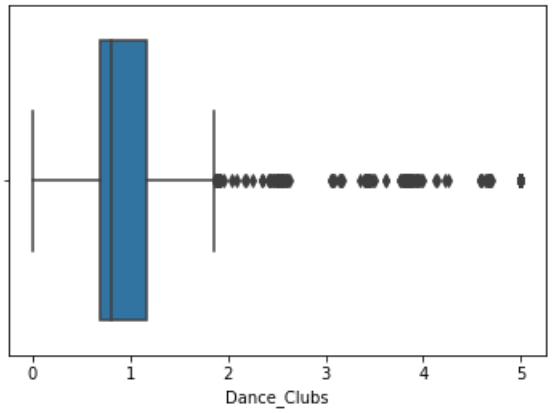
In [84]:
```
# plot density curve of Dance_Clubs after cleaning null values
read_df.Dance_Clubs.plot.density()
```

Out[84]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa828cb38>
```



In [85]:
```
# plotting box plot of Dance_Clubs
sns.boxplot(read_df['Dance_Clubs'])
```

Out[85]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa82f6710>
```



In [86]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Dance_Clubs.describe()
```

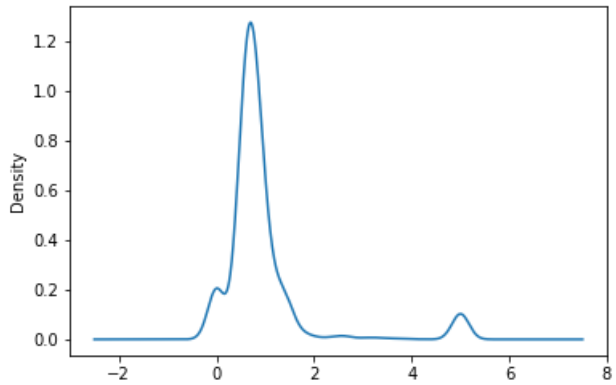
Out[86]:
```
count    5454.000000
mean        1.192710
std         1.107176
min         0.000000
25%         0.690000
50%         0.800000
75%         1.160000
max         5.000000
Name: Dance_Clubs, dtype: float64
```

In [87]:
```
# displaying mode
read_df.Dance_Clubs.mode()
```

Out[87]:
```
0    5.0
dtype: float64
```

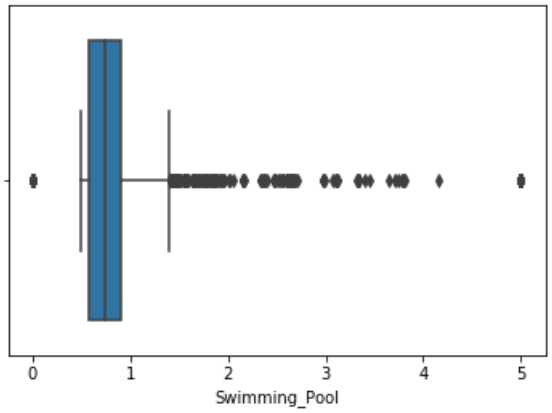
In [88]:
```
# plot density curve of Swimming_Pool after cleaning null values
read_df.Swimming_Pool.plot.density()
```

Out[88]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa9311a90>
```



In [89]:
```
# plotting box plot of Swimming_Pool
sns.boxplot(read_df['Swimming_Pool'])
```

Out[89]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa93726a0>
```



In [90]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Swimming_Pool.describe()
```

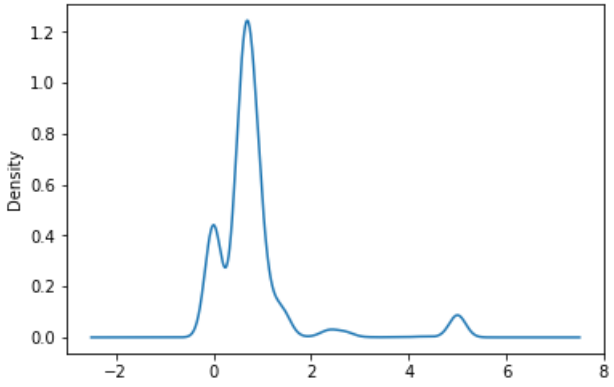
Out[90]:
```
count    5454.000000
mean        0.949349
std         0.973628
min         0.000000
25%         0.580000
50%         0.740000
75%         0.910000
max         5.000000
Name: Swimming_Pool, dtype: float64
```

In [91]:
```
# displaying mode
read_df.Swimming_Pool.mode()
```

Out[91]:
```
0    0.0
dtype: float64
```

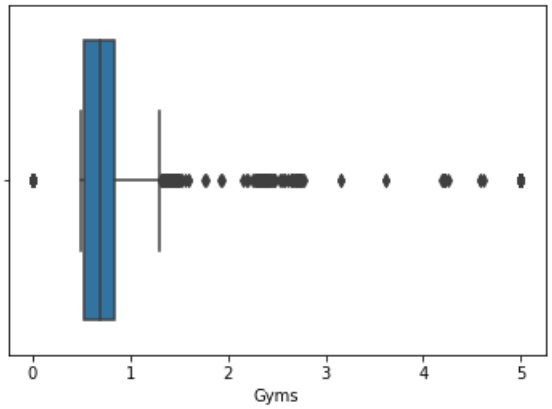
In [92]:
```
# plot density curve of Gyms after cleaning null values
read_df.Gyms.plot.density()
```

Out[92]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa93e1128>
```



In [93]:
```
# plotting box plot of Gyms
sns.boxplot(read_df['Gyms'])
```

Out[93]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa942ec50>
```



In [94]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Gyms.describe()
```

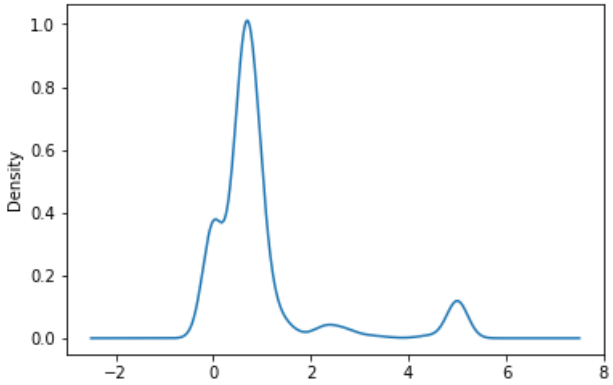
Out[94]:
```
count    5454.000000
mean        0.822525
std         0.948015
min         0.000000
25%         0.530000
50%         0.690000
75%         0.840000
max         5.000000
Name: Gyms, dtype: float64
```

In [95]:
```
# displaying mode
read_df.Gyms.mode()
```

Out[95]:
```
0    0.0
dtype: float64
```

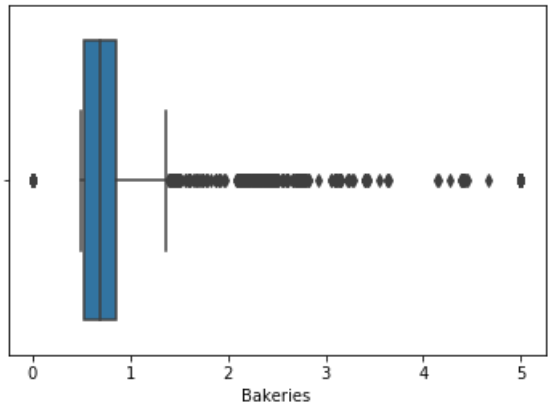
In [96]:
```
# plot density curve of Bakeries after cleaning null values
read_df.Bakeries.plot.density()
```

Out[96]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa948d6d8>
```



In [97]:
```
# plotting box plot of Bakeries
sns.boxplot(read_df['Bakeries'])
```

Out[97]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa94dbfd0>
```



In [98]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Bakeries.describe()
```

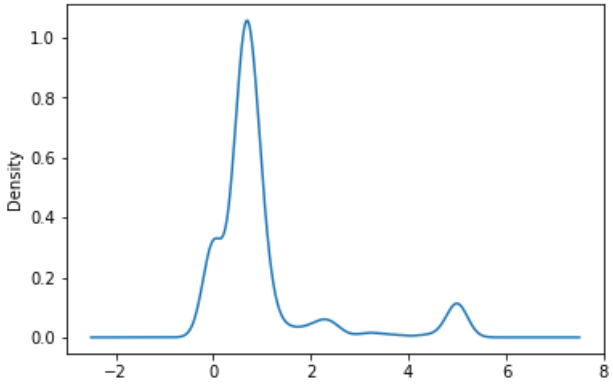
Out[98]:
```
count    5454.000000
mean        0.969250
std         1.202883
min         0.000000
25%         0.520000
50%         0.690000
75%         0.860000
max         5.000000
Name: Bakeries, dtype: float64
```

In [100]:
```
# displaying mode
read_df.Bakeries.mode()
```

Out[100]:
```
0    0.0
dtype: float64
```

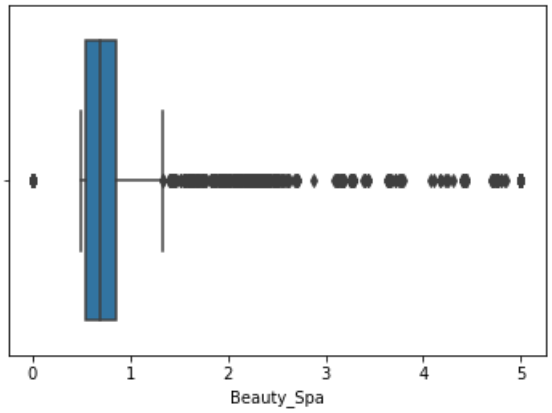
In [101]:
```
# plot density curve of Beauty_Spa after cleaning null values
read_df.Beauty_Spa.plot.density()
```

Out[101]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa952aeb8>
```



In [102]:
```
# plotting box plot of Beauty_Spa
sns.boxplot(read_df['Beauty_Spa'])
```

Out[102]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa9598080>
```



In [103]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Beauty_Spa.describe()
```

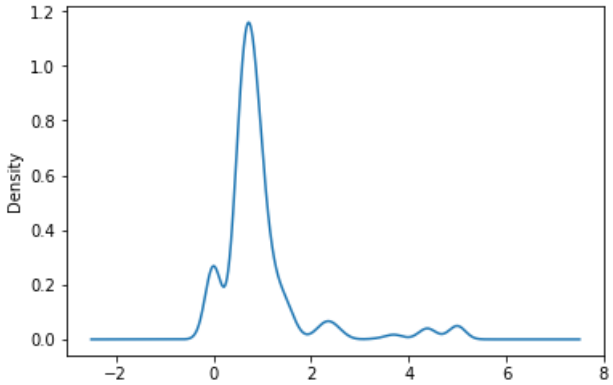
Out[103]:
```
count    5454.000000
mean        0.999626
std         1.193129
min         0.000000
25%         0.540000
50%         0.690000
75%         0.860000
max         5.000000
Name: Beauty_Spa, dtype: float64
```

In [104]:
```
# displaying mode
read_df.Beauty_Spa.mode()
```

Out[104]:
```
0    0.0
dtype: float64
```

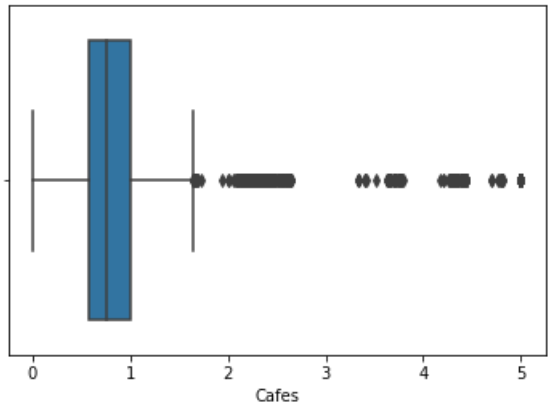
In [105]:
```
# plot density curve of Cafes after cleaning null values
read_df.Cafes.plot.density()
```

Out[105]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa95d74e0>
```



In [106]:
```
# plotting box plot of Cafes
sns.boxplot(read_df['Cafes'])
```

Out[106]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa963a668>
```



In [107]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Cafes.describe()
```

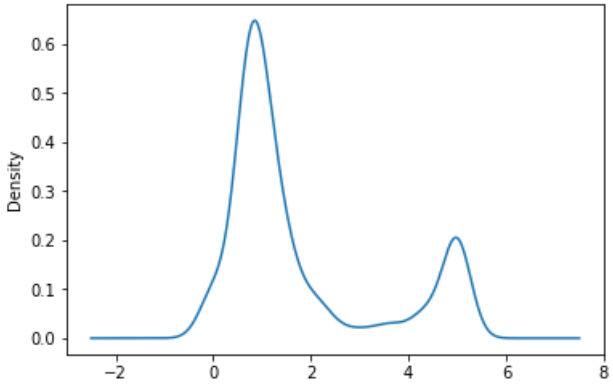
Out[107]:
```
count    5454.000000
mean        0.965275
std         0.928326
min         0.000000
25%         0.570000
50%         0.760000
75%         1.000000
max         5.000000
Name: Cafes, dtype: float64
```

