The Economic Relationship between Mobile Apps and Customer Location Data

Author: Shuvrangshu Mukhopadhyay
UID: 2236-9642
Faculty Advisor: Prof. Kaushik Dutta, Prof. Anik Mukherjee

Table of Contents:

Abstract

The rapid evolution of mobile technology has resulted in an exponential surge in the volume of data generated by various mobile devices. This study seeks to explore the economic correlation between mobile apps and customer location data in the United States, utilizing both exploratory and descriptive research methods, to identify patterns and correlations that can offer valuable insights for businesses. By examining the possible economic implications of mobile app usage and customer location data, this research aims to provide a new perspective for data-driven decision-making.

The study proposes recommendations for businesses to capitalize on this relationship. By applying advanced machine learning techniques and analyzing large datasets, businesses can gain a deeper understanding of customer preferences and behaviors. Furthermore, by utilizing customer location data, businesses can develop more personalized and effective marketing strategies, ultimately improving customer engagement and loyalty. This ongoing research provides a unique opportunity to explore the vast potential of mobile app and location data and to enhance our understanding of the relationship between technology and economics. Ultimately, the research aims to provide insights that can help businesses make more informed decisions, create greater value for their customers, and drive sustainable economic growth.

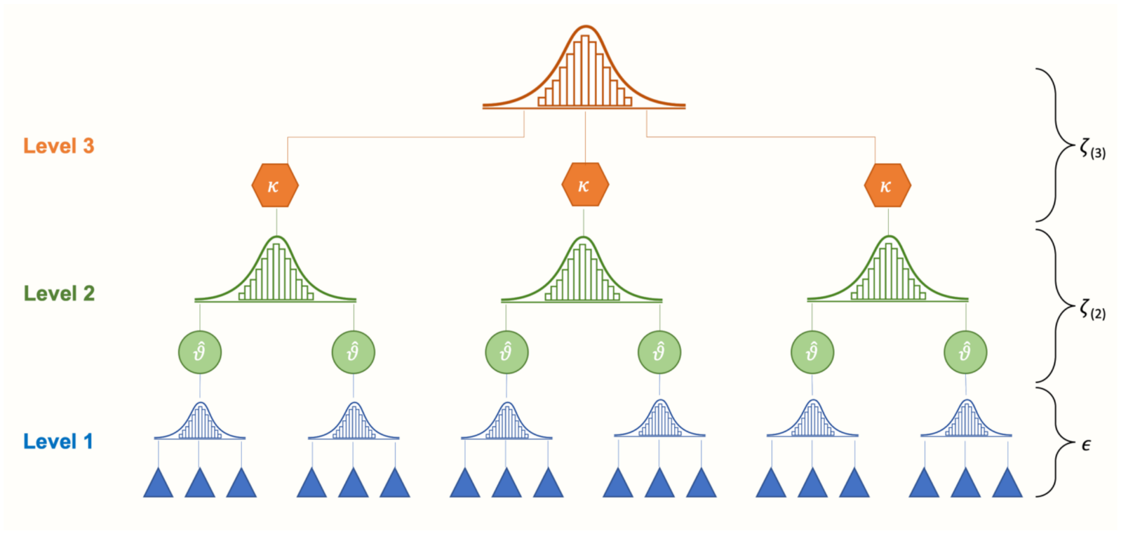
Econometric Analysis:

Economic models are simplified representations of complex economic phenomena. These models are developed by making assumptions and using mathematical equations to analyze and predict economic behavior. They assist in understanding how people, businesses, and governments make decisions and how these decisions impact the economy as a whole.

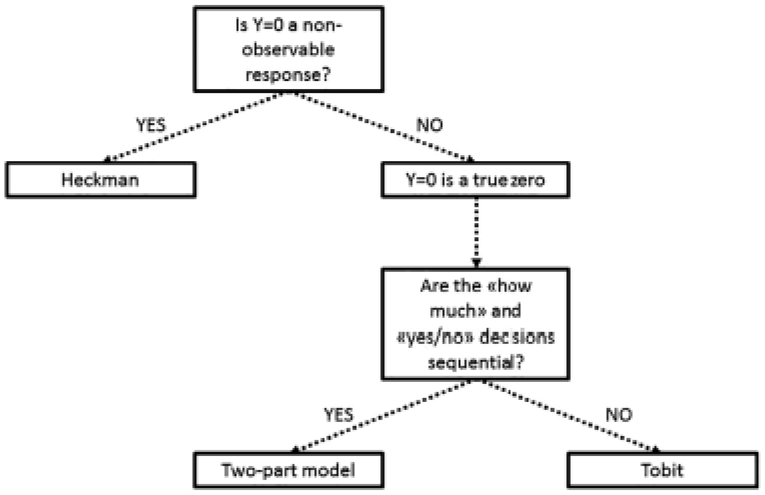
Various mathematical techniques such as calculus, linear algebra, and statistics are used to build economic models. These models are utilized to test hypotheses, analyze data, and make predictions about future economic behavior. They are also used to evaluate potential outcomes of different policies and identify potential risks and opportunities in the economy.

The implications of economic models are significant, as they help guide policy decisions at both micro and macroeconomic levels. By comprehending the impact of different factors on economic behavior, policymakers can make informed decisions about resource allocation, market regulation, and economic growth promotion. Economic models are also useful for businesses to make strategic decisions, such as pricing products or investing in research and development. In conclusion, economic models are a vital tool for understanding and managing the complexities of the modern economy.

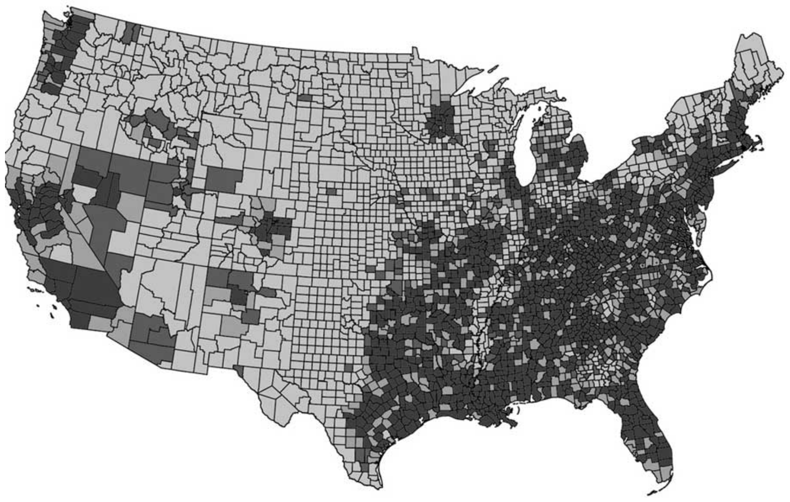
- **Multilevel regression model**: This model is useful for analyzing data that has a hierarchical structure, such as location data that is grouped by state, city, or neighborhood.



