Guilty Pleasures After Dark - Where to Set Up Your Eatery in Night-Time Sydney

A Data-Driven Case Study by Ossama Mughal

Introduction to Applied Data Science Capstone

This project will use location data providers to solve a business problem. The provided data will be wrangled, processed, and applied to machine learning models to effectively derive patterns and generate answers to the issue. The data science methodology will be implemented in this project, and practical data analysis conducted to reach a conclusion.

Table of Contents

1. Introduction/Business Problem

Since the Sydney Lockout Laws Legislation was introduced in 2014, the night-time economy of Australia's most famous city has become an interesting subject of study. Enforcing 1.30AM lockouts and 3AM last drinks at bars, pubs, and clubs in Sydneys entertainment precincts (such as Kings Cross) to deter alcohol- and drug-fuelled violence, these businesses have suffered a decrease in trade and demand. Given pedestrian traffic dropped by 40% in Kings Cross, falling from a Saturday peak of 5,590 per hour between 1AM and 2AM in 2010, to a Saturday peak of 3,888 between 12AM and 1AM in 2015, other night-time facility types, such as eateries, have a greater pressure to pick up the fall. A Deloitte Access Economics report, called ImagineSydney: Play, examines the city's night-time economy is underperforming and losing $16bn, further stating 'a number of sectors would need to expand their night-time services, ranging from restaurants and bars to arts and culture, entertainment and fitness centres'.

A greater demand for these other services creates an opportunity for increased market share. Investors, interested particularly in rising fads of international desserts such as bubble tea, can capitalise on the dynamic shift in Sydney's night-time activities by assessing the factors that have contributed to successful night-time restaurants/cafes. Indeed, factors such as parking space proximity, public transport accessibility, and proximity to existing restaraunts with a brand name, are now critical to a eatery businesses success in light of the Lockout Laws and the decrease in accessibility of these facilities (eg. less frequent trains, more expensive parking during lockout period etc.). It is then critical that an investor is able to discern patterns in these details pertaining to existing night-time eateries, and apply this to predict the optimal locations in setting up the business that consider it successful.

The **business problem**, then, is defined as determining the ideal location(s) to set up a successful night-time eatery, based on factors including private/public transport accessibility, pre-existing successful eatery street locations, and general facility features (eg. Wi-Fi, outdoor seating etc.). Foursquare's location API services will be utilised to obtain late-night venue details, such as coordinates, ratings, and restaurant features, and investigate these details for patterns and classifying. The **stakeholder** of this business problem can be identified as an investor attempting to capitalise on a night-time economy shifting demand to more eateries, and opening a new eatery that is successful and based primarily on its ease of access and distance from other competition.

2. Data

On a functional perspective, the data used to solve the business problem can be categorised into 3 areas:

1. Late-Night Sydney Venues Details
2. Parking Facility (Private Transport) Details
3. Public Transport Details

The sub-sections below briefly detail these data areas from a technical perspective, as well as examples of the datasets.

Late-Night Sydney Venues Details

Two Foursquare Developer API's will be used to retrieve details about Sydney's venues in the late-night, eatery category. The first API will search for venues, with request parameters defining a query for Sydney CBD's coordinates (33°52′5″S 151°12′44″E) with a radius of 837 metres, and retrieving venue details in JSON format as response, such as name, coordinates, restaurant features, etc. The second API will provide details on venues, with request parameters defining the venue ID (obtained from the first API response), and retrieving venue details in JSON format as response, such as rating, hours, price level etc.

The response fields that will be used for this business problem will be stored in a dataframe, and are as follows:

API response attribute	Field Name	Field Description/ Relevance
id	Venue ID	The Foursquare ID of the venue
name	Venue Name	The venue name
categories	Venue Category	The venue categories. This field will be filtered for late-night eateries.
hasPerk	Venue Feature Indicator	Indicates whether the venue contains features (eg. Wi-Fi, outdoor seating). This field will help interpret the significance of restaurant perks in an eatery's success alongside accessibility.
location.address	Venue Address	The street address of the venue
location.lat	Venue Latitude Coordinate	The latitude coordinate of the venue
location.lng	Venue Longitude Coordinate	The longitude coordinate of the venue
location.state	Venue State	The state of the venue
location.country	Venue Country	The country of the venue
location.formattedAddress	Venue Formatted Address	The overall address of the venue (ie. including state and country)
delivery.id	Delivery Vendor ID	The Foursquare ID of the delivery vendor. This field will be transformed to a boolean value, and used to help interpret it's significance as a restaurant perk in an eatery's success alongside accessibility
rating	Venue Rating	The rating of the venue, ranging from 0-10, as chosen by Foursquare users. This will be used as a predictor variable, and once cluster groups are developed, used to calculate it's average in each group and hence identify specific clusters as those representing the 'successful eatery' group.

These properties will be used in conjunction with private/public transport data to train and test the classification model, and predict the locations for new eateries based on these instances.

An example dataframe of one restaurant's details, retrieved using the Foursquare Search For Venues API and processed for attributes of interest, are output below:

In [60]:

```
#Import relevant libraries
import requests # library to handle requests
import json #library to handle json files
import pandas as pd # library for data analsysis
import numpy as np # library to handle data in a vectorized manner
from pandas.io.json import json_normalize # tranforming json file into a pandas dataframe library

#Set request parameter values
search_query = 'Bay Vista Gateau'
radius = 500
CLIENT_ID = 'HBUXUQEAE3GOZWGZP02LRNRJ2HY52DUWQSHNCLNYJCL3JZYP' # your Foursquare ID
CLIENT_SECRET = '14PBFW41D5HWAZNWT2WC4NNHS3CGVXEWDYAWWTN4H0BEGRKF' # your Foursquare Secret
latitude = -33.961275
longitude = 151.155991
VERSION = '20180604'
LIMIT = 30

url = 'https://api.foursquare.com/v2/venues/search?client_id={}&client_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.format(CLIENT_ID, CLIENT_SECRET, latitude, longitude, VERSION, search_query, radius, LIMIT)
```

```
# call API and store response values
results_venues = requests.get(url).json()

# assign relevant part of JSON to venues
venues = results_venues['response']['venues']

# tranform venues into a dataframe
df_venues = json_normalize(venues)

# filter columns for relevant fields
filtered_col_venues = ['id', 'name', 'categories','hasPerk','location.address','location.lat','loca
tion.lng','location.state','location.country','location.formattedAddress','delivery.id','location.c
rossStreet']
df_venues_filtered = df_venues.loc[:, filtered_col_venues]

df_venues_filtered
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/pandas/core/indexing.py:1418:
FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#deprecate-loc-reindex-
listlike
  return self._getitem_tuple(key)
```

Out[60]:

	id	name	categories	hasPerk	location.address	location.lat	location.lng	location.state
0	4b05875ef964a5201b8e22e3	Bay Vista Gateau	[{'id': '4bf58dd8d48988d1d0941735', 'name': 'D...	False	83 The Grand Pde.	-33.961268	151.156108	NSW

The ID of the venue obtained here will be used for the Venue Details API to retrieve the venue's rating. This data will be joined to the dataframe above to be treated as the dependent variable.

Usage

This dataset will have it's **categories** column filtered to only show venues that are open late-night in Sydney CBD, and are considered eateries under various labels.

The venue coordinates will be calculated in proximity to private/public transport, available in the respective datasets, which will be flagged if within a certain radius of the venue and included in a count with respect to the venue. Proximity to other venues, that claim to be competition, will also be calculated. It is assumed that competing venues operating on the same street are direct competition.

The **hasPerk** and **delivery.id** values will be applied against OR logic, as either one is treated as a restaurant feature for the sake of the business problem.