In [108]:
```
# displaying mode
read_df.Cafes.mode()
```

Out[108]:
```
0    0.0
dtype: float64
```

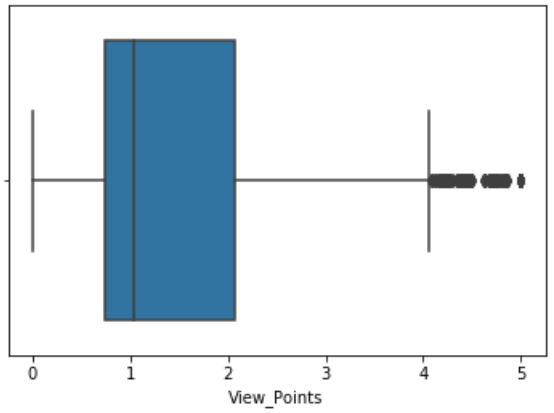
In [109]:
```
# plot density curve of View_Points after cleaning null values
read_df.View_Points.plot.density()
```

Out[109]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa967eb38>
```



In [110]:
```
# plotting box plot of View_Points
sns.boxplot(read_df['View_Points'])
```

Out[110]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa96f5da0>
```



In [111]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.View_Points.describe()
```

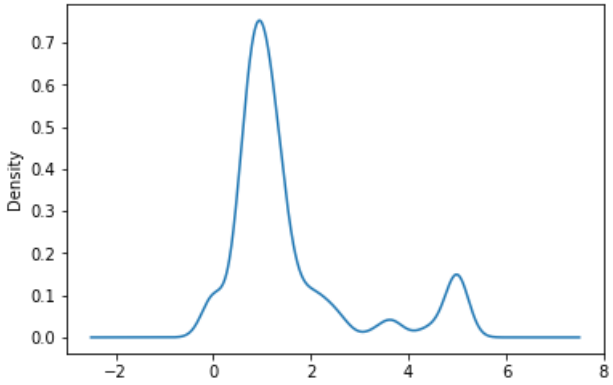
Out[111]:
```
count    5454.000000
mean        1.749345
std         1.597816
min         0.000000
25%         0.740000
50%         1.030000
75%         2.070000
max         5.000000
Name: View_Points, dtype: float64
```

In [112]:
```
# displaying mode
read_df.View_Points.mode()
```

Out[112]:
```
0    5.0
dtype: float64
```

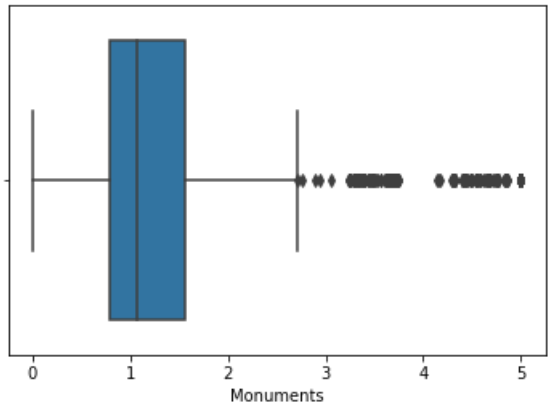
In [113]:
```
# plot density curve of Monuments after cleaning null values
read_df.Monuments.plot.density()
```

Out[113]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa9743b70>
```



In [114]:
```
# plotting box plot of Monuments
sns.boxplot(read_df['Monuments'])
```

Out[114]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa9687cc0>
```



In [115]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Monuments.describe()
```

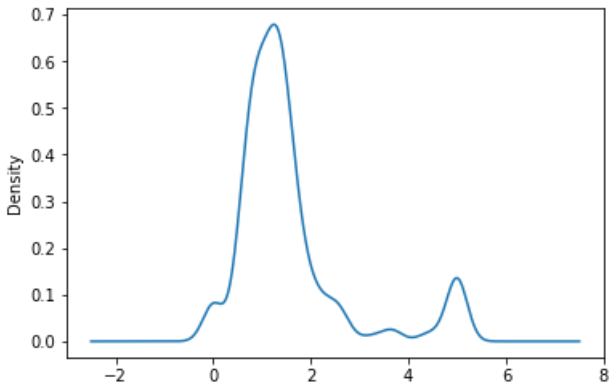
Out[115]:
```
count    5454.000000
mean        1.531051
std         1.316180
min         0.000000
25%         0.790000
50%         1.070000
75%         1.560000
max         5.000000
Name: Monuments, dtype: float64
```

In [116]:
```
# displaying mode
read_df.Monuments.mode()
```

Out[116]:
```
0    5.0
dtype: float64
```

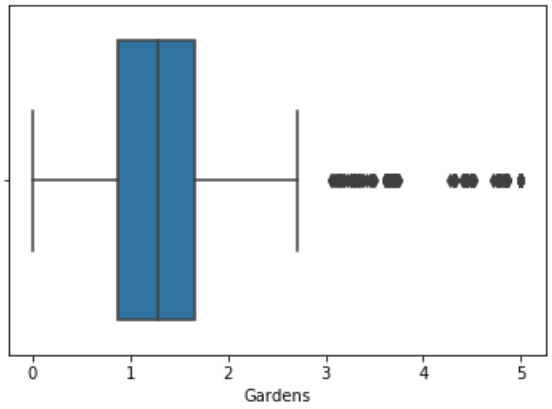
In [117]:
```
# plot density curve of Gardens after cleaning null values
read_df.Gardens.plot.density()
```

Out[117]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa97e9c88>
```



In [118]:
```
# plotting box plot of Gardens
sns.boxplot(read_df['Gardens'])
```

Out[118]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aa9868d68>
```



In [119]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
read_df.Gardens.describe()
```

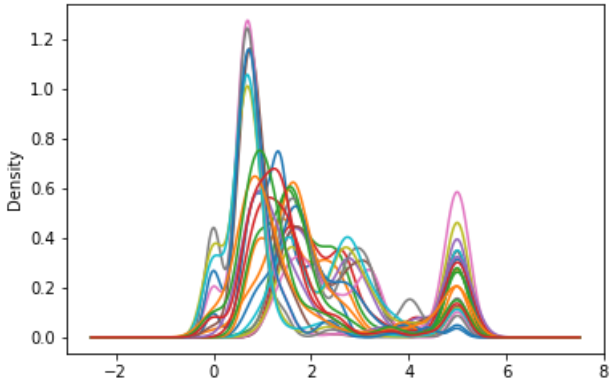
Out[119]:
```
count    5454.000000
mean        1.560570
std         1.171784
min         0.000000
25%         0.880000
50%         1.290000
75%         1.660000
max         5.000000
Name: Gardens, dtype: float64
```

In [120]:
```
# displaying mode
read_df.Gardens.mode()
```

Out[120]:
```
0    5.0
dtype: float64
```

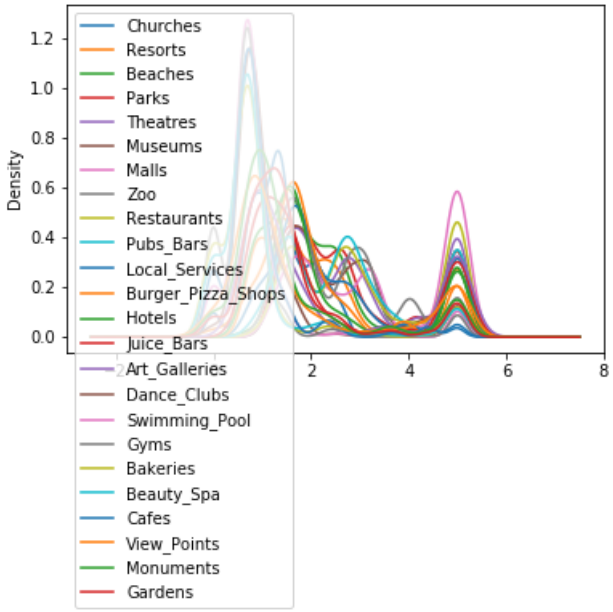
In [121]:
```
read_df.Churches.plot.density()
read_df.Resorts.plot.density()
read_df.Beaches.plot.density()
read_df.Parks.plot.density()
read_df.Theatres.plot.density()
read_df.Museums.plot.density()
read_df.Malls.plot.density()
read_df.Zoo.plot.density()
read_df.Restaurants.plot.density()
read_df.Pubs_Bars.plot.density()
read_df.Local_Services.plot.density()
read_df.Burger_Pizza_Shops.plot.density()
read_df.Hotels.plot.density()
read_df.Juice_Bars.plot.density()
read_df.Art_Galleries.plot.density()
read_df.Dance_Clubs.plot.density()
read_df.Swimming_Pool.plot.density()
read_df.Gyms.plot.density()
read_df.Bakeries.plot.density()
read_df.Beauty_Spa.plot.density()
read_df.Cafes.plot.density()
read_df.View_Points.plot.density()
read_df.Monuments.plot.density()
read_df.Gardens.plot.density()
```

Out[121]: <matplotlib.axes._subplots.AxesSubplot at 0x17aa98cc320>



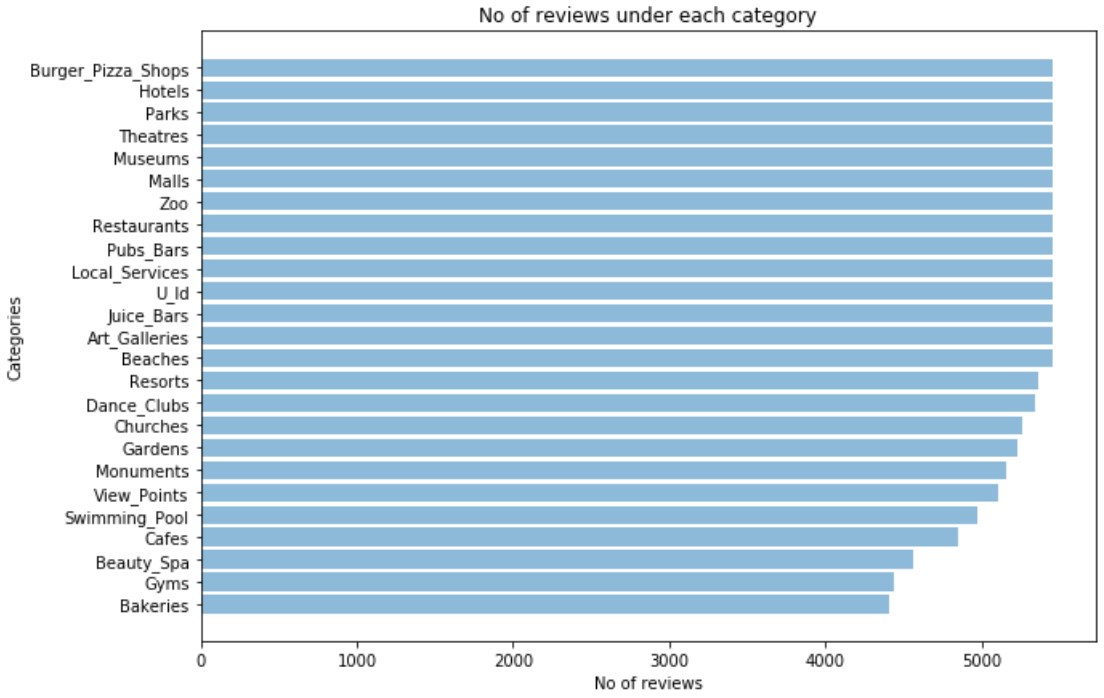
In [122]:
```
read_df.plot.density()
```

Out[122]: <matplotlib.axes._subplots.AxesSubplot at 0x17aa991d898>



In [123]:
```
# number of user have given ratings to each category; Gyms and Bakeries having least ratings
no_of_zeros = read_df[column_names[:]].astype(bool).sum(axis=0).sort_values()


plt.figure(figsize=(10,7))

plt.barh(np.arange(len(column_names[:])), no_of_zeros.values, align='center', alpha=0.5)

plt.yticks(np.arange(len(column_names[:])), no_of_zeros.index)

plt.xlabel('No of reviews')

plt.ylabel('Categories')