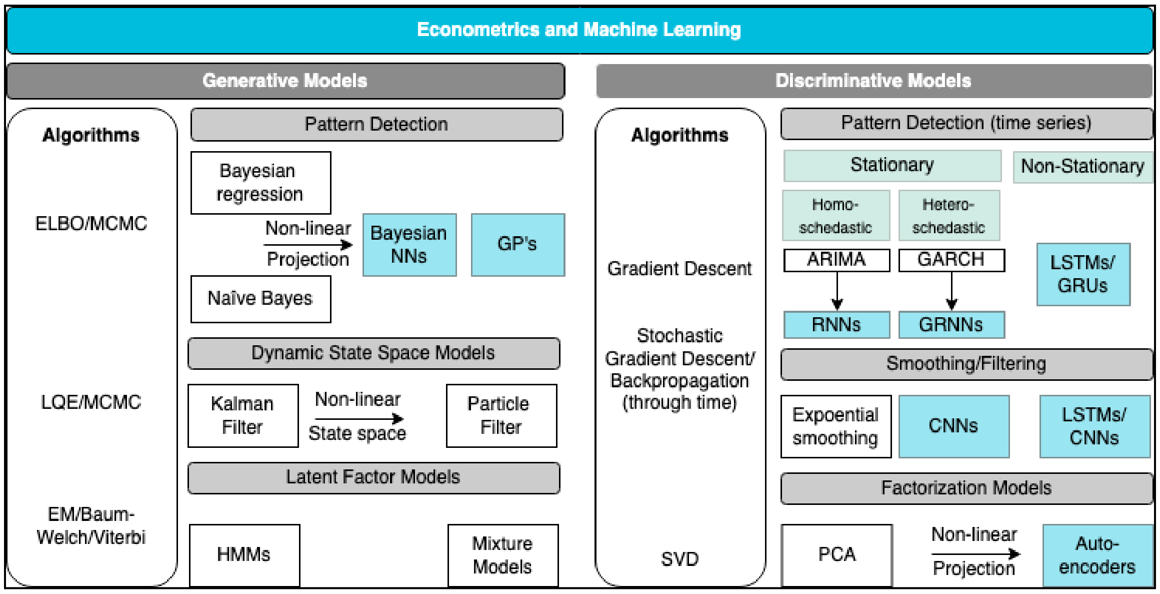
- **Tobit model**: This model is often used to analyze data that primarily do not follow any distribution and is censored data, e.g. Click Conversion Rate. The model can estimate the likelihood of a user visiting a location and the number of times they are likely to visit.


Is Y=0 a non-observable response?
YES
NO
Heckman
Y=0 is a true zero
Are the «how much» and «yes/no» decisions sequential?
YES
NO
Two-part model
Tobit

- **Spatial econometric models**: These models are used to analyze spatial data and account for spatial dependence or autocorrelation in the data. They can be useful for analyzing patterns of mobile app usage and location data across different geographic areas.



- **Machine learning models**: These models use algorithms to learn patterns in the data & make predictions or classifications. They can be used for a wide range of analyses, including predicting user behavior based on location data and identifying clusters of users with similar app usage patterns.



Methodology and Implementation

There are several steps in the methodology that is suggested for this investigation. The first step will be to extract pertinent information about consumer location data and mobile app activity in the United States from the USF's on premise research cluster. After that, this data will be cleaned to assure its precision and consistency. The data will then be processed and analyzed to find patterns and correlations that can offer business insights using exploratory and descriptive research methodologies.

Using the statistical software program Stata, econometric models will be created and run to better investigate the economic relationship between mobile apps and client location data. The data will also be subjected to advanced machine learning algorithms to get a greater understanding of client preferences and habits, which will guide the development of customized and efficient.

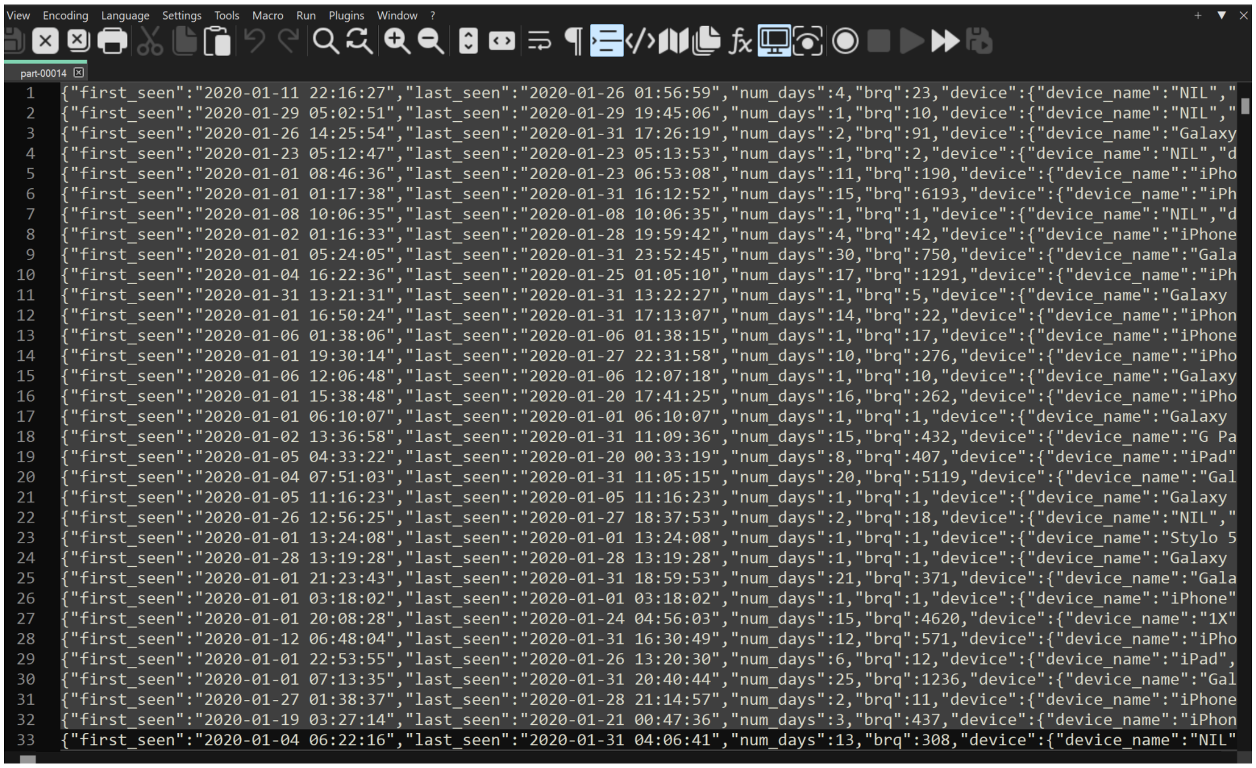
Data Extraction

The primary data source is the mobile app usage data collected from various mobile service providers across the United States. The data is stored in an unstructured nested Javascript Object Notation (json) format, and is located in an on-premise cloud. The data set includes information on mobile app usage of users from across the United States, and the study focuses on analyzing app usage data for customers across the USA.