The value of using this dataset is two-fold: The venues will be classified using their details to showcase dissimilarities, and will enable a correlation analysis of private/public transport accessibility to the success (ie. rating) of a venue. The venues in each cluster can then be investigated for their dataset information to understand how each classification is dissimilar, and ultimately form a conclusion on ideal locations to set up a new late-night eatery.

Parking Facility (Private Transport) Details

An OpenData Transport for NSW (tfNSW) API will be used to retrieve details about Sydney's off-street parking provided by major parking providers for over 20,000 off-street parking spaces. The retrieved parking details will be in JSON format as response, such as building name, coordinates, total number of bays, address, etc. Given the API response is constricted to a compressed file content type and has a limit of 5 calls a day, the GeoJSON file has been directly uploaded to the GitHub repository for simplicity, and accessed using the *urllib* library.

The response fields that will be used for this business problem will be stored in a dataframe, and are as follows:

API response attribute	Field Name	Field Description/Relevance
geometry.coordinates	Parking Space Coordinates	The latitude and longitude coordinates of the parking space

properties.FRD_ID	Parking Space ID	The tfNSW ID for the parking space
properties.Building_name_location	Parking Space Building Name	The name of the CBD building associated with the parking space.
properties.Street_Number_GPS	Parking Space Street Number	The street number of the parking space
properties.Street_Name_GPS	Parking Space Street Name	The street name of the parking space
properties.Suburb_GPS	Parking Space Suburb	The suburb of the parking space
properties.State	Parking Space State	The state of the parking space
properties.Postcode	Parking Space Postal Code	The postal code of the parking space
properties.Total_number_of_bays	Total Number of Parking Space Bays	The number of total bays (unavailable/available) in the parking space. A larger space may compensate for the parking space location being slightly further than the eatery, and hence maintain accessibility for the individual. Therefore, this data field will be considered for training the clustering model.

These properties will be used in conjunction with public transport data and late-night eatery venue data to train and test the classification model, and classify the venues into groups based on the instances. Hence, this data will be critical in assessing the ideal locations to place a new late-night eatery.

An example dataframe of some parking spaces details, retrieved using the GeoJSON file provided by the OpenData tfNSW API and processed for attributes of interest, are output below:

In [61]:

```
import urllib.request, json

with urllib.request.urlopen('https://raw.githubusercontent.com/ozzie-
mughal/Coursera_Capstone/master/OffstreetparkingData.geojson') as geojson_offstreetparking:
    results_offstreetparking = json.loads(geojson_offstreetparking.read().decode())

# assign relevant part of JSON to venues
offstreetparking = results_offstreetparking['features']

# tranform venues into a dataframe
df_offstreetparking = json_normalize(offstreetparking)

# filter columns for relevant fields
filtered_col_offstreetparking =
['geometry.coordinates','properties.FRD_ID','properties.Building_name_location','properties.Street_
Number_GPS','properties.Street_Name_GPS','properties.Suburb_GPS','properties.State','properties.Pos
tcode','properties.Total_number_of_bays']
df_offstreetparking_filtered = df_offstreetparking.loc[:,filtered_col_offstreetparking]

df_offstreetparking_filtered.head()
```

Out[61]:

	geometry.coordinates	properties.FRD_ID	properties.Building_name_location	properties.Street_Number_GPS	properties.Street_Name_G
0	[151.212657, -33.86218]	FRD-001	93 Macquarie Street	2	Alber
1	[151.209166, -33.861878]	FRD-002	Gold Fields House	18	Pit
2	[151.209767, -33.8644]	FRD-003	No. 1 OConnell St Car Park	3	Ben
3	[151.211869, -33.865398]	FRD-004	The Chifley Tower	27	Ben
4	[151.2107, -33.865404]	FRD-005	Sofitel Wentworth Hotel	15	Bligh

Usage

This dataset will have it's coordinates column split, and adjusted to represent latitude and longitude values appropriately.

These parking space coordinates will finally be calculated in proximity to each venue available in the venue dataset, and flagged if within a certain radius. A categorical variable will be assigned to the venue dataset, and the proximity calculations transformed to this variable (eg. Very accessible if number of parking spaces within radius > 10). The variable will be one-hot encoded to prepare the dataset for classification model data infeed.

The value of using this dataset is two-fold: It will provide an effective predictor for grouping venues into classes under the assumption that more successful venues are more accessible to parking spaces, and will enable a correlation analysis of parking space accessibility to the success (ie. rating) of a venue.

Public Transport Details

An OpenData Transport for NSW (tfNSW) API will be used to retrieve details about Sydney's public transport locations, ranging across train stations, ferry wharves, and bus interchanges. Furthermore, information on associated facilities (eg. bicycle racks, commuter car parks) will also be available. The retrieved transport location details will be in JSON format as response, such as types of transport, coordinates, facilities available, address, etc. Given the API response is constricted to a compressed file content type and has a limit of 5 calls a day, the CSV file has been directly uploaded to the GitHub repository for simplicity, and accessed using the *pandas* library.

The response fields that will be used for this business problem will be stored in a dataframe, and are as follows:

API response attribute	Field Name	Field Description/Relevance
LOCATION	Public Transport Location Name	Location name of the Station/Wharf/Light Rail/Bus Interchange
TSN	Transit Stop Number	Transit Stop Number (TSN) ID for each stop at the Station/Wharf/Light Rail/Bus Interchange
X_COORD	Public Transport Location Longitude	Longitude coordinate of the Station/Wharf/Light Rail/Bus Interchange
Y_COORD	Public Transport Location Latitude	Latitude coordinate of the Station/Wharf/Light Rail/Bus Interchange
ADDRESS	Public Transport Location Street Name	Street Name of the Station/Wharf/Light Rail/Bus Interchange
TRANSPORT	Public Transport Type(s)	The type of transport mode(s) available at the location

These properties will be used in conjunction with private transport data and late-night eatery venue data to train and test the classification model, and classify the venues into groups based on the instances. Hence, this data will be critical in assessing the ideal locations to place a new late-night eatery.

An example dataframe of some public transport location details, retrieved using the CSV file provided by the OpenData tfNSW API and processed for attributes of interest, are output below:

In [62]:

```
results_publictransport = pd.read_csv('https://raw.githubusercontent.com/ozzie-
mughal/Coursera_Capstone/master/LocationFacilitiesData.csv')
results_publictransport.drop(['EFA_ID','FACILITIES','ACCESS','PHONE'],axis=1,inplace=True)
results_publictransport.head()
```

Out[62]:

	LOCATION_NAME	TSN	X_COORD	Y_COORD	ADDRESS	TRANSPORT
0	Aberdeen Station	233610	150.892052	-32.167104	New England Hwy, Aberdeen	Train
1	Adamstown Station	228920	151.720081	-32.933730	Park Ave, Adamstown	Train|Bus
2	Albion Park Station	252710	150.798500	-34.562647	Princes Hwy, Albion Park	Train|Bus
3	Allawah Station	222020	151.114330	-33.969584	Railway Pde, Allawah	Train|Bus
4	Arncliffe Station	220520	151.147300	-33.936456	Firth St, Arncliffe	Train|Bus|Taxi rank

Usage

This dataset will have it's tuples split to be unique to a single mode of transport. Furthermore, the coordinates columns will be adjusted to represent latitude and longitude values appropriately.

These transport location coordinates will finally be calculated in proximity to each venue available in the venue dataset, and flagged if within a certain radius. A categorical variable will be assigned to the venue dataset, and the proximity calculations transformed to this variable (eg. Very accessible if number of public transport within radius > 10). The variable will be one-hot encoded to prepare the dataset for classification model data infeed.

The value of using this dataset is two-fold: It will provide an effective predictor for grouping venues into classes under the assumption that more successful venues are more accessible to public transport, and will enable a correlation analysis of public transport accessibility to the success (ie. rating) of a venue.