plt.title('No of reviews under each category')
```

Out[123]: Text(0.5, 1.0, 'No of reviews under each category')

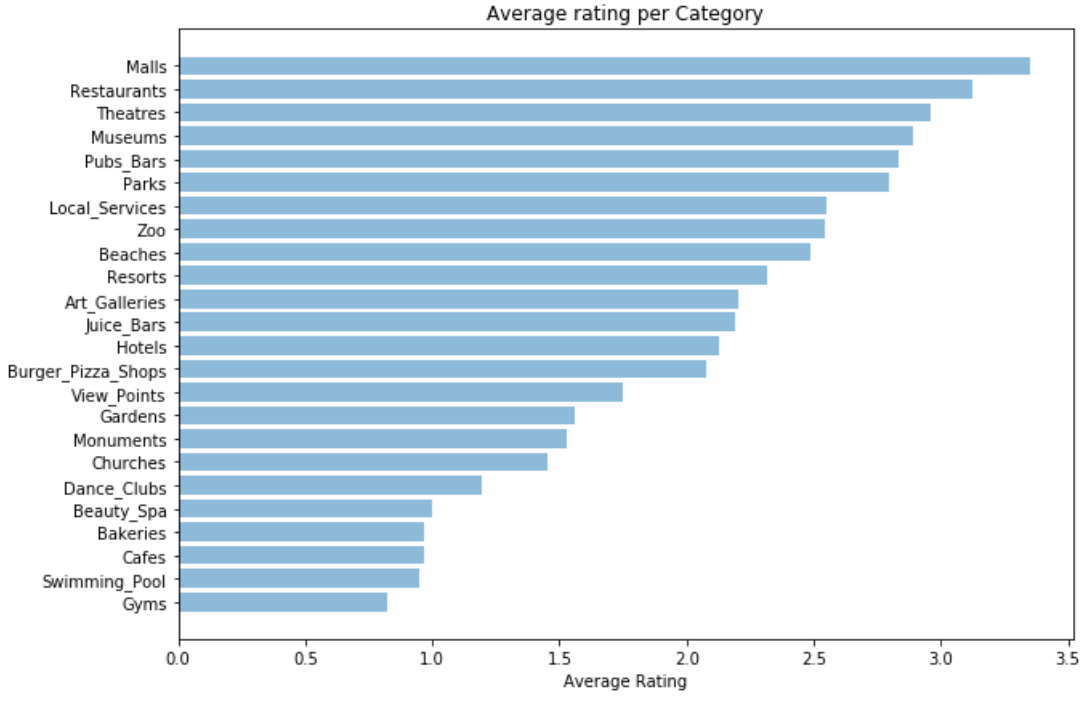


In [124]:
```
avg_rating = read_df[column_names[1:]].mean()
avg_rating = avg_rating.sort_values()
avg_rating
```

Out[124]:
```
Gyms                 0.822525
Swimming_Pool        0.949349
Cafes                0.965275
Bakeries             0.969250
Beauty_Spa           0.999626
Dance_Clubs          1.192710
Churches             1.455746
Monuments            1.531051
Gardens              1.560570
View_Points          1.749345
Burger_Pizza_Shops   2.078401
Hotels               2.125820
Juice_Bars           2.190429
Art_Galleries        2.206140
Resorts              2.320048
Beaches              2.489059
Zoo                  2.541177
Local_Services       2.549622
Parks                2.797103
Pubs_Bars            2.832695
Museums              2.893423
Theatres             2.958904
Restaurants          3.126542
Malls                3.351476
dtype: float64
```

In [125]:
```
plt.figure(figsize=(10,7))
plt.barh(np.arange(len(column_names[1:])), avg_rating.values, align='center', alpha=0.5)
plt.yticks(np.arange(len(column_names[1:])), avg_rating.index)
plt.xlabel('Average Rating')
plt.title('Average rating per Category')
```

Out[125]:
```
Text(0.5, 1.0, 'Average rating per Category')
```



In [139]:
```
Entertainment = ['Theatres', 'Dance_Clubs', 'Malls']
Food_Travel = ['Restaurants', 'Pubs_Bars', 'Burger_Pizza_Shops', 'Juice_Bars', 'Bakeries', 'Cafes']
Places_Of_Stay = ['Hotels', 'Resorts']
Historical = ['Churches', 'Museums', 'Art_Galleries', 'Monuments']
Nature = ['Beaches', 'Parks', 'Zoo', 'View_Points', 'Gardens']
Services = ['Local_Services', 'Swimming_Pool', 'Gyms', 'Beauty_Spa']
```

In [140]:
```
df_category_reviews = pd.DataFrame(columns = ['Entertainment', 'Food_Travel', 'Places_Of_Stay', 'Hist
```

In [141]:
```
df_category_reviews.head()
```

Out[141]:

	Entertainment	Food_Travel	Places_Of_Stay	Historical	Nature	Services

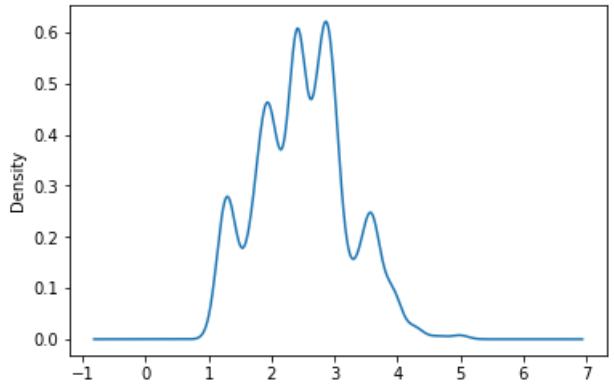
In [142]:
```
df_category_reviews['Entertainment'] = read_df[Entertainment].mean(axis = 1)
df_category_reviews['Food_Travel'] = read_df[Food_Travel].mean(axis = 1)
df_category_reviews['Places_Of_Stay'] = read_df[Places_Of_Stay].mean(axis = 1)
df_category_reviews['Historical'] = read_df[Historical].mean(axis = 1)
df_category_reviews['Nature'] = read_df[Nature].mean(axis = 1)
df_category_reviews['Services'] = read_df[Services].mean(axis = 1)
```

In [143]:
```
df_category_reviews['Entertainment']
```

Out[143]:
```
0       3.530000
1       3.530000
2       3.530000
3       3.530000
4       3.530000
5       3.530000
6       2.873333
7       3.533333
8       2.876667
9       3.530000
10      3.530000
11      3.526667
12      3.526667
13      3.540000
14      2.826667
15      2.826667
16      2.150000
17      2.143333
18      2.166667
19      2.160000
20      2.176667
21      2.166667
22      2.170000
23      2.193333
24      2.200000
25      2.193333
26      2.193333
27      2.193333
28      2.153333
29      2.156667
          ...
5426    2.006667
5427    1.986667
5428    1.986667
5429    2.013333
5430    1.986667
5431    1.980000
5432    1.983333
5433    1.983333
5434    1.976667
5435    1.980000
5436    1.976667
5437    1.973333
5438    1.980000
5439    1.973333
5440    1.973333
5441    1.970000
5442    1.966667
5443    1.963333
5444    1.966667
5445    1.963333
5446    1.963333
5447    1.960000
5448    1.960000
5449    1.956667
5450    1.956667
5451    1.953333
5452    1.733333
5453    1.726667
5454    1.730000
5455    1.953333
Name: Entertainment, Length: 5454, dtype: float64
```

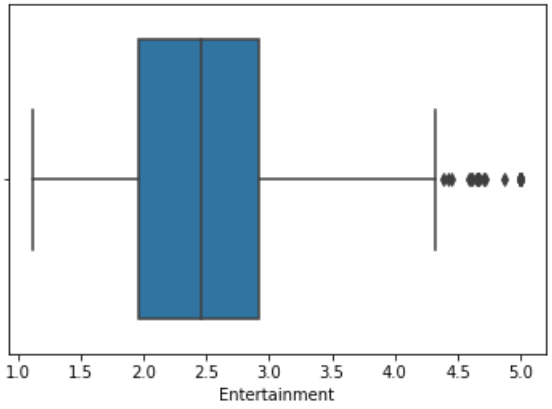
In [144]:
```
# plot density curve of Entertainment
df_category_reviews.Entertainment.plot.density()
```

Out[144]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aaabe5e10>
```



In [145]:
```
# plotting box plot of Entertainment
sns.boxplot(df_category_reviews['Entertainment'])
```

Out[145]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aaac33a58>
```



In [146]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
df_category_reviews.Entertainment.describe()
```

Out[146]:
```
count    5454.000000
mean        2.501030
std         0.722052
min         1.120000
25%         1.964167
50%         2.453333
75%         2.916667
max         5.000000
Name: Entertainment, dtype: float64
```

In [147]:
```
# displaying mode
df_category_reviews.Entertainment.mode()
```

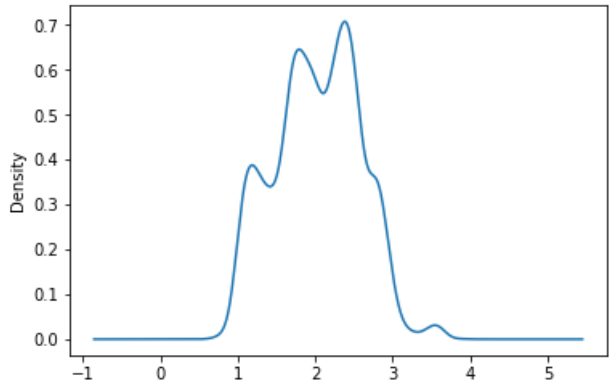
Out[147]:
```
0    2.373333
dtype: float64
```

In [148]:
```
df_category_reviews['Food_Travel']
```

Out[148]:
```
0       1.480000
1       1.481667
2       1.480000
3       1.480000
4       1.480000
5       1.481667
6       1.476667
7       1.476667
8       1.470000
9       1.386667
10      1.383333
11      1.376667
12      1.306667
13      1.701667
14      1.306667
15      1.258333
16      1.205000
17      1.205000
18      1.205000
19      1.205000
20      1.205000
21      1.153333
22      1.153333
23      1.155000
24      1.221667
25      1.220000
26      1.221667
27      1.220000
28      1.218333
29      1.218333
          ...
5426    1.748333
5427    1.415000
5428    1.176667
5429    1.468333
5430    1.468333
5431    1.251667
5432    1.911667
5433    2.603333
5434    2.595000
5435    1.938333
5436    2.575000
5437    2.565000
5438    1.181667
5439    1.171667
5440    1.963333
5441    1.911667
5442    1.851667
5443    2.468333
5444    2.461667
5445    2.498333
5446    2.443333
5447    1.726667
5448    1.083333
5449    1.186667
5450    1.310000
5451    1.150000
5452    1.290000
5453    1.135000
5454    1.126667
5455    1.121667
Name: Food_Travel, Length: 5454, dtype: float64
```

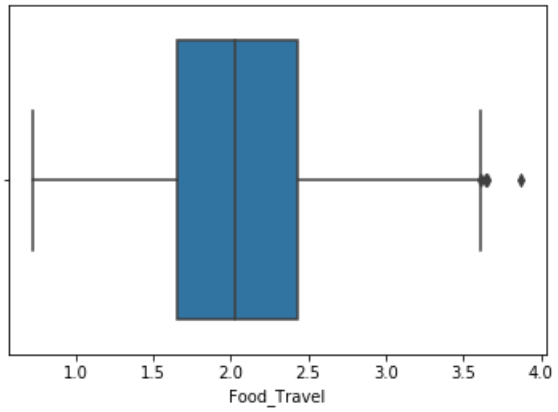
In [149]:
```
# plot density curve of Food_Travel
df_category_reviews.Food_Travel.plot.density()
```

Out[149]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aaaca2240>
```



In [150]:
```
# plotting box plot of Food_Travel
sns.boxplot(df_category_reviews['Food_Travel'])
```

Out[150]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aaacfcba8>
```