The data set includes multiple levels of data, below are the details:

- **Customer Level:**
    - **Day_part**: Part of the day the observation was recorded
    - **Geohash**: String value that represents the exact location of a customer
    - **Zip**: Zip Value of customer's location
    - **City**: City where the customer's residence is
    - **State**: State where the customer's residence is
    - **Country**: Country where the customer's residence is
- **Device Level**:
    - **Asn**: Application name which the customer has on his/her phone
    - **Asn_exhcnage**: The exchange which was used to download the application from
    - **Device_name**: Model name of the mobile device
    - **Manufacturer**: Mobile model manufacturing company
    - **Sha1_did**: Identifier assigned by the owner of data
    - **Year_released**: Year when the mobile device was released

Below is a sample snippet of the JSON data sourced from Professor’s Directory on Terminal:


View Encoding Language Settings Tools Macro Run Plugins Window ?
part-00014

```
{"first_seen":"2020-01-11 22:16:27","last_seen":"2020-01-26 01:56:59","num_days":4,"brq":23,"device":{"device_name":"NIL","
{"first_seen":"2020-01-29 05:02:51","last_seen":"2020-01-29 19:45:06","num_days":1,"brq":10,"device":{"device_name":"NIL","
{"first_seen":"2020-01-26 14:25:54","last_seen":"2020-01-31 17:26:19","num_days":2,"brq":91,"device":{"device_name":"Galaxy
{"first_seen":"2020-01-23 05:12:47","last_seen":"2020-01-23 05:13:53","num_days":1,"brq":2,"device":{"device_name":"NIL","d
{"first_seen":"2020-01-01 08:46:36","last_seen":"2020-01-23 06:53:08","num_days":11,"brq":190,"device":{"device_name":"iPho
{"first_seen":"2020-01-01 01:17:38","last_seen":"2020-01-31 16:12:52","num_days":15,"brq":6193,"device":{"device_name":"iPh
{"first_seen":"2020-01-08 10:06:35","last_seen":"2020-01-08 10:06:35","num_days":1,"brq":1,"device":{"device_name":"NIL","d
{"first_seen":"2020-01-02 01:16:33","last_seen":"2020-01-28 19:59:42","num_days":4,"brq":42,"device":{"device_name":"iPhone
{"first_seen":"2020-01-01 05:24:05","last_seen":"2020-01-31 23:52:45","num_days":30,"brq":750,"device":{"device_name":"Gala
{"first_seen":"2020-01-04 16:22:36","last_seen":"2020-01-25 01:05:10","num_days":17,"brq":1291,"device":{"device_name":"iPh
{"first_seen":"2020-01-31 13:21:31","last_seen":"2020-01-31 13:22:27","num_days":1,"brq":5,"device":{"device_name":"Galaxy
{"first_seen":"2020-01-01 16:50:24","last_seen":"2020-01-31 17:13:07","num_days":14,"brq":22,"device":{"device_name":"iPhon
{"first_seen":"2020-01-06 01:38:06","last_seen":"2020-01-06 01:38:15","num_days":1,"brq":17,"device":{"device_name":"iPhone
{"first_seen":"2020-01-01 19:30:14","last_seen":"2020-01-27 22:31:58","num_days":10,"brq":276,"device":{"device_name":"iPho
{"first_seen":"2020-01-06 12:06:48","last_seen":"2020-01-06 12:07:18","num_days":1,"brq":10,"device":{"device_name":"Galaxy
{"first_seen":"2020-01-01 15:38:48","last_seen":"2020-01-20 17:41:25","num_days":16,"brq":262,"device":{"device_name":"iPho
{"first_seen":"2020-01-01 06:10:07","last_seen":"2020-01-01 06:10:07","num_days":1,"brq":1,"device":{"device_name":"Galaxy
{"first_seen":"2020-01-02 13:36:58","last_seen":"2020-01-31 11:09:36","num_days":15,"brq":432,"device":{"device_name":"G Pa
{"first_seen":"2020-01-05 04:33:22","last_seen":"2020-01-20 00:33:19","num_days":8,"brq":407,"device":{"device_name":"iPad"
{"first_seen":"2020-01-04 07:51:03","last_seen":"2020-01-31 11:05:15","num_days":20,"brq":5119,"device":{"device_name":"Gal
{"first_seen":"2020-01-05 11:16:23","last_seen":"2020-01-05 11:16:23","num_days":1,"brq":1,"device":{"device_name":"Galaxy
{"first_seen":"2020-01-26 12:56:25","last_seen":"2020-01-27 18:37:53","num_days":2,"brq":18,"device":{"device_name":"NIL","
{"first_seen":"2020-01-01 13:24:08","last_seen":"2020-01-01 13:24:08","num_days":1,"brq":1,"device":{"device_name":"Stylo 5
{"first_seen":"2020-01-28 13:19:28","last_seen":"2020-01-28 13:19:28","num_days":1,"brq":1,"device":{"device_name":"Galaxy
{"first_seen":"2020-01-01 21:23:43","last_seen":"2020-01-31 18:59:53","num_days":21,"brq":371,"device":{"device_name":"Gala
{"first_seen":"2020-01-01 03:18:02","last_seen":"2020-01-01 03:18:02","num_days":1,"brq":1,"device":{"device_name":"iPhone"
{"first_seen":"2020-01-01 20:08:28","last_seen":"2020-01-24 04:56:03","num_days":15,"brq":4620,"device":{"device_name":"1X"
{"first_seen":"2020-01-12 06:48:04","last_seen":"2020-01-31 16:30:49","num_days":12,"brq":571,"device":{"device_name":"iPho
{"first_seen":"2020-01-01 22:53:55","last_seen":"2020-01-26 13:20:30","num_days":6,"brq":12,"device":{"device_name":"iPad",
{"first_seen":"2020-01-01 07:13:35","last_seen":"2020-01-31 20:40:44","num_days":25,"brq":1236,"device":{"device_name":"Gal
{"first_seen":"2020-01-27 01:38:37","last_seen":"2020-01-28 21:14:57","num_days":2,"brq":11,"device":{"device_name":"iPhone
{"first_seen":"2020-01-19 03:27:14","last_seen":"2020-01-21 00:47:36","num_days":3,"brq":437,"device":{"device_name":"iPhon
{"first_seen":"2020-01-04 06:22:16","last_seen":"2020-01-31 04:06:41","num_days":13,"brq":308,"device":{"device_name":"NIL"
```

To ingest and process the data, we will be using Linux on MobaXterm. We will first automate the data processing script for a single JSON dataset, which will then be used to process the other datasets as well. This will ensure that the data is consistent and accurate, and ready for further analysis using exploratory and descriptive research methods.