3. Methodology

3.1 Introduction

This section provides a walkthrough of the exploratory data analysis conducted to solve the business problem of predicting ideal locations for a new late-night eatery. The data used is loaded, visualised and processed to prepare it for machine learning modelling, including feature extraction/selection. Furthermore, several classification models are are trained and evaluated with respect to optimal parameters to use. These models are compared via an accuracy report using various scoring methods, and the best model selected. This model is finally executed to predict the most ideal locations for a new eatery by selecting the street names of test data tuples that predict a successful rating for a hypothetical eatery on that street.

The machine learning models used are of classification type. This is because the data provided is labelled and the model can be supervised during training to have defined independent and dependent variables. Namely, the fields detailing venue location details and private/public transport accessibility details are treated as independent variables that govern the rating of the eatery (ie. the dependent variable). The classification models that will be generated and evaluated are of the following:

- K-Nearest Neighbours Model
  - Justification: Simple implementation, robust with regard to the search space; for instance, classes don't have to be linearly separable, can be updated at very little cost as new instances with known classes are presented.
- Decision Tree Model
  - Justification: Robust to outliers, scalable, and able to naturally model non-linear decision boundaries due to their hierarchical structure.
- Support-Vector Machine Model
  - Can model non-linear decision boundaries, with many kernels to choose from. Quite robust against overfitting, especially in high-dimensional space.
- Logistic Regression Model
  - Justification: Outputs have a nice probabilistic interpretation, and the algorithm can be regularized to avoid overfitting. Logistic models can be updated easily with new data using stochastic gradient descent

The accuracy scores that will be used to evaluate and compare these classification models are as follows:

- Jaccard Similarity Index
- F1-Score
- Logarithmic Loss

3.2 Library & Data Load

The libraries and datasets are loaded for each group of information identified in Section 2, via API GET methods to obtain results in either JSON or CSV format.

3.2.1 Library Load

In [63]:

```
#pip install geopy #enable for first run, comment out after
```

In [64]:

```
# Data Scraping and Loading
import requests # library to handle requests
import json #library to handle json files
import pandas as pd # library for data analsysis
import numpy as np # library to handle data in a vectorized manner
from pandas.io.json import json_normalize # tranforming json file into a pandas dataframe library
import urllib.request, json # library to handle URL requests

```

```
# Data file processing
from sklearn import preprocessing # library to handle data preprocessing
from sklearn.preprocessing import MinMaxScaler # library to normalised values within a given range
from geopy import distance #import distance calculator from geopy library

# Data Classification
from sklearn.model_selection import train_test_split #to split data for training and testing of cl
assifier models
from sklearn.neighbors import KNeighborsClassifier # library to generate K-Nearest Neighbors class
ifier model
from sklearn.tree import DecisionTreeClassifier # library to generate Decision Tree classifier mod
el
from sklearn import svm # library to generate Support Vector Machine classifier model
from sklearn.linear_model import LogisticRegression # library to generate Logistic Regression clas
sifier model
from sklearn import metrics # library to generate model accuracy scores
from scipy.stats import pearsonr # library to calculate correlation between variables

#Classification Accuracy Scoring
from sklearn.metrics import jaccard_similarity_score # library to create Jaccard score
from sklearn.metrics import f1_score # library to create F1 Score
from sklearn.metrics import log_loss # library to create logarithmic loss score
```

3.2.2 Late-Night Sydney Venues Data Load

Important Note: This API call only allows a limit of 50 venues search results. The model will therefore be limited by this number of samples.

In [65]:

```
#Search for Venues API GET

#Set request parameter values
radius = 837
CLIENT_ID = 'HBUXUQEAE3GOZWGZP02LRNRJ2HY52DUWQSHNCLNYJCL3JZYP' # your Foursquare ID
CLIENT_SECRET = '14PBFW41D5HWAZNWT2WC4NNHS3CGVXEWDYAWWTN4H0BEGRKF' # your Foursquare Secret
latitude = -33.866685
longitude = 151.201458
intent = 'browse'
VERSION = '20200101'
#LIMIT = 49 #this is the maximum limit value available for a FourSquare Developer account

url = 'https://api.foursquare.com/v2/venues/search?client_id={}&client_secret={}&ll={},{}&v={}&rad
ius={}&intent={}'.format(CLIENT_ID, CLIENT_SECRET, latitude, longitude, VERSION, radius, intent)

# call API and store response values
results_venues = requests.get(url).json()

# assign relevant part of JSON to venues
venues = results_venues['response']['venues']

# tranform venues into a dataframe
df_venues = json_normalize(venues)

# A preview of the results for the first API call
print("There are ",df_venues.shape[0]," eateries found in this Sydney CBD area.")
df_venues.head()
```

```
There are  30  eateries found in this Sydney CBD area.
```

Out[65]:

	id	name	categories	referralId	hasPerk	location.lat	location.lng	location.labele
0	52c7acf8498ec0824440cd06	Georges Mediterranean Bar&Grill (Sydney, NSW)	[{'id': '4bf58dd8d48988d10e941735', 'name': 'G...	v-1578838812	False	-33.866690	151.201338	[{'label': 'disp 33.866
1	4b1a411cf964a52071e823e3	Steersons Steakhouse	[{'id': '4bf58dd8d48988d1cc941735', 'name': 'S...	v-1578838812	False	-33.866674	151.201425	[{'label': 'disp 33.866673
		Georges	[{'id':	v-				[{'label': 'disp

	id	name	categories	referralId	hasPerk	location.lat	location.lng	location.labele
2	4b2dd2b8f964a5208cdb24e3	Mediterranean Bar & Grill	'4bf58dd8d48988d1c0941735', 'name': 'M...	1578838812	False	-33.866665	151.201309	33.866664
3	5cf474dd0e3239002b94780e	Blue Oasis	[{'id': '4bf58dd8d48988d1c4941735', 'name': 'R...	v-1578838812	False	-33.866763	151.201572	[{'label': 'disp 33.866763
4	4b8dcde9f964a520690e33e3	Darling Harbour Ferry Wharf	[{'id': '4e74f6cabd41c4836eac4c31', 'name': 'P...	v-1578838812	False	-33.866927	151.200822	[{'label': 'disp 33.866927

In [66]:

```python
#Get Details of a Venue API GET

#Set request parameter values
venue_id_query = ''
CLIENT_ID = 'HBUXUQEAE3GOZWGZP02LRNRJ2HY52DUWQSHNCLNYJCL3JZYP' # your Foursquare ID
CLIENT_SECRET = '14PBFW41D5HWAZNWT2WC4NNHS3CGVXEWDYAWWTN4H0BEGRKF' # your Foursquare Secret
VERSION = '20180604'

# create venues ratings and ID list
results_venues_ratings = pd.DataFrame(columns=['id','rating'])

#loop through venue ID's, obtained in Search for Venues API response, to retrieve venue ratings
late_night_times = ['9:00 PM', '10:00 PM', '11:00 PM', 'midnight', '1:00 AM', '2:00 AM', '3:00 AM',
'4:00 AM']

for i in range(len(df_venues)):
    venue_id_query = df_venues['id'][i]
    url = 'https://api.foursquare.com/v2/venues/{}?
client_id={}&client_secret={}&v={}'.format(venue_id_query, CLIENT_ID, CLIENT_SECRET, VERSION)

    results_venues_details = requests.get(url).json()
    venue_id = results_venues_details['response']['venue']['id']

    #Loop through nested JSON properties to check for late-night eateries only
    if any(time for time in late_night_times):
        if 'rating' in results_venues_details['response']['venue']:
            rating = results_venues_details['response']['venue']['rating']
            results_venues_ratings.loc[len(results_venues_ratings.index)+1] = [venue_id,rating]
    else:
        continue
```

In [67]:

```python
# A preview of the results for the second API call
print("There are ",results_venues_ratings.shape[0]," late-night eateries found in this Sydney CBD
area.")
results_venues_ratings.head
```

```
There are  15  late-night eateries found in this Sydney CBD area.
```