In [151]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
df_category_reviews.Food_Travel.describe()
```

Out[151]:
```
count    5454.000000
mean        2.027099
std         0.549316
min         0.721667
25%         1.650000
50%         2.027500
75%         2.433333
max         3.873333
Name: Food_Travel, dtype: float64
```

In [152]:
```
# displaying mode
df_category_reviews.Food_Travel.mode()
```

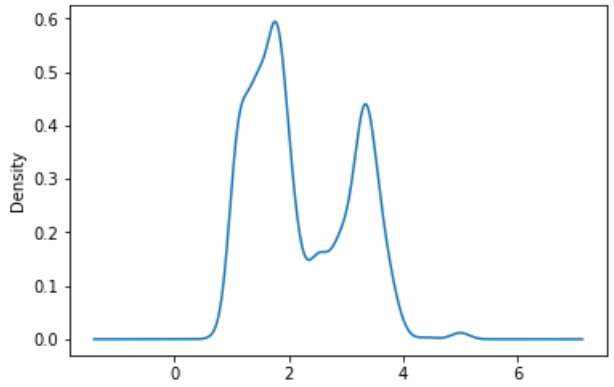
Out[152]:
```
0    2.343333
dtype: float64
```

In [153]:
```
df_category_reviews['Places_Of_Stay']
```

Out[153]:
```
0       0.850
1       0.850
2       0.850
3       1.100
4       0.850
5       0.845
6       3.345
7       3.345
8       3.340
9       3.335
10      1.100
11      1.095
12      1.095
13      1.095
14      1.090
15      1.080
16      1.085
17      1.090
18      1.085
19      3.325
20      1.080
21      3.320
22      3.320
23      1.075
24      1.075
25      1.080
26      1.095
27      1.075
28      1.085
29      1.085
        ...
5426    2.625
5427    2.580
5428    2.570
5429    2.565
5430    2.565
5431    2.565
5432    2.565
5433    3.235
5434    2.560
5435    3.220
5436    2.570
5437    2.565
5438    2.565
5439    3.190
5440    2.125
5441    3.230
5442    2.625
5443    3.215
5444    2.560
5445    3.200
5446    2.625
5447    2.620
5448    3.170
5449    2.625
5450    2.620
5451    3.145
5452    3.135
5453    3.040
5454    2.555
5455    2.540
Name: Places_Of_Stay, Length: 5454, dtype: float64
```

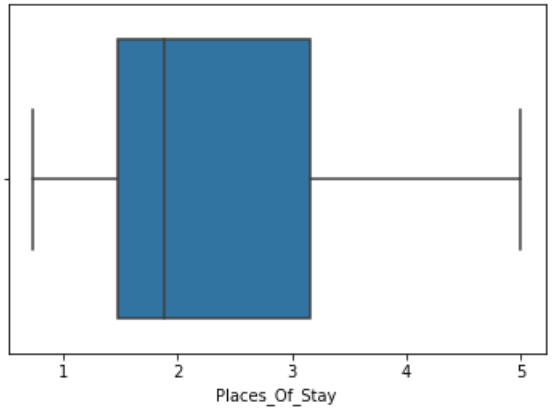
In [154]:
```
# plot density curve of Places_Of_Stay
df_category_reviews.Places_Of_Stay.plot.density()
```

Out[154]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aaae83eb8>
```



In [155]:
```
# plotting box plot of Places_Of_Stay
sns.boxplot(df_category_reviews['Places_Of_Stay'])
```

Out[155]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aaaef3b70>
```



In [156]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
df_category_reviews.Places_Of_Stay.describe()
```

Out[156]:
```
count    5454.000000
mean        2.222934
std         0.886587
min         0.730000
25%         1.470000
50%         1.885000
75%         3.160000
max         5.000000
Name: Places_Of_Stay, dtype: float64
```

In [157]:
```
# displaying mode
df_category_reviews.Places_Of_Stay.mode()
```

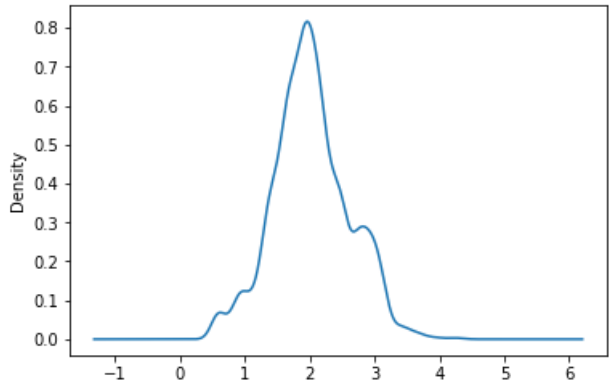
Out[157]:
```
0    1.71
dtype: float64
```

In [158]:
```
df_category_reviews['Historical']
```

Out[158]:
```
0       1.1650
1       1.1650
2       1.1650
3       1.1650
4       1.1650
5       1.1650
6       1.1675
7       1.1650
8       0.9175
9       0.9150
10      0.9150
11      0.9150
12      0.9125
13      0.9225
14      0.9225
15      0.9225
16      0.9225
17      0.9225
18      0.9250
19      0.9350
20      0.9350
21      0.9350
22      0.9600
23      1.1575
24      1.1600
25      1.1750
26      1.1750
27      1.1750
28      1.1750
29      1.1750
         ...
5426    2.1775
5427    2.1850
5428    3.1275
5429    3.9100
5430    2.3700
5431    2.8425
5432    2.6600
5433    2.1800
5434    2.1875
5435    2.3600
5436    2.6875
5437    2.6900
5438    3.7725
5439    2.7050
5440    2.7125
5441    2.7300
5442    2.6725
5443    4.3075
5444    4.2200
5445    4.2200
5446    3.4125
5447    4.2300
5448    4.3225
5449    3.4975
5450    2.4525
5451    3.3700
5452    2.3475
5453    2.3450
5454    3.3475
5455    2.3425
Name: Historical, Length: 5454, dtype: float64
```

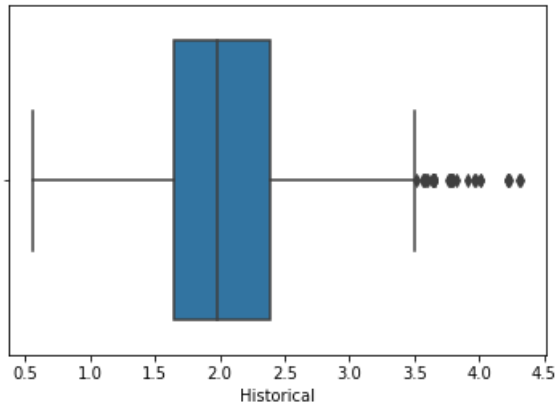
In [159]:
```
# plot density curve of Historical
df_category_reviews.Historical.plot.density()
```

Out[159]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aaaf40710>
```



In [160]:
```
# plotting box plot of Historical
sns.boxplot(df_category_reviews['Historical'])
```

Out[160]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aaafa0da0>
```



In [161]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
df_category_reviews.Historical.describe()
```

Out[161]:
```
count    5454.000000
mean        2.021590
std         0.584872
min         0.557500
25%         1.647500
50%         1.977500
75%         2.392500
max         4.322500
Name: Historical, dtype: float64
```

In [162]:
```
# displaying mode
df_category_reviews.Historical.mode()
```

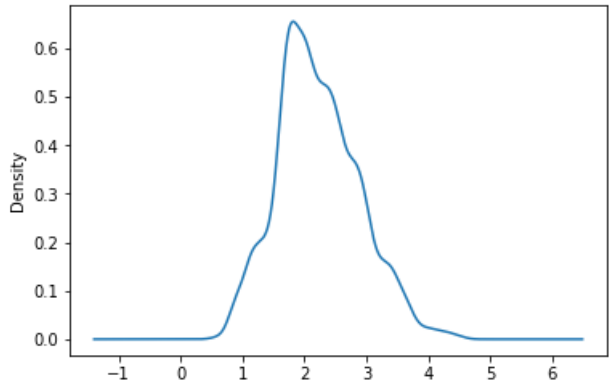
Out[162]:
```
0    2.215
dtype: float64
```

In [163]:
```
df_category_reviews['Nature']
```

Out[163]:
```
0       1.926
1       1.984
2       1.980
3       1.922
4       1.980
5       1.978
6       1.922
7       1.978
8       1.980
9       1.926
10      1.986
11      1.988
12      1.990
13      1.930
14      1.932
15      2.066
16      2.064
17      2.070
18      2.330
19      2.070
20      2.330
21      2.332
22      2.332
23      2.326
24      1.362
25      1.428
26      1.362
27      2.260
28      1.364
29      1.364
        ...
5426    4.282
5427    2.958
5428    2.696
5429    2.696
5430    2.696
5431    2.696
5432    2.960
5433    2.962
5434    2.706
5435    2.714
5436    2.436
5437    2.506
5438    2.510
5439    2.516
5440    2.730
5441    2.520
5442    2.524
5443    2.524
5444    2.748
5445    2.882
5446    2.762
5447    3.568
5448    2.772
5449    2.890
5450    3.574
5451    2.888
5452    2.794
5453    2.798
5454    2.802
5455    3.002
Name: Nature, Length: 5454, dtype: float64
```

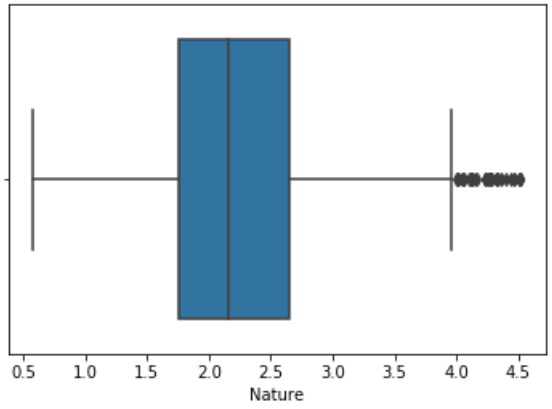
In [164]:
```
# plot density curve of Nature
df_category_reviews.Nature.plot.density()
```

Out[164]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aab146a58>
```



In [165]:
```
# plotting box plot of Nature
sns.boxplot(df_category_reviews['Nature'])
```

Out[165]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aab1afa90>
```



In [166]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
df_category_reviews.Nature.describe()
```

Out[166]:
```
count    5454.000000
mean        2.227451
std         0.662565
min         0.576000
25%         1.762000
50%         2.160000
75%         2.656000
max         4.520000
Name: Nature, dtype: float64
```

In [167]:
```
# displaying mode
df_category_reviews.Nature.mode()
```

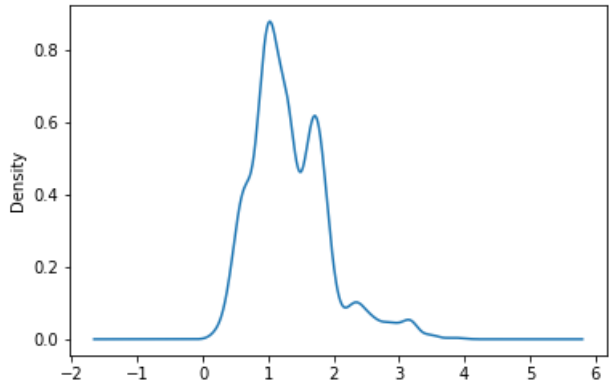
Out[167]:
```
0    1.832
dtype: float64
```

In [168]:
```
df_category_reviews['Services']
```

Out[168]:
```
0       0.5500
1       0.5500
2       0.5500
3       0.5575
4       0.5500
5       0.5525
6       0.5575
7       0.5500
8       0.4275
9       0.4225
10      0.4175
11      0.4175
12      0.4175
13      0.4125
14      0.5400
15      0.5400
16      0.5400
17      0.5425
18      0.5450
19      0.5400
20      0.5425
21      0.5575
22      0.5375
23      0.4125
24      0.6825
25      0.5475
26      0.6925
27      0.6775
28      0.6800
29      0.6800
         ...
5426    0.9475
5427    0.9475
5428    0.9350
5429    0.9450
5430    0.9450
5431    0.9425
5432    0.9425
5433    0.9375
5434    0.9325
5435    0.9325
5436    0.9225
5437    0.9175
5438    0.9150
5439    0.9075
5440    0.9100
5441    0.8575
5442    1.9025
5443    0.8950
5444    1.8900
5445    1.8800
5446    1.8725
5447    1.8675
5448    1.8575
5449    1.8475
5450    1.8425
5451    1.8325
5452    0.9800
5453    1.8150
5454    1.8100
5455    1.8000
Name: Services, Length: 5454, dtype: float64
```

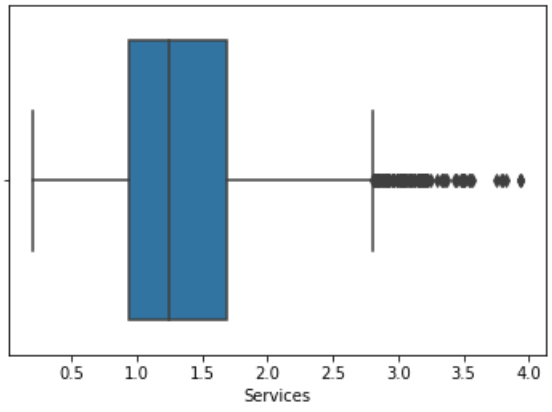
In [169]:
```
# plot density curve of Services
df_category_reviews.Services.plot.density()
```

Out[169]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aab208cc0>
```



In [170]:
```
# plotting box plot of Services
sns.boxplot(df_category_reviews['Services'])
```

Out[170]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17aab2080f0>
```