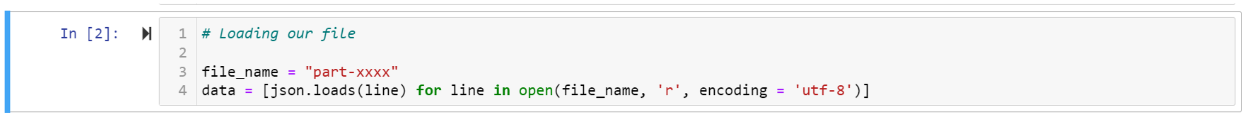
Data Loading

The JSON files are compressed using the .gz extension, requiring us to extract them before accessing the data. Once we have decompressed the files, we can save them to a directory where they can be easily loaded into our Python script.

Due to the size of the JSON files under consideration, it is necessary to carefully manage the memory resources required for processing. As such, a line-by-line reading approach has been adopted in order to store the data in a JSON object in Python.

To accomplish this, the ‘**pd_json’** method along with list comprehension has been utilized, which allows for the sequential loading of each line of the JSON file, converting it to a Python object. These individual JSON objects are then appended to a list, facilitating further processing in the Python script.

This method is highly efficient as it avoids the need to load the entire file into memory at once, thereby mitigating the risk of time-consuming and resource-intensive operations. Instead, data processing is performed one line at a time, rendering it possible to work with large datasets without encountering memory-related constraints.


In [2]:

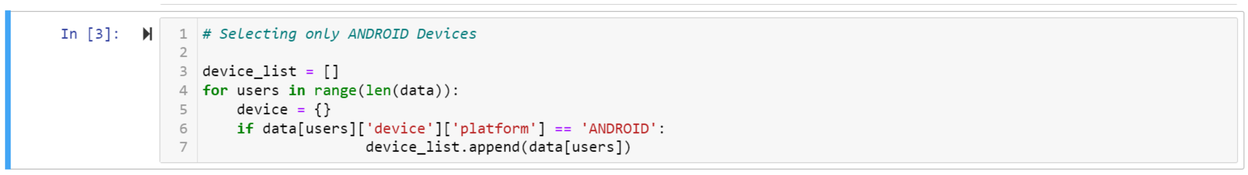
```python
# Loading our file

file_name = "part-xxxx"
data = [json.loads(line) for line in open(file_name, 'r', encoding = 'utf-8')]
```

Data Cleaning

Within our dataset, we have identified two distinct types of mobile devices: Android and IOS. After conducting an initial analysis, it became evident that Android devices were more prevalent than IOS devices. As a result, we have opted to focus solely on those users who possess an Android device.

By narrowing our focus in this way, we aim to streamline our data analysis and provide more accurate insights into the usage patterns and behaviors of Android users. This approach will help ensure that our research is both targeted and effective, while minimizing the potential for data bias or confusion that could arise from analyzing both Android and IOS users together.

We have sub-selected the data from our dataset by filtering for users with an Android operating system.


In [3]:

```python
# Selecting only ANDROID Devices

device_list = []
for users in range(len(data)):
    device = {}
    if data[users]['device']['platform'] == 'ANDROID':
                   device_list.append(data[users])
```

This has resulted in the creation of a new list, referred to as device_list, which contains all the relevant user details for those who own an Android device.

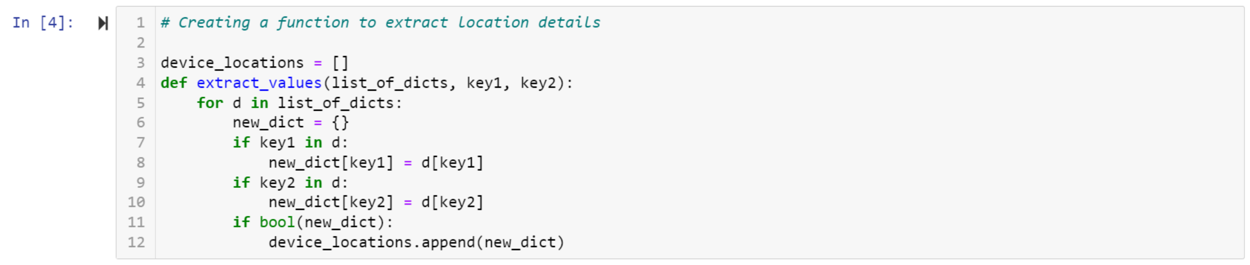
In order to draw meaningful econometric comparisons between mobile device ownership and travel destinations, we must further refine our data analysis. Specifically, we are now focusing on extracting three key categories of information: Location Details, Device Details, and Application Details. These three dataframes has one primary key which is common and is later used to join the dataframes.

By honing in on these specific data points, we can gain a more nuanced understanding of how mobile devices are utilized in the context of travel and the various applications that are being employed. This will enable us to draw more robust econometric conclusions and provide greater insights into the relationship between mobile device usage and travel destinations.

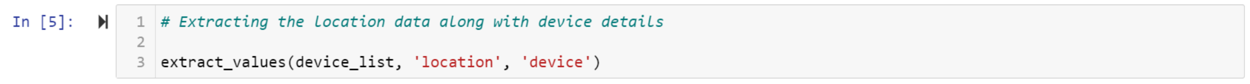
Location Details:

As each user in our dataset has visited multiple locations, this information is nested on the second level of our JSON file. To extract the relevant location and device details from each record of the JSON object, we have created a custom function.

To apply this function to the entire dataset, we need to read each line of the JSON file and extract the necessary key-value pairs. This can be achieved through the use of a for loop, which iterates through each line of the JSON file and applies the function to each record and stores the extracted value to a blank list.

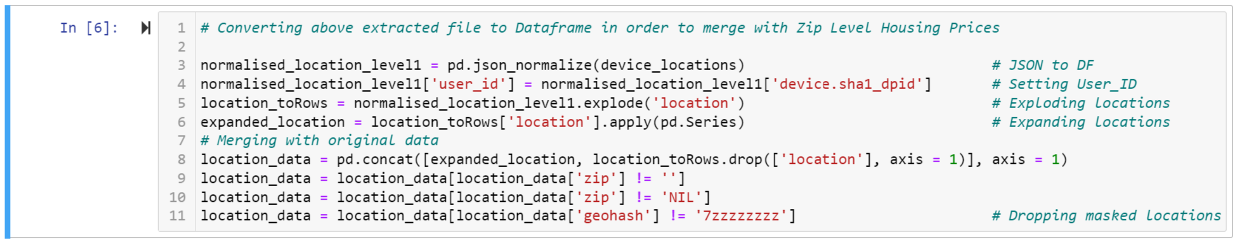


```
In [4]:

# Creating a function to extract location details

device_locations = []
def extract_values(list_of_dicts, key1, key2):
    for d in list_of_dicts:
        new_dict = {}
        if key1 in d:
            new_dict[key1] = d[key1]
        if key2 in d:
            new_dict[key2] = d[key2]
        if bool(new_dict):
            device_locations.append(new_dict)
```



```
In [5]:

# Extracting the location data along with device details

extract_values(device_list, 'location', 'device')
```