Out[67]:

```
<bound method NDFrame.head of                                  id  rating
1   4b1a411cf964a52071e823e3     7.4
2   4b2dd2b8f964a5208cdb24e3     5.9
3   5462e750498ecce55bab57ed     6.9
4   50d78864e4b09e2e6257180e     6.7
5   5667972f498e360d3e2729a4     6.9
6   4b186acff964a52071d223e3     6.4
7   4b0cb6c1f964a520414123e3     8.4
8   4b079ba2f964a520f2fe22e3     6.7
9   4bd7cc1edc4b952178f77788     7.3
10  5a34e34f872f7d4eeb89ad43     8.4
11  57f31be1498eaf52f9d89cc6     6.6
12  5a16883f6e465009cd555aca     7.4
13  58507ff37d0f6d3a931023fd     7.8
14  4b058767f964a520af9022e3     8.3
15  4b444e76f964a520b8f325e3     5.4>
```

3.2.3 Parking Facility (Private Transport) Data Load

In [68]:

```
with urllib.request.urlopen('https://raw.githubusercontent.com/ozzie-
mughal/Coursera_Capstone/master/OffstreetparkingData.geojson') as geojson_offstreetparking:
    results_offstreetparking = json.loads(geojson_offstreetparking.read().decode())

# assign relevant part of JSON to venues
offstreetparking = results_offstreetparking['features']

# tranform venues into a dataframe
df_offstreetparking = json_normalize(offstreetparking)

#A preview of the results for the GeoJSON load
df_offstreetparking.head()
```

Out[68]:

	type	geometry.type	geometry.coordinates	properties.FRD_ID	properties.Owner	properties.Building_name_location	properties.Stre
0	Feature	Point	[151.212657, -33.86218]	FRD-001	Sir Stamford Hotel	93 Macquarie Street	
1	Feature	Point	[151.209166, -33.861878]	FRD-002	Wilson	Gold Fields House	
2	Feature	Point	[151.209767, -33.8644]	FRD-003	Wilson	No. 1 OConnell St Car Park	
3	Feature	Point	[151.211869, -33.865398]	FRD-004	Wilson	The Chifley Tower	
4	Feature	Point	[151.2107, -33.865404]	FRD-005	Secure	Sofitel Wentworth Hotel	

5 rows × 24 columns

3.2.4 Public Transport Data Load

In [69]:

```
results_publictransport = pd.read_csv('https://raw.githubusercontent.com/ozzie-
mughal/Coursera_Capstone/master/LocationFacilitiesData.csv')

#A preview of the results for the CSV load
results_publictransport.head()
```

Out[69]:

	LOCATION_NAME	TSN	X_COORD	Y_COORD	EFA_ID	PHONE	ADDRESS	FACILITIE
0	Aberdeen Station	233610	150.892052	-32.167104	Coach1192	02 6543 1018	New England Hwy, Aberdeen	2004|2017|200
1	Adamstown Station	228920	151.720081	-32.933730	Coach1159	02 4962 9295	Park Ave, Adamstown	2003|2004|2005|2006|201
2	Albion Park Station	252710	150.798500	-34.562647	Coach1381	02 4223 5416	Princes Hwy, Albion Park	2002|2003|2004|2005|2006|2015|2008|200
3	Allawah Station	222020	151.114330	-33.969584	Coach1338	02 9563 7415	Railway Pde, Allawah	2002|2003|2004|2005|2006|2015|2008|2009|2011|201
4	Arncliffe Station	220520	151.147300	-33.936456	Coach1333	02 9563 7431	Firth St, Arncliffe	2002|2003|2004|2005|2006|2015|200

3.3 Data Visualisation & Pre-Processing

The data loaded in the previous section is now pre-processed in preparation for classification model ingestion. This will involve alignment such as one-hot encoding, in order to use binary variables in place of categorical for the models, and normalising values to enhance data integrity. Other functions such as *count()* and *numpy.linalg.norm* will also be applied to various fields to conduct proximity calculations. The descriptions of the features to be extracted for a final dataset, as well as their relevance to the business problem and modeling, can be viewed in the Section 2 tables. Ultimately, the final dataset, used to train the models, will contain some venue details, such as their coordinates and ratings, along with proximity to private parking spaces and public transport locations.

3.3.1 Preparing the Venue Details

The datasets obtained from the two FourSquare API calls will be joined on the Venue ID.

In [70]:

```python
#Merge the datasets on inner join
df_venues_details = pd.merge(df_venues, results_venues_ratings, on='id', how='inner')
df_venues_details.head()
```

Out[70]:

	id	name	categories	referralId	hasPerk	location.lat	location.lng	location.label
0	4b1a411cf964a52071e823e3	Steersons Steakhouse	[{'id': '4bf58dd8d48988d1cc941735', 'name': 'S...	v-1578838812	False	-33.866674	151.201425	[{'label': 'dis 33.86667
1	4b2dd2b8f964a5208cdb24e3	Georges Mediterranean Bar & Grill	[{'id': '4bf58dd8d48988d1c0941735', 'name': 'M...	v-1578838812	False	-33.866665	151.201309	[{'label': 'dis 33.86666
2	5462e750498ecce55bab57ed	Meat District Co.	[{'id': '4bf58dd8d48988d16c941735', 'name': 'B...	v-1578838812	False	-33.866472	151.201361	[{'label': 'dis 33.86647
3	50d78864e4b09e2e6257180e	Simplicity Cafe	[{'id': '4bf58dd8d48988d1e0931735', 'name': 'C...	v-1578838812	False	-33.866847	151.201457	[{'label': 'dis 33.86684
4	5667972f498e360d3e2729a4	Beer Deluxe	[{'id': '56aa371ce4b08b9a8d57356c', 'name': 'B...	v-1578838812	False	-33.866442	151.201168	[{'label': 'dis 33.86644

Various response fields are removed as they are not relevant to the business problem (see Section 2)

In [71]:

```python
# filter columns for relevant fields
filtered_col_venues = ['id', 'name', 'categories','hasPerk','location.address','location.lat','loca
tion.lng','location.state','location.country','location.formattedAddress','delivery.id','rating']
df_venues_details = df_venues_details.loc[:, filtered_col_venues]

df_venues_details.head()
```

Out[71]:

	id	name	categories	hasPerk	location.address	location.lat	location.lng	location.
0	4b1a411cf964a52071e823e3	Steersons Steakhouse	[{'id': '4bf58dd8d48988d1cc941735', 'name': 'S...	False	King St. Wharf	-33.866674	151.201425	
1	4b2dd2b8f964a5208cdb24e3	Georges Mediterranean Bar & Grill	[{'id': '4bf58dd8d48988d1c0941735', 'name': 'M...	False	Wharf 3, The Promenade, King Street Wharf	-33.866665	151.201309	
2	5462e750498ecce55bab57ed	Meat District Co.	[{'id': '4bf58dd8d48988d16c941735', 'name': 'B...	False	R3/11 Lime Street	-33.866472	151.201361	
3	50d78864e4b09e2e6257180e	Simplicity Cafe	[{'id': '4bf58dd8d48988d1e0931735', 'name': 'C...	False	Lot 207, 19 Lime St.	-33.866847	151.201457	
4	5667972f498e360d3e2729a4	Beer Deluxe	[{'id': '56aa371ce4b08b9a8d57356c', 'name': 'B...	False	9 Lime Street	-33.866442	151.201168	

If the eatery has delivery available, this will also be considered as the restaurant having a 'perk' (along with the **hasPerk** column. To account for scenarios where Foursquare might have **hasPerk** = *False* and delivery existing, the **hasPerk** column is updated to reflect if delivery exists.