In [171]:
```
# displaying no. of rows(count), mean, median(50%), standard deviation, min, max values
df_category_reviews.Services.describe()
```

Out[171]:
```
count    5454.000000
mean        1.330281
std         0.580751
min         0.205000
25%         0.937500
50%         1.245000
75%         1.685000
max         3.937500
Name: Services, dtype: float64
```

In [172]:
```
# displaying mode
df_category_reviews.Services.mode()
```

Out[172]:
```
0    1.25
dtype: float64
```

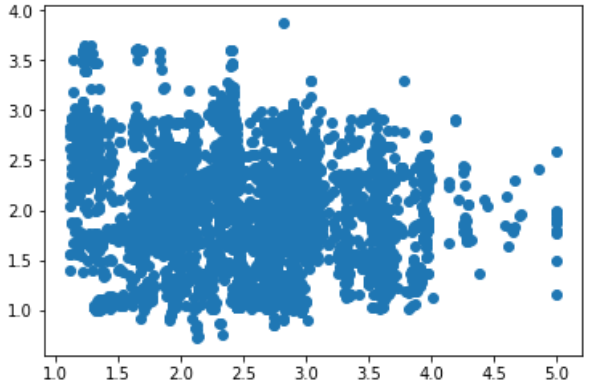
In [173]:
```
df_category_reviews.describe()
```

Out[173]:

	Entertainment	Food_Travel	Places_Of_Stay	Historical	Nature	Services
count	5454.000000	5454.000000	5454.000000	5454.000000	5454.000000	5454.000000
mean	2.501030	2.027099	2.222934	2.021590	2.227451	1.330281
std	0.722052	0.549316	0.886587	0.584872	0.662565	0.580751
min	1.120000	0.721667	0.730000	0.557500	0.576000	0.205000
25%	1.964167	1.650000	1.470000	1.647500	1.762000	0.937500
50%	2.453333	2.027500	1.885000	1.977500	2.160000	1.245000
75%	2.916667	2.433333	3.160000	2.392500	2.656000	1.685000
max	5.000000	3.873333	5.000000	4.322500	4.520000	3.937500

In [174]:
```
X=np.array(df_category_reviews)
plt.scatter(X[:,0],X[:,1], label='True Position')     # X contains row and column- row all and column
```

Out[174]:
```
<matplotlib.collections.PathCollection at 0x17aab2f73c8>
```



In [176]:
```
kmeans = KMeans(n_clusters=6)
kmeans.fit(X)
print(kmeans.cluster_centers_)
```

```
[[2.80505784 2.03656578 3.41617837 2.15934221 2.49661663 1.13696731]
 [1.87833997 2.65907481 2.97482736 1.61199203 1.43502523 1.52372178]
 [2.25594001 1.42673215 1.69634981 2.19659696 3.06356907 1.18546261]
 [3.26075623 1.88447657 1.55697509 2.22949066 2.13619573 1.02879671]
 [2.03941667 1.94501786 2.01819643 2.27545536 2.13846786 2.33957143]
 [2.18772497 2.22159252 1.44847321 1.64123357 1.98064105 1.31185794]]
```

In [177]:
```
y_predict=kmeans.fit_predict(df_category_reviews[['Entertainment', 'Food_Travel', 'Places_Of_Stay', '
y_predict
```

Out[177]:
```
array([5, 5, 5, ..., 0, 0, 1])
```

In [178]:
```
df_category_reviews['cluster']=y_predict
df_category_reviews.head(20)
#df_category_reviews.drop('Avg_Ratings',axis=1)
```

Out[178]:

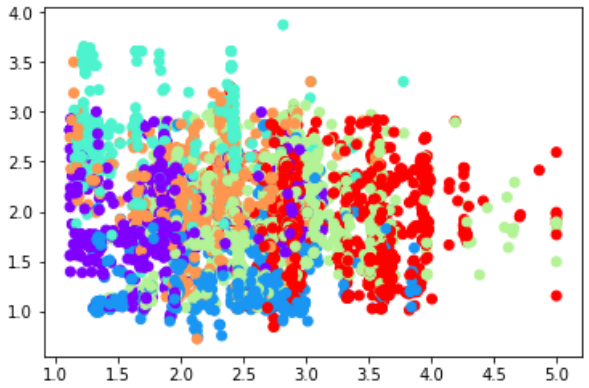
	Entertainment	Food_Travel	Places_Of_Stay	Historical	Nature	Services	cluster
0	3.530000	1.480000	0.850	1.1650	1.926	0.5500	5
1	3.530000	1.481667	0.850	1.1650	1.984	0.5500	5
2	3.530000	1.480000	0.850	1.1650	1.980	0.5500	5
3	3.530000	1.480000	1.100	1.1650	1.922	0.5575	5
4	3.530000	1.480000	0.850	1.1650	1.980	0.5500	5
5	3.530000	1.481667	0.845	1.1650	1.978	0.5525	5
6	2.873333	1.476667	3.345	1.1675	1.922	0.5575	3
7	3.533333	1.476667	3.345	1.1650	1.978	0.5500	3
8	2.876667	1.470000	3.340	0.9175	1.980	0.4275	3
9	3.530000	1.386667	3.335	0.9150	1.926	0.4225	3
10	3.530000	1.383333	1.100	0.9150	1.986	0.4175	5
11	3.526667	1.376667	1.095	0.9150	1.988	0.4175	5
12	3.526667	1.306667	1.095	0.9125	1.990	0.4175	5
13	3.540000	1.701667	1.095	0.9225	1.930	0.4125	5
14	2.826667	1.306667	1.090	0.9225	1.932	0.5400	4
15	2.826667	1.258333	1.080	0.9225	2.066	0.5400	4
16	2.150000	1.205000	1.085	0.9225	2.064	0.5400	4
17	2.143333	1.205000	1.090	0.9225	2.070	0.5425	4
18	2.166667	1.205000	1.085	0.9250	2.330	0.5450	4
19	2.160000	1.205000	3.325	0.9350	2.070	0.5400	3

In [179]:
```
print(kmeans.labels_)
```

```
[5 5 5 ... 0 0 1]
```

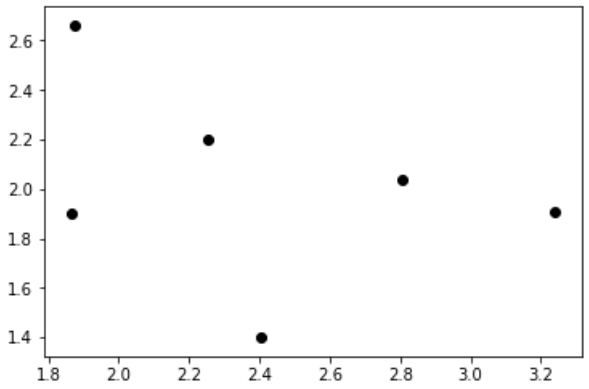
In [180]:
```
plt.scatter(X[:,0],X[:,1], c=kmeans.labels_, cmap='rainbow')
```

Out[180]:
```
<matplotlib.collections.PathCollection at 0x17aab366710>
```



In [181]:
```
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],color='black')
```

Out[181]:
```
<matplotlib.collections.PathCollection at 0x17aab7dc7b8>
```



In [182]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
y=df_category_reviews.cluster
y.head()
x=df_category_reviews.iloc[:,0:6]
x.head()
x_train, x_test, y_train, y_test= train_test_split(x,y,test_size=0.2,random_state=0)
#trainset, testset = train_test_split(df_category_reviews, test_size=0.25, random_state = 0)
```

In [183]:
```
x_train.head()
```

Out[183]:

	Entertainment	Food_Travel	Places_Of_Stay	Historical	Nature	Services
104	3.590000	1.246667	1.045	3.0150	1.480	1.1275
188	2.920000	2.346667	3.295	2.7300	2.214	0.5700
3757	1.196667	2.535000	3.190	1.9225	1.098	1.7750
3204	5.000000	1.806667	3.370	2.0625	1.908	1.7200
3133	3.010000	2.200000	3.140	1.1825	2.876	1.9400

In [184]:
```
x_test.head()
```

Out[184]:

	Entertainment	Food_Travel	Places_Of_Stay	Historical	Nature	Services
3916	3.070000	2.080000	5.000	2.2925	3.278	1.1200
3101	2.333333	2.178333	1.735	2.1900	3.244	1.5975
1527	3.586667	2.275000	1.105	2.1475	1.646	1.6700
502	2.623333	1.956667	1.765	1.5500	3.016	1.0300
1172	3.710000	2.288333	1.785	1.1150	2.350	1.2950

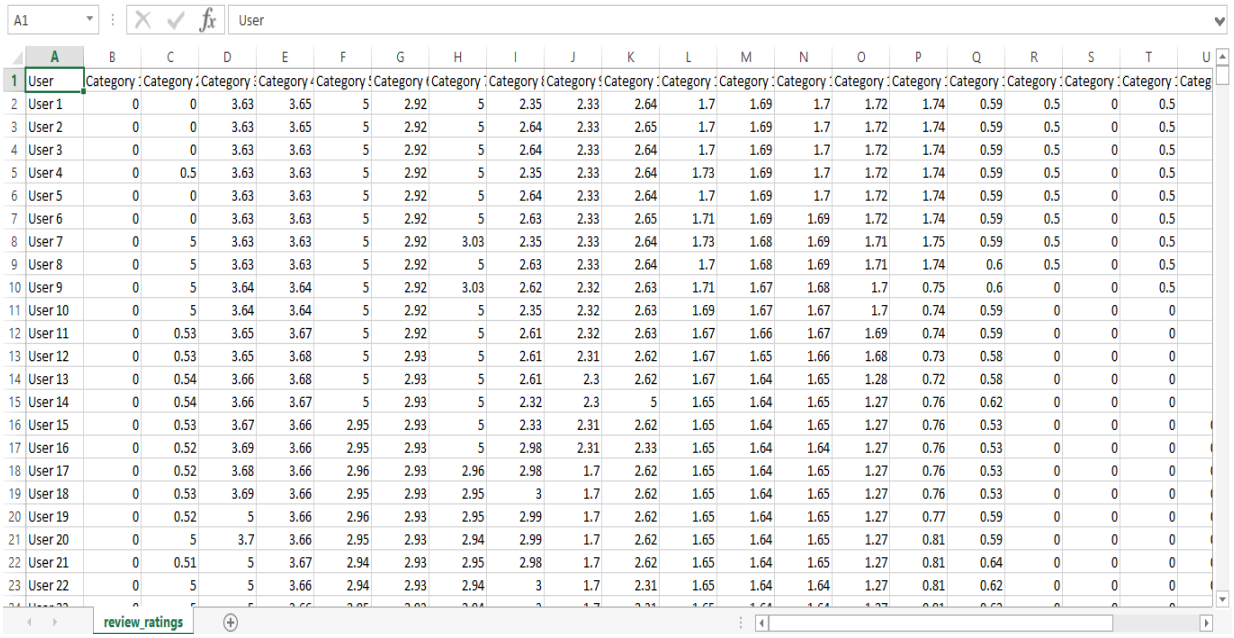
In [185]:
```
for k in range(1,100):
    k_value=k+1
    model=KNeighborsClassifier(n_neighbors=k_value, weights="uniform", algorithm='auto')
    model.fit(x_train,y_train)
    y_predict=model.predict(x_test)
    print("Accuracy", accuracy_score(y_test,y_predict)*100,"% for k values:", k_value)
```