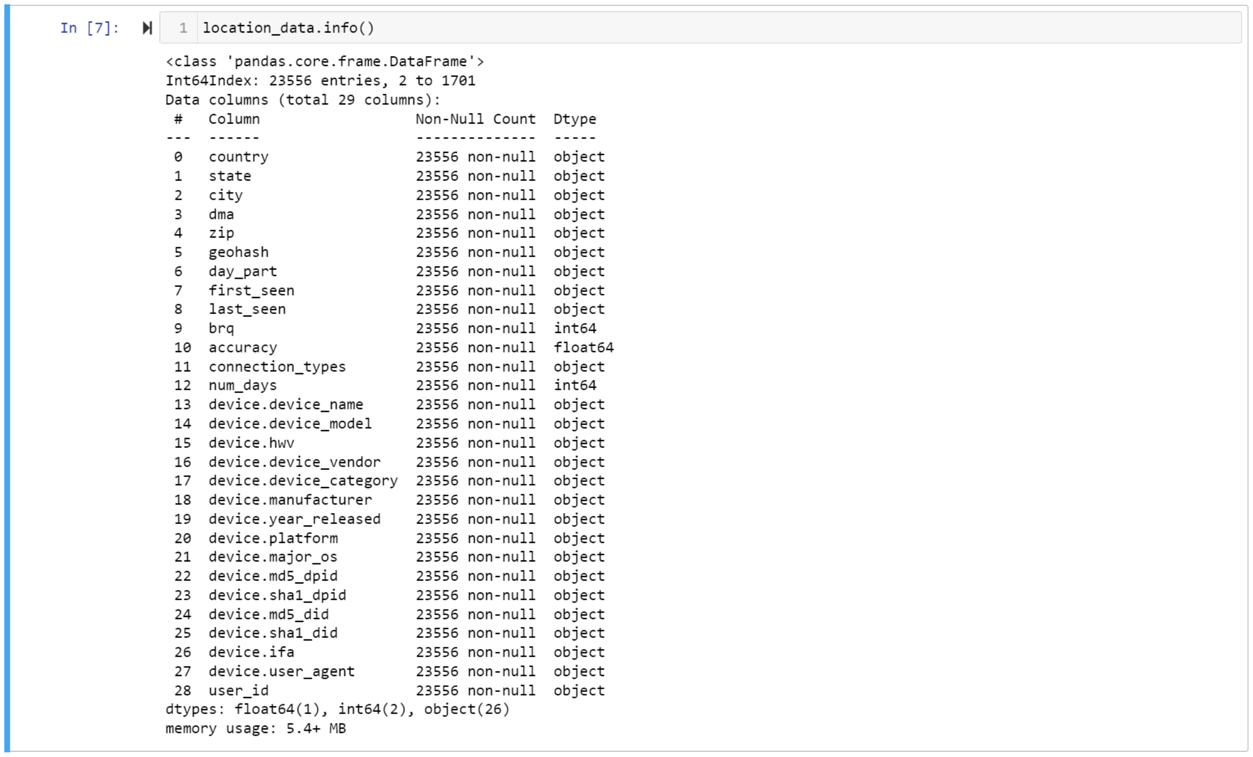
Once the location and device details are stored in device_loation list, we start creating a dataframe which will have columns related to locations e.g. Zip, Geohash, etc and device, e.g. Platform, Model Name, etc.



```
In [6]:

# Converting above extracted file to Dataframe in order to merge with Zip Level Housing Prices

normalised_location_level1 = pd.json_normalize(device_locations)                       # JSON to DF
normalised_location_level1['user_id'] = normalised_location_level1['device.sha1_dpid']       # Setting User_ID
location_toRows = normalised_location_level1.explode('location')                      # Exploding Locations
expanded_location = location_toRows['location'].apply(pd.Series)                      # Expanding Locations
# Merging with original data
location_data = pd.concat([expanded_location, location_toRows.drop(['location'], axis = 1)], axis = 1)
location_data = location_data[location_data['zip'] != '']
location_data = location_data[location_data['zip'] != 'NIL']
location_data = location_data[location_data['geohash'] != '7zzzzzzzz']                # Dropping masked Locations
```

During our analysis, we identified a number of geohashes within our dataset that had been masked, which we ultimately dropped from our analysis to ensure the accuracy and quality of our results.

Furthermore, we also observed instances where the values for certain data points were either NIL or blank. As these records provided no useful information for our analysis, we made the decision to drop them from our dataset as well.



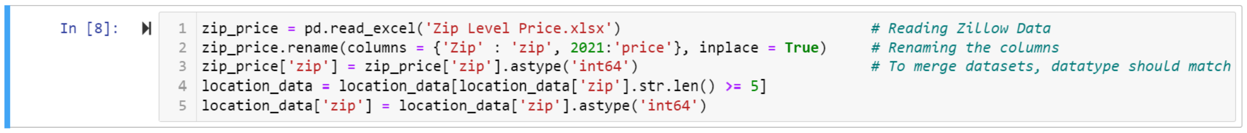
```
In [7]:  location_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23556 entries, 2 to 1701
Data columns (total 29 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   country                 23556 non-null  object 
 1   state                   23556 non-null  object 
 2   city                    23556 non-null  object 
 3   dma                     23556 non-null  object 
 4   zip                     23556 non-null  object 
 5   geohash                 23556 non-null  object 
 6   day_part                23556 non-null  object 
 7   first_seen              23556 non-null  object 
 8   last_seen               23556 non-null  object 
 9   brq                     23556 non-null  int64  
 10  accuracy                23556 non-null  float64
 11  connection_types        23556 non-null  object 
 12  num_days                23556 non-null  int64  
 13  device.device_name      23556 non-null  object 
 14  device.device_model     23556 non-null  object 
 15  device.hwv              23556 non-null  object 
 16  device.device_vendor    23556 non-null  object 
 17  device.device_category  23556 non-null  object 
 18  device.manufacturer     23556 non-null  object 
 19  device.year_released    23556 non-null  object 
 20  device.platform         23556 non-null  object 
 21  device.major_os         23556 non-null  object 
 22  device.md5_dpid         23556 non-null  object 
 23  device.sha1_dpid        23556 non-null  object 
 24  device.md5_did          23556 non-null  object 
 25  device.sha1_did         23556 non-null  object 
 26  device.ifa              23556 non-null  object 
 27  device.user_agent       23556 non-null  object 
 28  user_id                 23556 non-null  object 
dtypes: float64(1), int64(2), object(26)
memory usage: 5.4+ MB
```

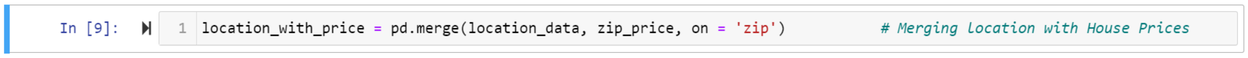
As part of our broader analysis, we sought to gain a deeper understanding of the economic value of the locations extracted from our dataset. To accomplish this, we obtained an additional dataframe from Zillow, leveraging their extensive research data to analyze the zip values associated with each of the locations in our dataset.

By merging this data with our existing dataset, we were able to gain a more complete understanding of the economic implications of mobile device usage in the context of travel patterns. This approach enabled us to draw more comprehensive and impactful conclusions from our analysis, providing greater insights into the underlying dynamics at play in this complex and rapidly-evolving space.