In [72]:

```
for i in range(len(df_venues_details)):
    if df_venues_details['delivery.id'].iloc[i]!="NaN" and df_venues_details['hasPerk'].iloc[i]=='T
rue':
        df_venues_details['hasPerk'].replace('False','True',inplace=True)
        print('A hasPerk column value was updated to True')
```

As the Delivery ID column is considered a restaurant perk, it can be removed following the applied logic above.

In [73]:

```
#Drop delivery.id column
df_venues_details.drop(['delivery.id'],axis=1,inplace=True)
```

Update column names to match functional names described in Section 2

In [74]:

```
df_venues_details.columns=['Venue ID','Venue Name','Venue Category','Venue Feature
Indicator','Venue Address','Venue Latitude Coordinate','Venue Longitude Coordinate','Venue State',
'Venue Country','Venue Formatted Address','Venue Rating']

df_venues_details
```

Out[74]:

	Venue ID	Venue Name	Venue Category	Venue Feature Indicator	Venue Address	Venue Latitude Coordinate	Venue Longitude Coordinate	Venue State	Ve Cou
0	4b1a411cf964a52071e823e3	Steersons Steakhouse	[{'id': '4bf58dd8d48988d1cc941735', 'name': 'S...	False	King St. Wharf	-33.866674	151.201425	NSW	Aust
1	4b2dd2b8f964a5208cdb24e3	Georges Mediterranean Bar & Grill	[{'id': '4bf58dd8d48988d1c0941735', 'name': 'M...	False	Wharf 3, The Promenade, King Street Wharf	-33.866665	151.201309	NSW	Aust
2	5462e750498ecce55bab57ed	Meat District Co.	[{'id': '4bf58dd8d48988d16c941735', 'name': 'B...	False	R3/11 Lime Street	-33.866472	151.201361	NSW	Aust
3	50d78864e4b09e2e6257180e	Simplicity Cafe	[{'id': '4bf58dd8d48988d1e0931735', 'name': 'C...	False	Lot 207, 19 Lime St.	-33.866847	151.201457	NSW	Aust
4	5667972f498e360d3e2729a4	Beer Deluxe	[{'id': '56aa371ce4b08b9a8d57356c', 'name': 'B...	False	9 Lime Street	-33.866442	151.201168	NSW	Aust
5	4b186acff964a52071d223e3	Strike Bowling Bar	[{'id': '4bf58dd8d48988d1e4931735', 'name': 'B...	False	King St. Wharf	-33.867115	151.201618	NSW	Aust
6	4b0cb6c1f964a520414123e3	Macquarie Group (1SS)	[{'id': '503287a291d4c4b30a586d65', 'name': 'F...	False	1 Shelley St.	-33.866396	151.202186	NSW	Aust
7	4b079ba2f964a520f2fe22e3	Bungalow 8	[{'id': '4bf58dd8d48988d116941735', 'name': 'B...	False	King St Wharf	-33.866191	151.201248	NSW	Aust
8	4bd7cc1edc4b952178f77788	Fitness First Platinum	[{'id': '4bf58dd8d48988d176941735', 'name': 'G...	False	1 Shelley St.	-33.866623	151.201835	NSW	Aust

	Venue ID	Venue Name	Venue Category	Venue Feature Indicator	Venue Address	Venue Latitude Coordinate	Venue Longitude Coordinate	Venue State	Ve Cou
9	5a34e34f872f7d4eeb89ad43	Smoke	[{'id': '4bf58dd8d48988d116941735', 'name': 'B...	False	NaN	-33.865606	151.201467	NSW	Aust
10	57f31be1498eaf52f9d89cc6	Lal Qila Darling Harbour	[{'id': '52e81612bcbc57f1066b79f8', 'name': 'P...	False	30 Lime Street	-33.867487	151.201907	NSW	Aust
11	5a16883f6e465009cd555aca	All Hands Brewing House	[{'id': '50327c8591d4c4b30a586d5d', 'name': 'B...	False	22 The Promenade	-33.867259	151.201431	NSW	Aust
12	58507ff37d0f6d3a931023fd	JOE & THE JUICE	[{'id': '4bf58dd8d48988d112941735', 'name': 'J...	False	400 Barangaroo Avenue	-33.865676	151.201723	NSW	Aust
13	4b058767f964a520af9022e3	The Malaya	[{'id': '4bf58dd8d48988d156941735', 'name': 'M...	False	39 Lime St.	-33.867776	151.201419	NSW	Aust
14	4b444e76f964a520b8f325e3	Ibis Sydney King Street Wharf	[{'id': '4bf58dd8d48988d1fa931735', 'name': 'H...	False	22 Shelley St	-33.866694	151.202628	NSW	Aust

3.3.2 Preparing the Parking Space Details

Various response fields are removed as they are not relevant to the business problem (see Section 2).

In [75]:

```
# filter columns for relevant fields
filtered_col_offstreetparking =
['geometry.coordinates','properties.FRD_ID','properties.Building_name_location','properties.Street_
Number_GPS','properties.Street_Name_GPS','properties.Suburb_GPS','properties.State','properties.Pos
tcode','properties.Total_number_of_bays']
df_offstreetparking = df_offstreetparking.loc[:,filtered_col_offstreetparking]

df_offstreetparking.head()
```

Out[75]:

	geometry.coordinates	properties.FRD_ID	properties.Building_name_location	properties.Street_Number_GPS	properties.Street_Name_G
0	[151.212657, -33.86218]	FRD-001	93 Macquarie Street	2	Alber
1	[151.209166, -33.861878]	FRD-002	Gold Fields House	18	Pit
2	[151.209767, -33.8644]	FRD-003	No. 1 OConnell St Car Park	3	Ben
3	[151.211869, -33.865398]	FRD-004	The Chifley Tower	27	Ben
4	[151.2107, -33.865404]	FRD-005	Sofitel Wentworth Hotel	15	Bligh

Update column names to reflect functional names described in section 2.

In [76]:

```
df_offstreetparking.columns = ['Parking Space Coordinates','Parking Space ID','Parking Space
Building Name','Parking Space Street Number','Parking Space Street Name','Parking Space
Suburb','Parking Space State','Parking Space Postal Code','Total Number of Parking Space Bays']
df_offstreetparking
```

Out[76]:

	Parking Space Coordinates	Parking Space ID	Parking Space Building Name	Parking Space Street Number	Parking Space Street Name	Parking Space Suburb	Parking Space State	Parking Space Postal Code	Total Number of Parking Space Bays

	Parking Space Coordinates	Parking Space ID	Parking Space Building Name	Parking Space Street Number	Parking Space Street Name	Parking Space Suburb	Parking Space State	Parking Space Postal Code	Total Number of Parking Space Bays
0	[151.212657, -33.86218]	FRD-001	93 Macquarie Street	2	Albert St	Sydney	NSW	2000	94.0
1	[151.209166, -33.861878]	FRD-002	Gold Fields House	18	Pitt St	Sydney	NSW	2000	114.0
2	[151.209767, -33.8644]	FRD-003	No. 1 OConnell St Car Park	3	Bent St	Sydney	NSW	2000	100.0
3	[151.211869, -33.865398]	FRD-004	The Chifley Tower	27	Bent St	Sydney	NSW	2000	362.0
4	[151.2107, -33.865404]	FRD-005	Sofitel Wentworth Hotel	15	Bligh St	Sydney	NSW	2000	173.0
...	...	...	...	...	...	...	...	...	...
73	[151.206304, -33.871104]	FRD-076	Queen Victoria Building	111	York St	Sydney	NSW	2000	634.0
74	[151.2093, -33.873586]	FRD-077	201 Elizabeth St	190	Casltereagh St	Sydney	NSW	2000	30.0
75	[151.202301, -33.856529]	FRD-078	Barangaroo Reserve Car Park	2	Towns Place	Millers Point	NSW	2000	300.0
76	[151.205553, -33.870599]	FRD-079	44 Market St	178	Clarence St	Sydney	NSW	2000	20.0
77	[151.207762, -33.863214]	FRD-080	220 George St	2	Dalley St	Sydney	NSW	2000	NaN

78 rows × 9 columns

3.3.3 Preparing the Public Transport Details

Various response fields are removed as they are not relevant to the business problem (see Section 2).