```
Accuracy 94.04216315307058 % for k values: 2
Accuracy 94.77543538038496 % for k values: 3
Accuracy 94.40879926672777 % for k values: 4
Accuracy 94.59211732355637 % for k values: 5
Accuracy 94.40879926672777 % for k values: 6
Accuracy 94.86709440879928 % for k values: 7
Accuracy 94.22548120989917 % for k values: 8
Accuracy 94.95875343721356 % for k values: 9
Accuracy 94.40879926672777 % for k values: 10
Accuracy 94.40879926672777 % for k values: 11
Accuracy 94.40879926672777 % for k values: 12
Accuracy 94.04216315307058 % for k values: 13
Accuracy 94.31714023831348 % for k values: 14
Accuracy 94.59211732355637 % for k values: 15
Accuracy 94.59211732355637 % for k values: 16
Accuracy 94.68377635197068 % for k values: 17
Accuracy 94.50045829514208 % for k values: 18
Accuracy 94.50045829514208 % for k values: 19
Accuracy 94.40879926672777 % for k values: 20
Accuracy 94.40879926672777 % for k values: 21
Accuracy 94.59211732355637 % for k values: 22
Accuracy 94.50045829514208 % for k values: 23
Accuracy 93.85884509624198 % for k values: 24
Accuracy 94.04216315307058 % for k values: 25
Accuracy 94.22548120989917 % for k values: 26
Accuracy 93.85884509624198 % for k values: 27
Accuracy 93.76718606782768 % for k values: 28
Accuracy 93.76718606782768 % for k values: 29
Accuracy 93.49220898258478 % for k values: 30
Accuracy 93.49220898258478 % for k values: 31
Accuracy 93.67552703941338 % for k values: 32
Accuracy 93.49220898258478 % for k values: 33
Accuracy 93.76718606782768 % for k values: 34
Accuracy 93.12557286892759 % for k values: 35
Accuracy 93.03391384051329 % for k values: 36
Accuracy 93.12557286892759 % for k values: 37
Accuracy 93.03391384051329 % for k values: 38
Accuracy 93.12557286892759 % for k values: 39
Accuracy 92.942254812099 % for k values: 40
Accuracy 92.942254812099 % for k values: 41
Accuracy 92.8505957836847 % for k values: 42
Accuracy 92.7589367552704 % for k values: 43
Accuracy 92.5756186984418 % for k values: 44
Accuracy 92.66727772685608 % for k values: 45
Accuracy 92.3923006416132 % for k values: 46
Accuracy 92.48395967002749 % for k values: 47
Accuracy 92.30064161319889 % for k values: 48
Accuracy 92.5756186984418 % for k values: 49
Accuracy 92.48395967002749 % for k values: 50
Accuracy 92.3923006416132 % for k values: 51
Accuracy 92.30064161319889 % for k values: 52
Accuracy 92.30064161319889 % for k values: 53
Accuracy 92.20898258478461 % for k values: 54
Accuracy 92.02566452795601 % for k values: 55
Accuracy 92.20898258478461 % for k values: 56
Accuracy 92.48395967002749 % for k values: 57
Accuracy 92.48395967002749 % for k values: 58
Accuracy 92.30064161319889 % for k values: 59
Accuracy 91.9340054995417 % for k values: 60
Accuracy 92.02566452795601 % for k values: 61
Accuracy 92.1173235563703 % for k values: 62
Accuracy 92.1173235563703 % for k values: 63
Accuracy 92.20898258478461 % for k values: 64
Accuracy 92.3923006416132 % for k values: 65
Accuracy 92.30064161319889 % for k values: 66
Accuracy 92.30064161319889 % for k values: 67
Accuracy 92.20898258478461 % for k values: 68
Accuracy 92.30064161319889 % for k values: 69
Accuracy 92.1173235563703 % for k values: 70
```

```
Accuracy 92.3923006416132 % for k values: 71
Accuracy 92.5756186984418 % for k values: 72
Accuracy 92.5756186984418 % for k values: 73
Accuracy 92.7589367552704 % for k values: 74
Accuracy 92.7589367552704 % for k values: 75
Accuracy 92.8505957836847 % for k values: 76
Accuracy 92.5756186984418 % for k values: 77
Accuracy 92.66727772685608 % for k values: 78
Accuracy 92.66727772685608 % for k values: 79
Accuracy 92.5756186984418 % for k values: 80
Accuracy 92.30064161319889 % for k values: 81
Accuracy 92.30064161319889 % for k values: 82
Accuracy 92.02566452795601 % for k values: 83
Accuracy 92.20898258478461 % for k values: 84
Accuracy 92.1173235563703 % for k values: 85
Accuracy 92.02566452795601 % for k values: 86
Accuracy 92.02566452795601 % for k values: 87
Accuracy 92.1173235563703 % for k values: 88
Accuracy 92.02566452795601 % for k values: 89
Accuracy 92.02566452795601 % for k values: 90
Accuracy 91.9340054995417 % for k values: 91
Accuracy 92.02566452795601 % for k values: 92
Accuracy 92.20898258478461 % for k values: 93
Accuracy 92.20898258478461 % for k values: 94
Accuracy 92.1173235563703 % for k values: 95
Accuracy 92.20898258478461 % for k values: 96
Accuracy 92.1173235563703 % for k values: 97
Accuracy 92.1173235563703 % for k values: 98
Accuracy 92.02566452795601 % for k values: 99
Accuracy 92.02566452795601 % for k values: 100
```

Outputs

Dataset



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	User	Category	Category	Category	Category	Category	Category	Category	Category	Category	Category	Category	Category	Category	Category	Category	Category	Category	Category	Category	Categ
2	User 1	0	0	3.63	3.65	5	2.92	5	2.35	2.33	2.64	1.7	1.69	1.7	1.72	1.74	0.59	0.5	0	0.5	
3	User 2	0	0	3.63	3.65	5	2.92	5	2.64	2.33	2.65	1.7	1.69	1.7	1.72	1.74	0.59	0.5	0	0.5	
4	User 3	0	0	3.63	3.63	5	2.92	5	2.64	2.33	2.64	1.7	1.69	1.7	1.72	1.74	0.59	0.5	0	0.5	
5	User 4	0	0.5	3.63	3.63	5	2.92	5	2.35	2.33	2.64	1.73	1.69	1.7	1.72	1.74	0.59	0.5	0	0.5	
6	User 5	0	0	3.63	3.63	5	2.92	5	2.64	2.33	2.64	1.7	1.69	1.7	1.72	1.74	0.59	0.5	0	0.5	
7	User 6	0	0	3.63	3.63	5	2.92	5	2.63	2.33	2.65	1.71	1.69	1.69	1.72	1.74	0.59	0.5	0	0.5	
8	User 7	0	5	3.63	3.63	5	2.92	3.03	2.35	2.33	2.64	1.73	1.68	1.69	1.71	1.75	0.59	0.5	0	0.5	
9	User 8	0	5	3.63	3.63	5	2.92	5	2.63	2.33	2.64	1.7	1.68	1.69	1.71	1.74	0.6	0.5	0	0.5	
10	User 9	0	5	3.64	3.64	5	2.92	3.03	2.62	2.32	2.63	1.71	1.67	1.68	1.7	0.75	0.6	0	0	0.5	
11	User 10	0	5	3.64	3.64	5	2.92	5	2.35	2.32	2.63	1.69	1.67	1.67	1.7	0.74	0.59	0	0	0	
12	User 11	0	0.53	3.65	3.67	5	2.92	5	2.61	2.32	2.63	1.67	1.66	1.67	1.69	0.74	0.59	0	0	0	
13	User 12	0	0.53	3.65	3.68	5	2.93	5	2.61	2.31	2.62	1.67	1.65	1.66	1.68	0.73	0.58	0	0	0	
14	User 13	0	0.54	3.66	3.68	5	2.93	5	2.61	2.3	2.62	1.67	1.64	1.65	1.28	0.72	0.58	0	0	0	
15	User 14	0	0.54	3.66	3.67	5	2.93	5	2.32	2.3	5	1.65	1.64	1.65	1.27	0.76	0.62	0	0	0	
16	User 15	0	0.53	3.67	3.66	2.95	2.93	5	2.33	2.31	2.62	1.65	1.64	1.65	1.27	0.76	0.53	0	0	0	
17	User 16	0	0.52	3.69	3.66	2.95	2.93	5	2.98	2.31	2.33	1.65	1.64	1.64	1.27	0.76	0.53	0	0	0	
18	User 17	0	0.52	3.68	3.66	2.96	2.93	2.96	2.98	1.7	2.62	1.65	1.64	1.65	1.27	0.76	0.53	0	0	0	
19	User 18	0	0.53	3.69	3.66	2.95	2.93	2.95	3	1.7	2.62	1.65	1.64	1.65	1.27	0.76	0.53	0	0	0	
20	User 19	0	0.52	5	3.66	2.96	2.93	2.95	2.99	1.7	2.62	1.65	1.64	1.65	1.27	0.77	0.59	0	0	0	
21	User 20	0	5	3.7	3.66	2.95	2.93	2.94	2.99	1.7	2.62	1.65	1.64	1.65	1.27	0.81	0.59	0	0	0	
22	User 21	0	0.51	5	3.67	2.94	2.93	2.95	2.98	1.7	2.62	1.65	1.64	1.65	1.27	0.81	0.64	0	0	0	
23	User 22	0	5	5	3.66	2.94	2.93	2.94	3	1.7	2.31	1.65	1.64	1.64	1.27	0.81	0.62	0	0	0	

review_ratings

EDA (Exploratory Data Analysis)

1. load data in Jupyter Notebook using pandas: Df=pd.read_csv('review_ratings.csv')

2. to check the shape of data: Df.shape

3. print first 5 rows of data Df.head()

4. change the column name: Df.columns = cols

cols=['U_Id','Churches','Resorts','Beaches','Parks','Theatres','Museums','Malls','Zoo','Restaurants','Pubs_Bars','Local_Services','Burger_Pizza_Shops','Hotels','Juice_Bars','Art_Galleries','Dance_Clubs','Swimming_Pool','Gyms','Bakeries','Beauty_Spa','Cafes','View_Points','Monuments','Gardens']

5. describe each column:

```
In [9]: read_df.Churches.describe()

Out[9]: count    5456.000000
        mean        1.455720
        std         0.827604
        min         0.000000
        25%         0.920000
        50%         1.340000
        75%         1.810000
        max         5.000000
        Name: Churches, dtype: float64
```

```
In [11]: read_df.Resorts.describe()

Out[11]: count    5456.000000
         mean        2.319707
         std         1.421438
         min         0.000000
         25%         1.360000
         50%         1.905000
         75%         2.682500
         max         5.000000
         Name: Resorts, dtype: float64
```

```
In [12]: read_df.Beaches.describe()

Out[12]: count    5456.000000
         mean        2.489331
         std         1.247815
         min         0.000000
         25%         1.540000
         50%         2.060000
         75%         2.740000
         max         5.000000
         Name: Beaches, dtype: float64
```

```
In [13]: read_df.Parks.describe()
Out[13]: count    5456.000000
         mean        2.796886
         std         1.309159
         min         0.830000
         25%         1.730000
         50%         2.460000
         75%         4.092500
         max         5.000000
         Name: Parks, dtype: float64
```

```
In [14]: read_df.Theatres.describe()
Out[14]: count    5456.000000
         mean        2.958941
         std         1.339056
         min         1.120000
         25%         1.770000
         50%         2.670000
         75%         4.312500
         max         5.000000
         Name: Theatres, dtype: float64
```

```
In [15]: read_df.Museums.describe()
Out[15]: count    5456.00000
         mean        2.89349
         std         1.28240
         min         1.11000
         25%         1.79000
         50%         2.68000
         75%         3.84000
         max         5.00000
         Name: Museums, dtype: float64
```

```
In [16]: read_df.Malls.describe()
Out[16]: count    5456.000000
         mean        3.351395
         std         1.413492
         min         1.120000
         25%         1.930000
         50%         3.230000
         75%         5.000000
         max         5.000000
         Name: Malls, dtype: float64
```

```
In [17]: read_df.Zoo.describe()
Out[17]: count    5456.000000
         mean        2.540795
         std         1.111391
         min         0.860000
         25%         1.620000
         50%         2.170000
         75%         3.190000
         max         5.000000
         Name: Zoo, dtype: float64
```

```
In [18]: read_df.Restaurants.describe()
Out[18]: count    5456.000000
         mean        3.126019
         std         1.356802
         min         0.840000
         25%         1.800000
         50%         2.800000
         75%         5.000000
         max         5.000000
         Name: Restaurants, dtype: float64
```

```
In [19]: read_df.Pubs_Bars.describe()
Out[19]: count    5456.000000
         mean        2.832729
         std         1.307665
         min         0.810000
         25%         1.640000
         50%         2.680000
         75%         3.530000
         max         5.000000
         Name: Pubs_Bars, dtype: float64
```

```
In [20]: read_df.Local_Services.describe()
Out[20]: count    5456.000000
         mean        2.553636
         std         1.406511
         min         0.780000
         25%         1.580000
         50%         2.000000
         75%         3.220000
         max        22.000000
         Name: Local_Services, dtype: float64
```

```
In [21]: read_df.Burger_Pizza_Shops.describe()
Out[21]: count    5455.000000
         mean        2.078339
         std         1.249208
         min         0.780000
         25%         1.290000
         50%         1.690000
         75%         2.285000
         max         5.000000
         Name: Burger_Pizza_Shops, dtype: float64
```

```
In [22]: read_df.Hotels.describe()
Out[22]: count    5456.000000
         mean        2.125511
         std         1.406542
         min         0.770000
         25%         1.190000
         50%         1.610000
         75%         2.360000
         max         5.000000
         Name: Hotels, dtype: float64
```

```
In [23]: read_df.Juice_Bars.describe()
Out[23]: count    5456.000000
         mean        2.190861
         std         1.576686
         min         0.760000
         25%         1.030000
         50%         1.490000
         75%         2.740000
         max         5.000000
         Name: Juice_Bars, dtype: float64
```