Zillow Home Value Index (ZHVI): A measure of the typical home value and market changes across a given region and housing type. It reflects the typical value for homes in the 35th to 65th percentile range. Available as a smoothed, seasonally adjusted measure and as a raw measure.

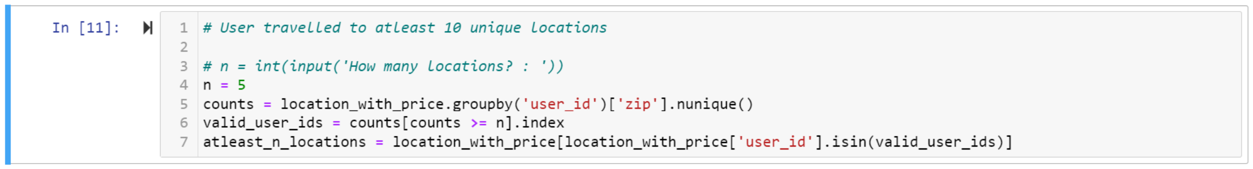


```
In [8]:  zip_price = pd.read_excel('Zip Level Price.xlsx')                          # Reading Zillow Data
         zip_price.rename(columns = {'Zip' : 'zip', 2021:'price'}, inplace = True)     # Renaming the columns
         zip_price['zip'] = zip_price['zip'].astype('int64')                          # To merge datasets, datatype should match
         location_data = location_data[location_data['zip'].str.len() >= 5]
         location_data['zip'] = location_data['zip'].astype('int64')
```



```
In [9]:  location_with_price = pd.merge(location_data, zip_price, on = 'zip')          # Merging Location with House Prices
```

In order to further refine our research and focus on the most relevant data points, we made the decision to analyze only those users who had traveled to at least 5 different locations. By setting this threshold, we were able to filter out users with minimal travel activity, allowing us to concentrate our analysis on individuals who had engaged in more substantial travel experiences.

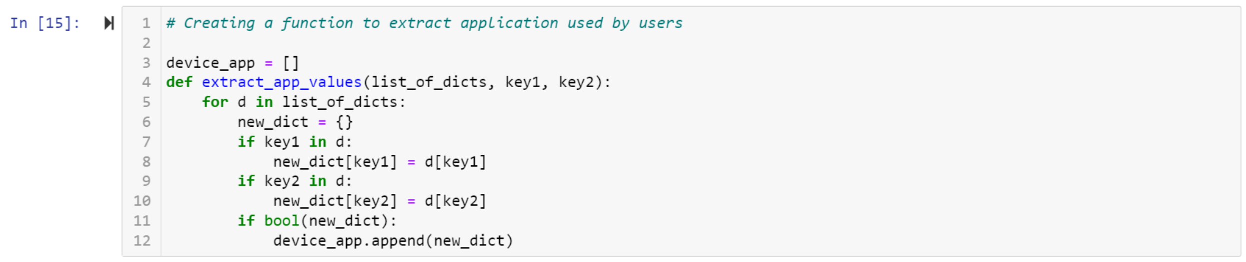


In [11]:

```python
# User travelled to atleast 10 unique locations

# n = int(input('How many locations? : '))
n = 5
counts = location_with_price.groupby('user_id')['zip'].nunique()
valid_user_ids = counts[counts >= n].index
atleast_n_locations = location_with_price[location_with_price['user_id'].isin(valid_user_ids)]
```

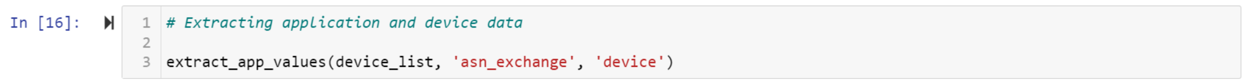
Application Data:

We have developed a new function that works similar to extract_location() and can extract information about the applications on a user's phone. This data is nested on the second level of our JSON file and may vary between different users.



In [15]:

```python
# Creating a function to extract application used by users

device_app = []
def extract_app_values(list_of_dicts, key1, key2):
    for d in list_of_dicts:
        new_dict = {}
        if key1 in d:
            new_dict[key1] = d[key1]
        if key2 in d:
            new_dict[key2] = d[key2]
        if bool(new_dict):
            device_app.append(new_dict)
```



In [16]:

```python
# Extracting application and device data

extract_app_values(device_list, 'asn_exchange', 'device')
```

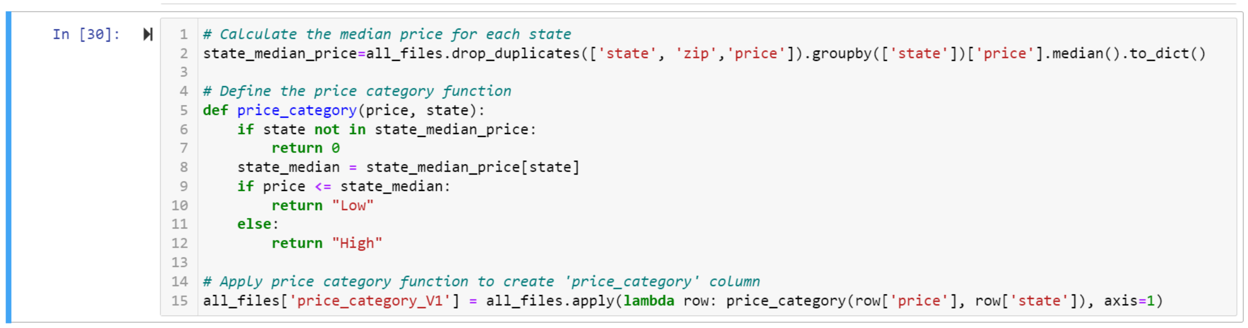
As we analyzed our data, we discovered that certain Application Bundle IDs were either empty or contained the value "NIL". Since these records did not contribute any valuable information to our analysis, we made the decision to remove them from our dataset.

Additionally, we set a criteria to only include users in our analysis who have a minimum of 5 applications installed on their phone.

We carried out the previously mentioned steps and as a result, we were able to extract three separate dataframes from each file. The first dataframe contained location data, the second contained application data, and the third contained device data. These dataframes were generated after dropping records with empty or "NIL" Application Bundle IDs and including only users who had at least 5 applications installed on their phone.

Median House Values:

In order to assess the economic impact of housing prices, we computed the median housing price for each state and then categorized each location as either "Low" or "High" based on this median value.