In [77]:

```
results_publictransport.drop(['EFA_ID','TSN','FACILITIES','ACCESS','PHONE'],axis=1,inplace=True)
```

Update column names to reflect functional names described in section 2.

In [78]:

```
results_publictransport.columns = ['Public Transport Location Name','Public Transport Location Lon
gitude','Public Transport Location Latitude','Public Transport Location Street Name','Public Trans
port Type(s)']

results_publictransport
```

Out[78]:

	Public Transport Location Name	Public Transport Location Longitude	Public Transport Location Latitude	Public Transport Location Street Name	Public Transport Type(s)
0	Aberdeen Station	150.892052	-32.167104	New England Hwy, Aberdeen	Train
1	Adamstown Station	151.720081	-32.933730	Park Ave, Adamstown	Train|Bus
2	Albion Park Station	150.798500	-34.562647	Princes Hwy, Albion Park	Train|Bus
3	Allawah Station	151.114330	-33.969584	Railway Pde, Allawah	Train|Bus
4	Arncliffe Station	151.147300	-33.936456	Firth St, Arncliffe	Train|Bus|Taxi rank
...	...	...	...	...	...
427	Wellington Station	148.946832	-32.554036	Swift St Wellington	Train|Coach
428	Werris Creek Station	150.646464	-31.350139	Single St, Werris Creek	Train|Taxi rank
429	Willow Tree Station	150.726123	-31.650065	Main North Rd, Willow Tree	Train
430	Wingham Station	152.367636	-31.868090	Price St, Wingham	Train
431	Yass Junction	148.915387	-34.808995	Faulder Ave, Yass	Train|Coach|Taxi rank

432 rows × 5 columns

3.3.4 Feature Selection/Extraction

The datasets are further modified to become more informative and compliant as training data for the classification models.

The **Venue Feature indicator** seems to have a *False* value for all chosen late-night eateries. There is no benefit is using this column, then, for classifying the eateries. This column will be removed.

In [79]:

```
# drop venue feature indicator column
df_venues_details.drop(['Venue Feature Indicator'],axis=1,inplace=True)
```

The **Total Number of Bays** values must be normalised to ensure zero variance in the mean and distribution. These normalised values will be used to determine weighting to a parking space's proximity to an eatery.

In [80]:

```
# scale total number of bays to a range between 0 and 1

df_parking_bay_values = df_offstreetparking['Total Number of Parking Space Bays'].values
amin, amax = min(df_parking_bay_values), max(df_parking_bay_values)
for i, val in enumerate(df_parking_bay_values):
    df_parking_bay_values[i] = (val-amin) / (amax-amin)
df_offstreetparking['Total Number of Parking Space Bays'] = df_parking_bay_values
```

In [81]:

```
df_offstreetparking
```

Out[81]:

	Parking Space Coordinates	Parking Space ID	Parking Space Building Name	Parking Space Street Number	Parking Space Street Name	Parking Space Suburb	Parking Space State	Parking Space Postal Code	Total Number of Parking Space Bays
0	[151.212657, -33.86218]	FRD-001	93 Macquarie Street	2	Albert St	Sydney	NSW	2000	0.070237
1	[151.209166, -33.861878]	FRD-002	Gold Fields House	18	Pitt St	Sydney	NSW	2000	0.087796
2	[151.209767, -33.8644]	FRD-003	No. 1 OConnell St Car Park	3	Bent St	Sydney	NSW	2000	0.075505
3	[151.211869, -33.865398]	FRD-004	The Chifley Tower	27	Bent St	Sydney	NSW	2000	0.305531
4	[151.2107, -33.865404]	FRD-005	Sofitel Wentworth Hotel	15	Bligh St	Sydney	NSW	2000	0.139596
...	...	...	...	...	...	...	...	...	...
73	[151.206304, -33.871104]	FRD-076	Queen Victoria Building	111	York St	Sydney	NSW	2000	0.544337
74	[151.2093, -33.873586]	FRD-077	201 Elizabeth St	190	Casltereagh St	Sydney	NSW	2000	0.014047
75	[151.202301, -33.856529]	FRD-078	Barangaroo Reserve Car Park	2	Towns Place	Millers Point	NSW	2000	0.251097
76	[151.205553, -33.870599]	FRD-079	44 Market St	178	Clarence St	Sydney	NSW	2000	0.005268
77	[151.207762, -33.863214]	FRD-080	220 George St	2	Dalley St	Sydney	NSW	2000	NaN

78 rows × 9 columns

To add final touches to the value of the venue features, the private/public transport location data will be analysed to create an *accessibility scale* of 1-10 for each venue below, with 1 being very inconvenient for accessibility and 10 being very convenient. It is assumed that a 100 metre walk to the venue is very convenient.

In [82]:

```
# create dataframe for accessibility scale
data = [['Very Accessible',10,'up to 50 metres'],['...',9,'Between 50 and 150 metres'],
['...',8,'Between 150 and 300 metres'],['...',7,'Between 300 and 500 metres'],['...',6,'Between 50
0 and 750 metres'],['...',5,'Between 750 and 1050 metres'],['...',4,'Between 1050 and 1400 metres'
],['...',3,'Between 1400 and 1800 metres'],['...',2,'Between 1800 and 2250 metres'],['Very
Inaccessible',1,'More than 2250 metres']]
acc_scale = pd.DataFrame(data,columns = ['Accessibility','Factor Value','Distance from Venue (m)']
)

acc_scale
```

Out[82]:

	Accessibility	Factor Value	Distance from Venue (m)
0	Very Accessible	10	up to 50 metres
1	...	9	Between 50 and 150 metres
2	...	8	Between 150 and 300 metres
3	...	7	Between 300 and 500 metres
4	...	6	Between 500 and 750 metres
5	...	5	Between 750 and 1050 metres
6	...	4	Between 1050 and 1400 metres
7	...	3	Between 1400 and 1800 metres
8	...	2	Between 1800 and 2250 metres
9	Very Inaccessible	1	More than 2250 metres

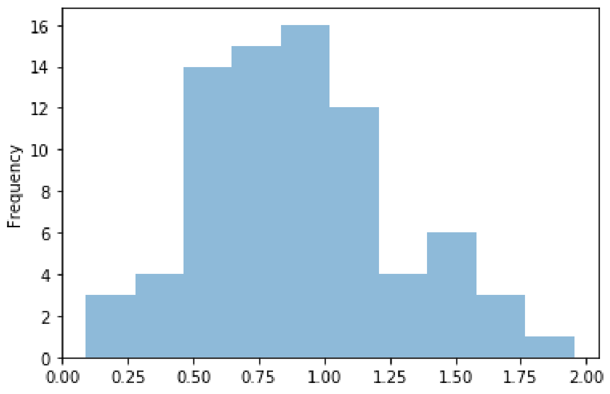
The euclidean distance from each transport location centre to a venue is calculated and placed into 10 bins for visualisation. The average euclidean distance is then calculated and scaled to the range of 1-10.