```
In [24]: read_df.Art_Galleries.describe()
Out[24]: count    5456.000000
         mean        2.206573
         std         1.715961
         min         0.000000
         25%         0.860000
         50%         1.330000
         75%         4.440000
         max         5.000000
         Name: Art_Galleries, dtype: float64
```

```
In [25]: read_df.Dance_Clubs.describe()
Out[25]: count    5456.000000
         mean        1.192801
         std         1.107005
         min         0.000000
         25%         0.690000
         50%         0.800000
         75%         1.160000
         max         5.000000
         Name: Dance_Clubs, dtype: float64
```

```
In [26]: read_df.Swimming_Pool.describe()
Out[26]: count    5456.000000
         mean        0.949203
         std         0.973536
         min         0.000000
         25%         0.580000
         50%         0.740000
         75%         0.910000
         max         5.000000
         Name: Swimming_Pool, dtype: float64
```

```
In [27]: read_df.Gyms.describe()
Out[27]: count    5456.000000
         mean        0.822414
         std         0.947911
         min         0.000000
         25%         0.530000
         50%         0.690000
         75%         0.840000
         max         5.000000
         Name: Gyms, dtype: float64
```

```
In [28]: read_df.Bakeries.describe()
Out[28]: count    5456.000000
         mean        0.969811
         std         1.203972
         min         0.000000
         25%         0.520000
         50%         0.690000
         75%         0.860000
         max         5.000000
         Name: Bakeries, dtype: float64
```

```
In [29]: read_df.Beauty_Spa.describe()
Out[29]: count    5456.000000
         mean        1.000071
         std         1.193891
         min         0.000000
         25%         0.540000
         50%         0.690000
         75%         0.860000
         max         5.000000
         Name: Beauty_Spa, dtype: float64
```

```
In [30]: read_df.Cafes.describe()
Out[30]: count    5456.000000
         mean        0.965838
         std         0.929853
         min         0.000000
         25%         0.570000
         50%         0.760000
         75%         1.000000
         max         5.000000
         Name: Cafes, dtype: float64
```

```
In [31]: read_df.View_Points.describe()
Out[31]: count    5456.000000
         mean        1.750537
         std         1.598734
         min         0.000000
         25%         0.740000
         50%         1.030000
         75%         2.070000
         max         5.000000
         Name: View_Points, dtype: float64
```

```
In [32]: read_df.Monuments.describe()
Out[32]: count    5456.000000
         mean        1.531453
         std         1.316889
         min         0.000000
         25%         0.790000
         50%         1.070000
         75%         1.560000
         max         5.000000
         Name: Monuments, dtype: float64
```

```
In [33]: read_df.Gardens.describe()
Out[33]: count    5455.000000
         mean        1.560755
         std         1.171756
         min         0.000000
         25%         0.880000
         50%         1.290000
         75%         1.660000
         max         5.000000
         Name: Gardens, dtype: float64
```

6. Check if any of the column having “na” values or not. We can see, Burger_Pizza_Shops and Gardens columns having na values.

```
In [120]: read_df[cols].isnull().sum()
Out[120]: U_Id                0
          Churches            0
          Resorts             0
          Beaches             0
          Parks               0
          Theatres            0
          Museums             0
          Malls               0
          Zoo                 0
          Restaurants         0
          Pubs_Bars           0
          Local_Services      0
          Burger_Pizza_Shops  1
          Hotels              0
          Juice_Bars          0
          Art_Galleries       0
          Dance_Clubs         0
          Swimming_Pool       0
          Gyms                0
          Bakeries            0
          Beauty_Spa          0
          Cafes               0
          View_Points         0
          Monuments           0
          Gardens             1
          dtype: int64
```

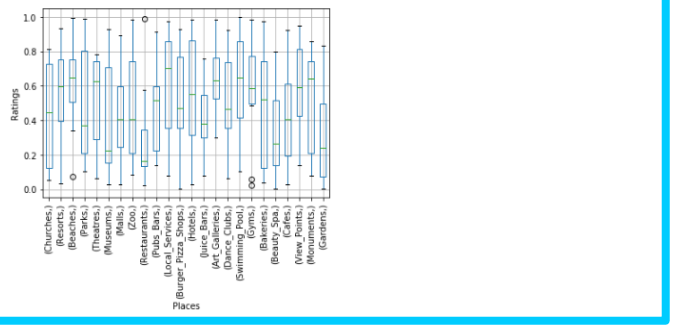
7. Remove Na values by dropping those rows. Using dropna(), the columns having na values will be removed by removing na rows from the dataset.

```
In [8]: # remove null values
        read_df=read_df.dropna()

In [9]: read_df.isnull().sum()
Out[9]: user_id             0
        churches            0
        resorts             0
        beaches             0
        parks               0
        theatres            0
        museums             0
        malls               0
        zoo                 0
        restaurants         0
        pubs_bars           0
        local_services      0
        burger_pizza_shops  0
```

8. Plotting Box Plot of all columns

Observation: It is observed that on X- axis, there are places and on Y- axis, there are user ratings. We can see that there are outliers in Beaches, Restaurants and Gyms columns but not affecting the rest of the data hence we can keep these outliers.

```
col_name=['Churches','Resorts','Beaches','Parks','Theatres','Museums','Malls','Zoo','Restaurants','F
df = pd.DataFrame(np.random.rand(10, 24), columns=[col_name]) # taken four columns and 10 rows rand
df.plot.box(grid='True',rot=90)
plt.xlabel('Places')
plt.ylabel('Ratings')

C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\cbook\__init__.py:424: MatplotlibDeprecationWa
Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated; use an actual boolean (True/
  warn_deprecated("2.2", "Passing one of 'on', 'true', 'off', 'false' as a "

Out[49]: Text(0, 0.5, 'Ratings')
```



K- Means Clustering

1. Make a category of different places

```
In [24]: entertainment = ['theatres', 'dance_clubs', 'malls']
         food_travel = ['restaurants', 'pubs_bars', 'burger_pizza_shops', 'juice_bars', 'bakeries', 'cafes']
         places_of_stay = ['hotels_other_lodgings', 'resorts']
         historical = ['churches', 'museums', 'art_galleries', 'monuments']
         nature = ['beaches', 'parks', 'zoo', 'view_points', 'gardens']
         services = ['local_services', 'swimming_pools', 'gyms', 'beauty_spas']
```

2. Store as data frame

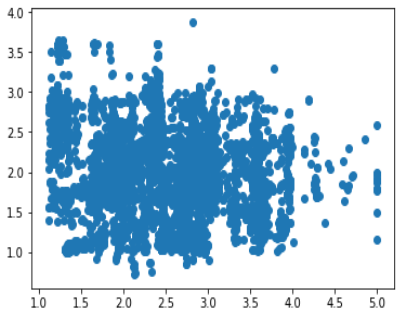
```
In [25]: df_category_reviews = pd.DataFrame(columns = ['entertainment', 'food_travel', 'places_of_stay', 'historical', 'nature', 'service
```

3. Storing values of all columns in new columns

```
In [26]: df_category_reviews['entertainment'] = read_df[entertainment].mean(axis = 1)
         df_category_reviews['food_travel'] = read_df[food_travel].mean(axis = 1)
         df_category_reviews['places_of_stay'] = read_df[places_of_stay].mean(axis = 1)
         df_category_reviews['historical'] = read_df[historical].mean(axis = 1)
         df_category_reviews['nature'] = read_df[nature].mean(axis = 1)
         df_category_reviews['services'] = read_df[services].mean(axis = 1)
```

4. Storing data in array and plotting scatter plot

```
In [35]: X=np.array(df_category_reviews)
         plt.scatter(X[:,0],X[:,1], label='True Position') # 0 columns

Out[35]: <matplotlib.collections.PathCollection at 0x184280758d0>
```



5. Using kmeans function in which no. of cluster we define and predict centroids

```
In [38]: kmeans = KMeans(n_clusters=6)
         kmeans.fit(X)
         print(kmeans.cluster_centers_)

[[2.03552119 1.95264318 2.01493986 2.27030498 2.14342268 2.3124055 ]
 [2.22085048 2.21692215 1.42566872 1.6171142  1.98834362 1.31366512]
 [2.25702577 1.42686734 1.70437896 2.19711027 3.06609125 1.18466096]
 [1.86930212 2.65485479 2.95715215 1.62366385 1.44246294 1.51140767]
```

6. Now we predict which row belongs to which cluster and store the value inserting new column “cluster”

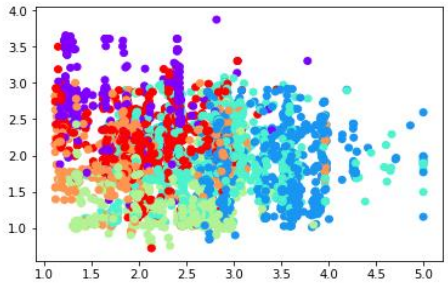
```
In [41]: y_predict=kmeans.fit_predict(df_category_reviews[['entertainment', 'food_travel', 'places_of_stay', 'historical', 'nature',
         y_predict
Out[41]: array([2, 2, 2, ..., 0, 0, 5])

In [44]: df_category_reviews['cluster']=y_predict
         df_category_reviews.head(20)
         #df_category_reviews.drop('Avg_Ratings',axis=1)
```

Out[44]:

	entertainment	food_travel	places_of_stay	historical	nature	services	cluster
0	3.530000	1.480000	0.850	1.1650	1.926	0.5500	2
1	3.530000	1.481667	0.850	1.1650	1.984	0.5500	2
2	3.530000	1.480000	0.850	1.1650	1.980	0.5500	2
3	3.530000	1.480000	1.100	1.1650	1.922	0.5575	2
4	3.530000	1.480000	0.850	1.1650	1.980	0.5500	2
5	3.530000	1.481667	0.845	1.1650	1.978	0.5525	2
6	2.873333	1.476667	3.345	1.1675	1.922	0.5575	3
7	3.533333	1.476667	3.345	1.1650	1.978	0.5500	3

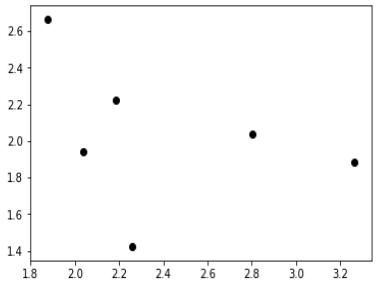
7. Plotting points of no of cluster as we entered in above code

```
In [103]: plt.scatter(X[:,0],X[:,1], c=kmeans.labels_, cmap='rainbow')
Out[103]: <matplotlib.collections.PathCollection at 0x21d07a0f6a0>
```



8. Plotting Centroids

```
In [104]: plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],color='black')
Out[104]: <matplotlib.collections.PathCollection at 0x21d05bd9a20>
```



Model Creation: Now take a sample data from the population we are having, to perform training and testing. In y we will store a target variable and in x we will store rest of the columns on which training and testing will perform.

```
In [61]: from sklearn.model_selection import train_test_split
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score
         y=df_category_reviews.cluster
         y.head()
         x=df_category_reviews.iloc[:,0:6]
         x.head()
         x_train, x_test, y_train, y_test= train_test_split(x,y,test_size=0.2,random_state=0)
         #trainset, testset = train_test_split(df_category_reviews, test_size=0.25, random_state = 0)
```

Final Model: In this Model, we can see Accuracy score for k values are 2-100 and our best Accuracy score is 94.77%, for k value, 2.