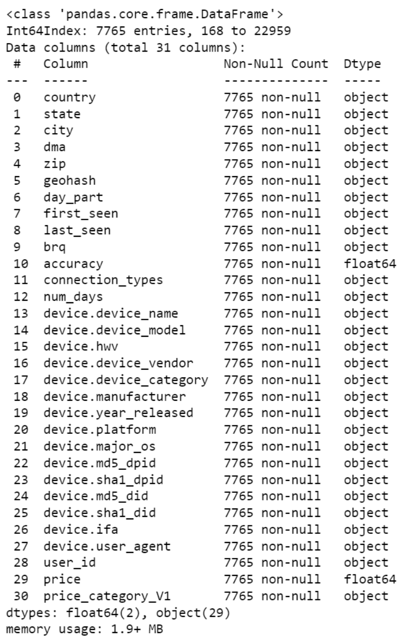


```
In [30]:
# Calculate the median price for each state
state_median_price=all_files.drop_duplicates(['state', 'zip','price']).groupby(['state'])['price'].median().to_dict()

# Define the price category function
def price_category(price, state):
    if state not in state_median_price:
        return 0
    state_median = state_median_price[state]
    if price <= state_median:
        return "Low"
    else:
        return "High"

# Apply price category function to create 'price_category' column
all_files['price_category_V1'] = all_files.apply(lambda row: price_category(row['price'], row['state']), axis=1)
```

This block of code calculates the median price of each state from the given data, and stores the result in a dictionary called state_median_price.

It then defines a function called price_category which takes two arguments, price and state. This function determines whether the given price is "Low" or "High" based on the median price of the state. If the state is not present in the state_median_price dictionary, the function returns 0.

Finally, the apply method is used to apply the price_category function to each row of the all_files dataframe and create a new column called price_category_V1 which stores the price category (i.e. "Low" "High") of each row based on the median price of the corresponding state.



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7765 entries, 168 to 22959
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   country                 7765 non-null   object 
 1   state                   7765 non-null   object 
 2   city                    7765 non-null   object 
 3   dma                     7765 non-null   object 
 4   zip                     7765 non-null   object 
 5   geohash                 7765 non-null   object 
 6   day_part                7765 non-null   object 
 7   first_seen              7765 non-null   object 
 8   last_seen               7765 non-null   object 
 9   brq                     7765 non-null   object 
 10  accuracy                7765 non-null   float64
 11  connection_types        7765 non-null   object 
 12  num_days                7765 non-null   object 
 13  device.device_name      7765 non-null   object 
 14  device.device_model     7765 non-null   object 
 15  device.hwv              7765 non-null   object 
 16  device.device_vendor    7765 non-null   object 
 17  device.device_category  7765 non-null   object 
 18  device.manufacturer     7765 non-null   object 
 19  device.year_released    7765 non-null   object 
 20  device.platform         7765 non-null   object 
 21  device.major_os         7765 non-null   object 
 22  device.md5_dpid         7765 non-null   object 
 23  device.sha1_dpid        7765 non-null   object 
 24  device.md5_did          7765 non-null   object 
 25  device.sha1_did         7765 non-null   object 
 26  device.ifa              7765 non-null   object 
 27  device.user_agent       7765 non-null   object 
 28  user_id                 7765 non-null   object 
 29  price                   7765 non-null   float64
 30  price_category_V1       7765 non-null   object 
dtypes: float64(2), object(29)
memory usage: 1.9+ MB
```

Conclusion & Future Work:

While our research has provided valuable insights into the relationship between mobile applications and customer location data, there are several areas for future exploration and investigation. One important avenue for further research is the development of more sophisticated analytical models and tools, which can enable more nuanced and granular analysis of the relationship between mobile applications and customer behavior.

Another area of future work is the expansion of the dataset to include a wider range of geographies and demographics, in order to identify regional or demographic variations in mobile app usage and travel patterns. This could include exploring the relationship between mobile application usage and other economic variables, such as income or employment status, as well as cultural or social factors that may influence consumer behavior.

Additionally, further research is needed to address the potential ethical implications of collecting and analyzing location data, particularly in the context of privacy and data security concerns. This could include exploring alternative data collection and anonymization techniques, as well as evaluating the effectiveness of existing regulations and policies in protecting consumer privacy.

By continuing to explore these and other related topics, we can build upon the insights gained from this research and continue to uncover new and valuable information about the relationship between mobile applications and customer location data.

References:

[1] Bolton, R. N. 1998. “A Dynamic Model of the Duration of the Customer’s Relationship with a Continuous Service Provider: The Role of Satisfaction,” Marketing Science (17:1), pp. 45-65.

[2] Rabinovich, E., House, L., & Bell, B. (2015). Using mobile apps to capture location data: An empirical study and practical implications. Journal of Interactive Marketing, 31, 50-61.

[3] Zhang, S., Zhu, T., & Li, J. (2019). The economic value of location data in mobile advertising. Information & Management, 56(1), 27-37.

[4] Klauser, K. (2018). The value of location data for businesses. Harvard Business Review. Retrieved from https://hbr.org/2018/09/the-value-of-location-data-for-businesses

[5] Chen, M., Li, X., Li, Q., Liu, J., & Wang, Y. (2021). The effects of mobile app usage on consumer travel behavior: An empirical study based on location-based services. Transportation Research Part C: Emerging Technologies, 128, 103140.

[6] Fagan, M. (2020). Location-based marketing: Unlocking the potential of GPS technology. Journal of Digital & Social Media Marketing, 8(4), 387-399.

Appendix I

Python Code:

```
# Importing the necessary libraries for our script

import json
import pandas as pd
import numpy as np

# Loading our file

file_name = part-xxxxx
# Taking the input file
data = [json.loads(line) for line in open(file_name, 'r', encoding = 'utf-8')]

# Selecting only ANDROID Devices
device_list = []
for users in range(len(data)):
    device = {}
    if data[users]['device']['platform'] == 'ANDROID':
                   device_list.append(data[users])

# Creating a function to extract location details
device_locations = []
def extract_values(list_of_dicts, key1, key2):
    for d in list_of_dicts:
        new_dict = {}
        if key1 in d:
            new_dict[key1] = d[key1]
        if key2 in d:
            new_dict[key2] = d[key2]
        if bool(new_dict):
            device_locations.append(new_dict)

# Extracting the location data along with device details

extract_values(device_list, 'location', 'device')

# Converting above extracted file to Dataframe in order to merge with Zip Level
Housing Prices
```

```
normalised_location_level1 = pd.json_normalize(device_locations)
# JSON to DF
normalised_location_level1['user_id'] = normalised_location_level1['device.sha1_dpid']
# Setting User_ID
location_toRows = normalised_location_level1.explode('location')
# Exploding locations
expanded_location = location_toRows['location'].apply(pd.Series)
# Expanding locations
location_data = pd.concat([expanded_location, location_toRows.drop(['location'], axis
= 1)], axis = 1)       # Merging with original data
location_data = location_data[location_data['zip'] != '']
location_data = location_data[location_data['zip'] != 'NIL']
location_data = location_data[location_data['geohash'] != '7zzzzzzzz']
# Dropping masked locations

location_data.info()

zip_price = pd.read_excel('Zip Level Price.xlsx')
# Reading Zillow Data
zip_price.rename(columns = {'Zip' : 'zip', 2021:'price'}, inplace = True)
# Renaming the columns
zip_price['zip'] = zip_price['zip'].astype('int64')
# In order to merge two data, we need both the columns to have same datatype
location_data = location_data[location_data['zip'].str.len() >= 5]
location_data['zip'] = location_data['zip'].astype('int64')

location_with_price = pd.merge(location_data, zip_price, on = 'zip')
# Merging location with House Prices

location_with_price = location_with_price.dropna(subset = ['price'])
# Dropping locations with house prices as NULL