In [83]:

```
# display histogram of 10 bins for the first venue
transp_dist = []

ven_lat = df_venues_details['Venue Latitude Coordinate'].iloc[0]
ven_long = df_venues_details['Venue Longitude Coordinate'].iloc[0]

for i in range(len(df_offstreetparking)):
    transp_lat = df_offstreetparking['Parking Space Coordinates'].iloc[i][1]
    transp_long = df_offstreetparking['Parking Space Coordinates'].iloc[i][0]
    transp_dist.append(distance.distance((ven_lat,ven_long),(transp_lat,transp_long)).km)

x = pd.Series(transp_dist)
x.plot.hist(bins=10, alpha=0.5)
```

Out[83]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f03337cf550>
```



We can see for Steerson's Steakhouse (located on 17 Lime street, Sydney), most private transport parking spaces lay between 0.875-1.0 kilometres from the venue. This is about a 10-12min walk, which is still a fair bit for Sydney's heaviest pedestrian-populated area. However, the histogram is skewed fairly to the right, so most of the parking spaces will prove less than a 12min walk.

area. However, the histogram is skewed fairly to the right, so most of the parking spaces will prove less than a 12min walk.

In [84]:

```
print('The average distance from a private parking space to the Steerson\'s Steakhouse venue is ',
x.mean(),' kilometres, with a standard deviation of ',x.std())
```

```
The average distance from a private parking space to the Steerson's Steakhouse venue is
0.8897846505302727  kilometres, with a standard deviation of  0.37189718105109537
```

As discussed earlier, the total number of bays aids to the convenience of a parking space, and compensates for it's distance from the venue to some extent. The normalised number of bays values are used as inverse multipliers to the distances of each parking space, and the scaled distances categorised into the *accessibility scale*.

In [85]:

```
# categorise distances on accessibility scale

transp_acc_scaled = []

for i in range(len(df_offstreetparking)):
    num_of_bays = df_offstreetparking['Total Number of Parking Space Bays'].iloc[i]
    x[i] = (x[i]*1000)*(1/(num_of_bays*10)) #units are in metres

    if x[i]<50:
        scale_val = 10
    elif (50<x[i] and x[i]<150):
        scale_val = 9
    elif (150<x[i] and x[i]<300):
        scale_val = 8
    elif (300<x[i] and x[i]<500):
        scale_val = 7
    elif (500<x[i] and x[i]<750):
        scale_val = 6
    elif (750<x[i] and x[i]<1050):
        scale_val = 5
    elif (1050<x[i] and x[i]<1400):
        scale_val = 4
    elif (1400<x[i] and x[i]<1800):
        scale_val = 3
    elif (1800<x[i] and x[i]<2250):
        scale_val = 2
    elif x[i]>2250:
        scale_val = 1

    transp_acc_scaled.append(scale_val)
```

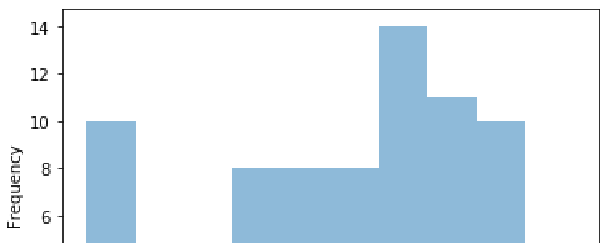
```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/ipykernel_launcher.py:7:
RuntimeWarning: divide by zero encountered in double_scalars
  import sys
```

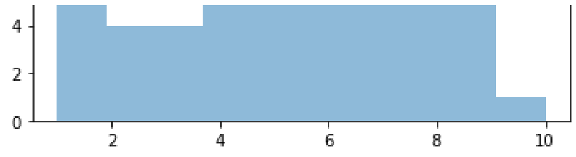
In [86]:

```
# display histogram

x_transp_acc_scaled = pd.Series(transp_acc_scaled)

x_transp_acc_scaled.plot.hist(bins=10, alpha=0.5)
```

Out[86]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0333745128>
```





We can see most private parking spaces, taking into account their total number of bays available, are fairly convenient (>6) to use to walk to a late-night eatery. It should be noted that a noticeable amount of spaces (10) are more than 2.25km away, compared to the peak of 14 spaces between 300m and 500m away. This spike could be due to the upper bound of 2.25km being poorly assumed.

In [87]:

```
print('The average accessibility scale value for the Steerson\'s Steakhouse venue is
',x_transp_acc_scaled.mean(),' kilometres, with a standard deviation of ',x_transp_acc_scaled.std(
))
```

```
The average accessibility scale value for the Steerson's Steakhouse venue is  5.589743589743589  ki
lometres, with a standard deviation of  2.65034130336246
```

The accessibility scale values are now calculated below for the rest of the venues, as well as the calculations for public transport locations.

NOTE: As the public transport dataset contains *all* public transport across the state of NSW, only the locations within the proximity of Sydney CBD (~837m, as derived in section 3.2.2) will be processed for accessibility to a late-night eatery in CBD.

In [88]:

```
avg_transp_acc_scaled = [] #this will hold the average accessibility scale value for each venue

for v in range(len(df_venues_details)):

    transp_dist = [] # this will hold the euclidean distances from each private/public transport l
ocation from a venue
    transp_acc_scaled = [] # this will hold the accessibility scale values for each private/public
transport location for a venue

    ven_lat = df_venues_details['Venue Latitude Coordinate'].iloc[v]
    ven_long = df_venues_details['Venue Longitude Coordinate'].iloc[v]

    for j in range(len(df_offstreetparking)):
        transp_lat = df_offstreetparking['Parking Space Coordinates'].iloc[j][1]
        transp_long = df_offstreetparking['Parking Space Coordinates'].iloc[j][0]
        transp_dist.append(distance.distance((ven_lat,ven_long),(transp_lat,transp_long)).km)

    for k in range(len(results_publictransport)):
        transp_lat = results_publictransport['Public Transport Location Latitude'].iloc[k]
        transp_long = results_publictransport['Public Transport Location Longitude'].iloc[k]
        transp_dist.append(distance.distance((ven_lat,ven_long),(transp_lat,transp_long)).km)


    x = pd.Series(transp_dist)

    for i in range(len(x)):
        x[i] = (x[i])*1000 # convert to metres units
        if i<(len(df_offstreetparking)):
            num_of_bays = df_offstreetparking['Total Number of Parking Space Bays'].iloc[i]
            x[i] = (x[i])*(1/(num_of_bays*10))
        elif x[i]>837:
            continue # if public transport location is outside of Sydney CBD, disregard and move tc
next iteration
        if x[i]<50:
            scale_val = 10
        elif (50<x[i] and x[i]<150):
            scale_val = 9
        elif (150<x[i] and x[i]<300):
            scale_val = 8
        elif (300<x[i] and x[i]<500):
            scale_val = 7
        elif (500<x[i] and x[i]<750):
            scale_val = 6
        elif (750<x[i] and x[i]<1050):
            scale_val = 5
        elif (1050<x[i] and x[i]<1400):
            scale_val = 4
        elif (1400<x[i] and x[i]<1800):
            scale_val = 3
        elif (1800<x[i] and x[i]<2250):
            scale_val = 2
        elif x[i]>2250:
            scale_val = 1

        transp_acc_scaled.append(scale_val)

    x_transp_acc_scaled = pd.Series(transp_acc_scaled)

    avg_transp_acc_scaled.append(x_transp_acc_scaled.mean())

avg_transp_acc_scaled
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/ipykernel_launcher.py:28:
RuntimeWarning: divide by zero encountered in double_scalars
```

Out[88]:

```
[5.690476190476191,
 5.690476190476191,
 5.666666666666667,
 5.690476190476191,
 5.642857142857143,
 5.726190476190476,
 5.845238095238095,
 5.630952380952381,
 5.785714285714286,
 5.626506024096385,
 5.752941176470588,
 5.7023809523809526,
 5.698795180722891,
 5.647058823529412,
 5.916666666666667]
```

In order to effectively assess competition between the late-night eateries, the average accessibility values for each eatery are normalised.