```
In [113]: for k in range(1,100):
              k_value=k+1
              model=KNeighborsClassifier(n_neighbors=k_value, weights="uniform", algorithm='auto')
              model.fit(x_train,y_train)
              y_predict=model.predict(x_test)
              print("Accuracy", accuracy_score(y_test,y_predict)*100,"% for k values:", k_value)
```

```
Accuracy 94.50045829514208 % for k values: 2
Accuracy 94.77543538038496 % for k values: 3
Accuracy 94.77543538038496 % for k values: 4
Accuracy 95.14207149404217 % for k values: 5
Accuracy 94.13382218148487 % for k values: 6
Accuracy 95.14207149404217 % for k values: 7
Accuracy 94.50045829514208 % for k values: 8
Accuracy 94.95875343721356 % for k values: 9
Accuracy 94.59211732355637 % for k values: 10
Accuracy 94.86709440879928 % for k values: 11
Accuracy 94.31714023831348 % for k values: 12
Accuracy 94.59211732355637 % for k values: 13
Accuracy 94.77543538038496 % for k values: 14
Accuracy 94.68377635197068 % for k values: 15
Accuracy 94.40879926672777 % for k values: 16
Accuracy 94.50045829514208 % for k values: 17
Accuracy 93.95050412465628 % for k values: 18
Accuracy 94.13382218148487 % for k values: 19
Accuracy 93.95050412465628 % for k values: 20
```

```
Accuracy 94.40879926672777 % for k values: 21
Accuracy 94.59211732355637 % for k values: 22
Accuracy 94.50045829514208 % for k values: 23
Accuracy 93.85884509624198 % for k values: 24
Accuracy 94.04216315307058 % for k values: 25
Accuracy 94.22548120989917 % for k values: 26
Accuracy 93.85884509624198 % for k values: 27
Accuracy 93.76718606782768 % for k values: 28
Accuracy 93.76718606782768 % for k values: 29
Accuracy 93.49220898258478 % for k values: 30
Accuracy 93.49220898258478 % for k values: 31
Accuracy 93.67552703941338 % for k values: 32
Accuracy 93.49220898258478 % for k values: 33
Accuracy 93.76718606782768 % for k values: 34
Accuracy 93.12557286892759 % for k values: 35
Accuracy 93.03391384051329 % for k values: 36
Accuracy 93.12557286892759 % for k values: 37
Accuracy 93.03391384051329 % for k values: 38
Accuracy 93.12557286892759 % for k values: 39
Accuracy 92.942254812099 % for k values: 40
Accuracy 92.942254812099 % for k values: 41
Accuracy 92.8505957836847 % for k values: 42
Accuracy 92.7589367552704 % for k values: 43
Accuracy 92.5756186984418 % for k values: 44
```

```
Accuracy 92.66727772685608 % for k values: 45
Accuracy 92.3923006416132 % for k values: 46
Accuracy 92.48395967002749 % for k values: 47
Accuracy 92.30064161319889 % for k values: 48
Accuracy 92.5756186984418 % for k values: 49
Accuracy 92.48395967002749 % for k values: 50
Accuracy 92.3923006416132 % for k values: 51
Accuracy 92.30064161319889 % for k values: 52
Accuracy 92.30064161319889 % for k values: 53
Accuracy 92.20898258478461 % for k values: 54
Accuracy 92.02566452795601 % for k values: 55
Accuracy 92.20898258478461 % for k values: 56
Accuracy 92.48395967002749 % for k values: 57
Accuracy 92.48395967002749 % for k values: 58
Accuracy 92.30064161319889 % for k values: 59
Accuracy 91.9340054995417 % for k values: 60
Accuracy 92.02566452795601 % for k values: 61
Accuracy 92.1173235563703 % for k values: 62
Accuracy 92.1173235563703 % for k values: 63
Accuracy 92.20898258478461 % for k values: 64
Accuracy 92.3923006416132 % for k values: 65
Accuracy 92.30064161319889 % for k values: 66
Accuracy 92.30064161319889 % for k values: 67
Accuracy 92.20898258478461 % for k values: 68
Accuracy 92.30064161319889 % for k values: 69
Accuracy 92.1173235563703 % for k values: 70
Accuracy 92.3923006416132 % for k values: 71
Accuracy 92.5756186984418 % for k values: 72
Accuracy 92.5756186984418 % for k values: 73
Accuracy 92.7589367552704 % for k values: 74
Accuracy 92.7589367552704 % for k values: 75
Accuracy 92.8505957836847 % for k values: 76
Accuracy 92.5756186984418 % for k values: 77
Accuracy 92.66727772685608 % for k values: 78
Accuracy 92.66727772685608 % for k values: 79
Accuracy 92.5756186984418 % for k values: 80
Accuracy 92.30064161319889 % for k values: 81
```

```
Accuracy 92.30064161319889 % for k values: 82
Accuracy 92.02566452795601 % for k values: 83
Accuracy 92.20898258478461 % for k values: 84
Accuracy 92.1173235563703 % for k values: 85
Accuracy 92.02566452795601 % for k values: 86
Accuracy 92.02566452795601 % for k values: 87
Accuracy 92.1173235563703 % for k values: 88
Accuracy 92.02566452795601 % for k values: 89
Accuracy 92.02566452795601 % for k values: 90
Accuracy 91.9340054995417 % for k values: 91
Accuracy 92.02566452795601 % for k values: 92
Accuracy 92.20898258478461 % for k values: 93
Accuracy 92.20898258478461 % for k values: 94
Accuracy 92.1173235563703 % for k values: 95
Accuracy 92.20898258478461 % for k values: 96
Accuracy 92.1173235563703 % for k values: 97
Accuracy 92.1173235563703 % for k values: 98
Accuracy 92.02566452795601 % for k values: 99
Accuracy 92.02566452795601 % for k values: 100
```

Conclusion

The impact of online reviews become more and more relevant in the era of social media, both for businesses and consumers. Travelers rely on online reviews to make decisions about trips planning, whereas businesses take advantage of them to establish effective marketing strategies. Nevertheless, the great amount of available data makes it unfeasible to analyse all the available reviews one by one. Thus, in recent years, several efforts have been done to propose methods that automatically analyse and summarize the reviews features.

In this, initially, we introduced the Travel Packages Prediction problem and the significance of using predictive modeling methods to overcome the problem of customer that travel. We surveyed the existing travel prediction methods in detail and summarized them. Unlike other surveys, which primarily focused only on the prediction models and the accuracy of travel packages prediction, in this survey we presented the characteristics of the existing publicly available travel packages prediction datasets. Further, we focused on different customer related variables that are used for travel packages prediction and categorized them. Finally, we surveyed the list of the commonly used metrics proposed in the literature for evaluating the performance of various travel packages prediction methods.

Summarizing, the contribution presented in this is

i. aligned with current trends of literature investigation,
ii. offer diverse kinds of analyses, and
iii. put the bases for the design and development of fine grained recommender systems.

A study on online travel reviews through Intelligent Data Analysis

The analyses have been performed by applying several techniques coming from the field of Intelligent Data Analysis, which include:

a) an overall statistical study on the reviewers activities,
b) a content analysis by applying Machine Learning techniques on the subset of English reviews, mainly to compare travel-related review platforms, and
c) the extraction of frequent patterns of destinations by exploiting an K- Means Clustering, Unsupervised algorithm.

The outcome allowed for a better characterization of the visitors' habits and preferences: from the time of the year of highest affluence (and the relative change in the visitors' satisfaction) to frequent destinations patterns common to users' groups. While our running scenario turns around an Europe tourist hub, the same explorations could be easily carry out on different locations, with diverse granularity degrees.

The main aim is to suggest similar routes to potential travelers that present characteristics similar to those experimenting recurrent tours. Another work in progress is the comparison of specific characteristics of locations, through, e.g., the analysis of the appreciation of the attractions and "things to do" expressed by the reviewers, about different towns in the same region (e.g., things to do in Lucca, Pisa, Florence, and Siena, all located in the Tuscany region).

Future Enhancement

"The past cannot be changed.
The future is yet in your power."

The project has a very vast scope in future. The project can be implemented on intranet in future. Project can be updated in near future as and when requirement for the same arises, as it is very flexible in terms of expansion. With the proposed, the client now able to manage all the travel requirements at very reasonable prices, manage time, minimize distance and hence run the entire work in a much better, accurate and risk free manner. Although our system had been completed but it is not perfect, we had planned to make some enhancement in the future. We think that our system still has potential to grow. The following are the future scope for the project:

- **To create a model in which any unsupervised dataset work out.**
- **To update user rating review automatically in dataset.**

Beside the challenges featured by relying on K- Means Clustering, additional limitations are mainly related to the fact that not all the actual visitors use to post a review after their travels. Thus, the obtained results give an incomplete overview of the visitors' traveling behaviour. Future work may consider:

- **The possibility of extracting user profiles from data, keeping however in mind that not all the possible profiles of travelers have a representation in the datasets.**
- **To incorporate such user profiles into a market analysis tool for providers, with the aim of defining marketing strategies adjusted to specific groups of users.**

Reference

[1]. Aghdam AR, Kamalpour M, Chen D, Sim ATH, Hee JM (2014) Identifying places of interest for tourists using knowledge discovery techniques. International Conference on Industrial Automation, Information and Communications Technology, pp 130–134.

[2]. Akman I, Mishra A (2017) Factors influencing consumer intention in social commerce adoption. IT & People 30(2):356–370.

[3]. Alaei AR, Becken S, Stantic B (2018) Sentiment analysis in tourism: Capitalizing on big data. Journal of Travel Research - in press.

[4]. Alstott J, Bullmore E, Plenz D (2014) Powerlaw: A Python package for analysis of heavy-tailed distributions. PLOS ONE 9(1):1–11.

[5]. Amaro S, Duarte P (2017) Social media use for travel purposes: across cultural comparison between portugal and the uk. Information Technology & Tourism 17(2):161–181.

[6]. Berezina K, Bilgihan A, Cobanoglu C, Okumus F (2016) Understanding satisfied and dissatisfied hotel customers: Text mining of online hotel reviews. Journal of Hospitality Marketing & Management 25(1):124.

[7]. Chen J, Zhang C, Niu Z (2016) Identifying helpful online reviews with word embedding features. In: Knowledge Science, Engineering and Management, Springer, pp 123–133.

[8]. Chong AYL, Ch'ng E, Liu MJ, Li B (2017) Predicting consumer product demands via big data: the roles of online promotional marketing and online reviews. International Journal of Production Research 55(17):5142–5156.

[9]. A study on online travel reviews through Intelligent Data Analysis Fang Q, Xu C, Sang J, Hossain MS, Muhammad G (2015) Word-of-mouth understanding: Entity-centric multimodal aspect-opinion mining in social media. IEEE Trans Multimedia 17(12):2281–2296.

[10]. Garc´ıa-Pablos A, Cuadros M, Linaza MT (2016) Automatic analysis of textual hotel reviews. Information Technology & Tourism 16(1):45–69.

[11]. Gretzel U, Yoo KH (2008) Use and Impact of Online Travel Reviews, Springer Vienna, Vienna, pp 35–46.

[12]. Hu YH, Chen YL, Chou HL (2017) Opinion mining from online hotel reviews a text summarization approach. Information Processing & Management 53(2):436–449.

[13]. Krawczyk M, Xiang Z (2016) Perceptual mapping of hotel brands using online reviews: a text analytics approach. Information Technology & Tourism 16(1):23–43.

[14]. Litvin SW, Goldsmith RE, Pan B (2008) Electronic word-of-mouth in hospitality and tourism management. Tourism Management 29(3):458–468.
[15]. Menner T, H¨opken W, Fuchs M, Lexhagen M (2016) Topic detection: Identifying relevant topics in tourism reviews. In: Inversini A, Schegg R (eds) Information and Communication Technologies in Tourism 2016, Springer International Publishing, Cham, pp 411–423.
[16]. Muchnik L, Pei S, Parra LC, Reis SD, Andrade Jr JS, Havlin S, Makse HA (2013) Origins of power-law degree distribution in the heterogeneity of human activity in social networks. Scientific Reports 3(1783).
[17]. Pantano E, Priporas CV, Stylos N (2017) You will like it! Using open data to predict tourists' response to a tourist attraction. Tourism Management 60:430–438.
[18]. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12:2825–2830.
[19]. Phillips P, Barnes S, Zigan K, Schegg R (2017) Understanding the impact of online reviews on hotel performance: An empirical analysis. Journal of Travel Research 56(2):235–249.
[20]. Raguseo E, Vitari C (2017) The effect of brand on the impact of e-wom on hotels' financial performance. International Journal of Electronic Commerce 21(2):249–269.
[21]. Rossetti M, Stella F, Zanker M (2016) Analyzing user reviews in tourism with topic models. Information Technology & Tourism 16(1):5–21.
[22]. Sparks BA, Browning V (2011) The impact of online reviews on hotel booking intentions and perception of trust. Tourism Management 32(6):1310–1323.
[23]. Wang Q, Wang L, Zhang X, Mao Y, Wang P (2017) The impact research of online reviews' sentiment polarity presentation on consumer purchase decision. IT & People 30(3):522–541.
24]. Z, Gretzel U (2010) Role of social media in online travel information search. Tourism Management 31(2):179–188.
[25]. Yang J, Sia C, Liu L, Chen H (2016) Sellers versus buyers: differences in user information sharing on social commerce sites. IT & People 29(2):444–470.
[26]. Ye Q, Law R, Gu B (2009) The impact of online user reviews on hotel room sales, International Journal of Hospitality Management 28(1):180–182.
[27]. Ye Q, Law R, Gu B, Chen W (2011) The influence of user-generated content on traveler behavior: An empirical investigation on the effects of e-word-of-mouth to hotel online bookings. Computers in Human Behavior 27(2):634–639.
[28]. Zhou S, Guo B (2017) The order effect on online review helpfulness. Decision Support Systems 93(C):77–87.

Thank

You!