# User traveled to at least 10 unique locations

# n = int(input('How many locations? : '))
n = 5
counts = location_with_price.groupby('user_id')['zip'].nunique()
valid_user_ids = counts[counts >= n].index
atleast_n_locations =
location_with_price[location_with_price['user_id'].isin(valid_user_ids)]

atleast_n_locations.info()

try:
    atleast_n_locations = atleast_n_locations[['user_id', 'country', 'state', 'city',
'dma', 'zip', 'county', 'geohash',
                                               'price', 'device.device_name',
'device.device_model', 'device.hwv', 'device.device_vendor',
                                               'device.device_category',
'device.manufacturer', 'device.year_released',
                                               'device.platform', 'device.major_os',
'device.md5_dpid',
                                               'device.sha1_dpid', 'device.md5_did',
'device.sha1_did', 'device.ifa',
                                               'device.user_agent']]
except KeyError:
    pass  # or you can print a message here

# Creating a function to extract application used by users

device_app = []
def extract_app_values(list_of_dicts, key1, key2):
    for d in list_of_dicts:
        new_dict = {}
        if key1 in d:
            new_dict[key1] = d[key1]
        if key2 in d:
            new_dict[key2] = d[key2]
        if bool(new_dict):
            device_app.append(new_dict)

# Extracting application and device data

extract_app_values(device_list, 'asn_exchange', 'device')

# Converting the json file to dataset (csv)

normalised_app_level1 = pd.json_normalize(device_app)
normalised_app_level1['user_id'] = normalised_app_level1['device.sha1_dpid']
app_data =
normalised_app_level1[normalised_app_level1['user_id'].isin(atleast_n_locations['user_
id'])]
app_toRows = app_data.explode('asn_exchange')
expanded_apps = app_toRows['asn_exchange'].apply(pd.Series)
device_apps = pd.concat([expanded_apps, app_data.drop(['asn_exchange'], axis = 1)],
axis = 1)
device_apps_cleaned = device_apps[device_apps['bundle'] != '']
# Dropping blank bundles
device_apps_cleaned = device_apps_cleaned[device_apps_cleaned['bundle'] != 'NIL']
device_apps_cleaned = device_apps_cleaned[device_apps_cleaned['device.device_name'] !=
'NIL']

# Subsetting based on minimum n apps
# n = int(input('How many apps? : '))
n = 5
counts = device_apps_cleaned.groupby('user_id')['asn'].nunique()
valid_user_ids = counts[counts >= n].index
atleast_n_apps =
device_apps_cleaned[device_apps_cleaned['user_id'].isin(valid_user_ids)]

# Preparing device dataset

device_data = atleast_n_apps[['device.device_name', 'device.device_model',
       'device.hwv', 'device.device_vendor', 'device.device_category',
       'device.manufacturer', 'device.year_released', 'device.platform',
       'device.major_os', 'device.md5_dpid', 'device.sha1_dpid',
       'device.md5_did', 'device.sha1_did', 'device.ifa', 'device.user_agent']]
device_data_cleaned = device_data.drop_duplicates()
device_data_cleaned

atleast_n_apps_n_locations =
atleast_n_locations[atleast_n_locations['user_id'].isin(atleast_n_apps['user_id'])]

atleast_n_apps_n_locations['user_id'].nunique()

atleast_n_apps.to_csv(file_name + '_' + 'Atleast_5_apps.csv')
atleast_n_apps_n_locations.to_csv(file_name + '_' +
'Atleast_5_applications_5_locations.csv')
device_data_cleaned.to_csv(file_name + '_' + 'Device_data.csv')

all_locations_df = pd.DataFrame(columns = atleast_n_apps_n_locations.columns)

all_locations_df = pd.concat([atleast_n_apps_n_locations, all_locations_df], axis = 0)

all_locations_df.to_csv('all_locations.csv')

all_locations_df.info()

all_files = all_locations_df

# Calculate the median price for each state
state_median_price=all_files.drop_duplicates(['state',
'zip','price']).groupby(['state'])['price'].median().to_dict()

# Define the price category function
def price_category(price, state):
    if state not in state_median_price:
        return 0
    state_median = state_median_price[state]
    if price <= state_median:
        return "Low"
    else:
        return "High"

# Apply price category function to create 'price_category' column
all_files['price_category_V1'] = all_files.apply(lambda row:
price_category(row['price'], row['state']), axis=1)

# Only keep the states with multiple zip (i.e., more than one)
temp = all_files.drop_duplicates(['state', 'zip','price'])
temp = temp[['state','city','zip','price','price_category_V1']].sort_values('state') #
sort by state name

# Count number of unique zips in each state
zip_count_statewise =
temp['state'].value_counts().rename_axis('state').reset_index(name = 'counts')

# States that have at least more than one zip (in this case at least three)
states_tobe_retained = zip_count_statewise.loc[(zip_count_statewise['counts'] >=
3)]['state']
states_tobe_retained

# Final data of all files with filtered states
all_files = all_files.loc[all_files['state'].isin(list(states_tobe_retained))]

all_files.to_csv("location_with_housing_price_category.csv")
# Extracting records with locations and device details

all_files.info()
```

Appendix II

All the files are stored in an on premise cloud infrastructure which is USF's own research cluster. An individual needs to raise a request via his/her supervisor. Once approved, the user will get the credentials to access the storage via Linux environment only. MobaXterm is used in our case.

Link to raise a request: https://wiki.rc.usf.edu/index.php/Main_Page

Once the user has access to the storage, below steps are needed to be followed to fetch the data:

1) Install MobaXterm from https://mobaxterm.mobatek.net/.
2) Follow the steps mentioned on this link to understand how to access the cluster.
3) Once the user gets access, the user should use the below details to connect to SSH via MobaXterm:
    - Your USF NetID and Password
    - Hostname: circe.rc.usf.edu
    - SSH Port: 22 (This is the default)
4) Once the above steps are followed the user will have his/her own environment (local), where miniconda, python, and required packages need to be installed. Please see below:
    - **Miniconda** - bash Miniconda3-latest-Linux-x86_64.sh
    - **Packages** - pip install pandas | pip install numpy
5) The big data resides in Professor Dutta's directory, use the below path to access the needed files from professor's directory:

   /rcfs/workprojects/dutta/ad-data/.

Appendix III

```
# Get the name of the Python script to run
python_script="App-Location-Zip.py"

# Loop over the file names in the range part-00022 to part-00069
for file_name in $(seq -f "part-%05g" {start} {end})
do
  # Print a message indicating which file is being processed
  echo "Processing file: $file_name"

  # Run the Python script with the current file name as an argument
  python "$python_script" "$file_name"
Done
```