In [89]:

```
# normalise absolute accessibility value of each venue between 0 and 1

amin, amax = min(avg_transp_acc_scaled), max(avg_transp_acc_scaled)
for i, val in enumerate(avg_transp_acc_scaled):
    avg_transp_acc_scaled[i] = (val-amin) / (amax-amin)
df_venues_details['Relative Accessibility Score (0-1)'] = avg_transp_acc_scaled
```

The correlation between the accessibility score and venue rating is calculated, using Pearson's Correlation Coefficient.

In [90]:

```
rating_acc_corr = pearsonr(df_venues_details['Relative Accessibility Score (0-
1)'].values,df_venues_details['Venue Rating'].values)
print('The Pearson Correlation Coefficient value between the relative accessibility score, and the
venue rating, is ',rating_acc_corr)
```

```
The Pearson Correlation Coefficient value between the relative accessibility score, and the venue
rating, is  (-0.30404470044157617, 0.27057175060019517)
```

The **Venue Rating** is the response variable, hence test records will be classified into a categorical form of this. The ratings, of continuous type, are grouped into the following categorical bins:

Rating Range	Rating Category
0-2.5	Very Poor
2.5-5	Poor

5-6.5	Okay
6.5-8	Great
8-10	Excellent

In [91]:

```
# categorise venue ratings into categorical bins

for r in range(len(df_venues_details['Venue Rating'])):
    rating_val = df_venues_details['Venue Rating'].iloc[r]
    if (0<=rating_val) and (rating_val<=2.5):
        df_venues_details['Venue Rating'].iloc[r] = 'Very Poor'
    elif (2.5<=rating_val) and (rating_val<=5):
        df_venues_details['Venue Rating'].iloc[r] = 'Poor'
    elif (5<=rating_val) and (rating_val<=6.5):
        df_venues_details['Venue Rating'].iloc[r] = 'Okay'
    elif (6.5<=rating_val) and (rating_val<=8):
        df_venues_details['Venue Rating'].iloc[r] = 'Great'
    elif (8<=rating_val) and (rating_val<=10):
        df_venues_details['Venue Rating'].iloc[r] = 'Excellent'
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/pandas/core/indexing.py:205:
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/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_with_indexer(indexer, value)
```

In [92]:

```
# rearrange columns so dependent variable is the last column

df_venues_details = df_venues_details[['Venue ID',
 'Venue Name',
 'Venue Category',
 'Venue Address',
 'Venue Latitude Coordinate',
 'Venue Longitude Coordinate',
 'Venue State',
 'Venue Country',
 'Venue Formatted Address',
 'Relative Accessibility Score (0-1)',
  'Venue Rating']]

df_venues_details
```

Out[92]:

	Venue ID	Venue Name	Venue Category	Venue Address	Venue Latitude Coordinate	Venue Longitude Coordinate	Venue State	Venue Country	For A
0	4b1a411cf964a52071e823e3	Steersons Steakhouse	[{'id': '4bf58dd8d48988d1cc941735', 'name': 'S...	King St. Wharf	-33.866674	151.201425	NSW	Australia	[W Li NSW
1	4b2dd2b8f964a5208cdb24e3	Georges Mediterranean Bar & Grill	[{'id': '4bf58dd8d48988d1c0941735', 'name': 'M...	Wharf 3, The Promenade, King Street Wharf	-33.866665	151.201309	NSW	Australia	[V Pron Kin Wha
2	5462e750498ecce55bab57ed	Meat District Co.	[{'id': '4bf58dd8d48988d16c941735', 'name': 'B...	R3/11 Lime Street	-33.866472	151.201361	NSW	Australia	[R3/ Sydı Cent
3	50d78864e4b09e2e6257180e	Simplicity Cafe	[{'id': '4bf58dd8d48988d1e0931735', 'name': 'C...	Lot 207, 19 Lime St.	-33.866847	151.201457	NSW	Australia	[Lot L NSV A
			[{'id':						

	Venue ID	Venue Name	Venue Category	Venue Street Address	Venue Latitude Coordinate	Venue Longitude Coordinate	Venue State	Venue Country	For A
4	5667972f498e360d3e2729a4	Beer Deluxe	[{'id': '56aa371ce4b08b9a8d57356c', 'name': 'B...	9 Lime Street	-33.866442	151.201166	NSW	Australia	NSW
5	4b186acff964a52071d223e3	Strike Bowling Bar	[{'id': '4bf58dd8d48988d1e4931735', 'name': 'B...	King St. Wharf	-33.867115	151.201618	NSW	Australia	[W Prom
6	4b0cb6c1f964a520414123e3	Macquarie Group (1SS)	[{'id': '503287a291d4c4b30a586d65', 'name': 'F...	1 Shelley St.	-33.866396	151.202186	NSW	Australia	[1 St., NS\ A
7	4b079ba2f964a520f2fe22e3	Bungalow 8	[{'id': '4bf58dd8d48988d116941735', 'name': 'B...	King St Wharf	-33.866191	151.201248	NSW	Australia	[V Li NS\
8	4bd7cc1edc4b952178f77788	Fitness First Platinum	[{'id': '4bf58dd8d48988d176941735', 'name': 'G...	1 Shelley St.	-33.866623	151.201835	NSW	Australia	[1 St. St.
9	5a34e34f872f7d4eeb89ad43	Smoke	[{'id': '4bf58dd8d48988d116941735', 'name': 'B...	NaN	-33.865606	151.201467	NSW	Australia	[Bara NS\ A
10	57f31be1498eaf52f9d89cc6	Lal Qila Darling Harbour	[{'id': '52e81612bcbc57f1066b79f8', 'name': 'P...	30 Lime Street	-33.867487	151.201907	NSW	Australia	[: NS\ A
11	5a16883f6e465009cd555aca	All Hands Brewing House	[{'id': '50327c8591d4c4b30a586d5d', 'name': 'B...	22 The Promenade	-33.867259	151.201431	NSW	Australia	Pror (Kin Sy
12	58507ff37d0f6d3a931023fd	JOE & THE JUICE	[{'id': '4bf58dd8d48988d112941735', 'name': 'J...	400 Barangaroo Avenue	-33.865676	151.201723	NSW	Australia	Bara Aven A Sydr
13	4b058767f964a520af9022e3	The Malaya	[{'id': '4bf58dd8d48988d156941735', 'name': 'M...	39 Lime St.	-33.867776	151.201419	NSW	Australia	[39 L (NSW
14	4b444e76f964a520b8f325e3	Ibis Sydney King Street Wharf	[{'id': '4bf58dd8d48988d1fa931735', 'name': 'H...	22 Shelley St	-33.866694	151.202628	NSW	Australia	[22 St, NS\ A

The features have now been extracted and selected, priming the data for classification model ingestion. The following numerical variables will be used for classification:

- Venue Latitude Coordinate
- Venue Longitude Coordinate
- Relative Accessibility Score (0-1)
- Venue Rating

3.4 Classification

Each model has training data selected from the final dataset, trained, and evaluated to determine the parameter values for optimum fitting. The models are then compared using accuracy scores, and the chosen model with highest accuracy identified. This model is then used for predicting the ideal locations for a late-night eatery in order to acquire a certain level of rating (ie. the response variable).

Some assumptions:

- The test size to be used is 20% (0.2), with a random state of 4.
- The Decision Tree classifier uses *entropy* as a criterion to determine information gain, and further, the best variable to split the tree by next
- The Support Vector Machine classifier will be tested across 4 kernel types: Linear, Polynomial, Radial-Basis, and Sigmoid.

- The Logistic Regression classifier will use the *liblinear* solver to allow for L1 and L2 regularisation and allow dense and sparse input

In [93]:

```
# Assign dependent and independent variable sets
X = df_venues_details[['Venue Latitude Coordinate','Venue Longitude Coordinate','Relative
Accessibility Score (0-1)']]
y = df_venues_details['Venue Rating']
```

3.4.1 K-Nearest Neighbor (KNN)

In [94]:

```
#TRAINING DATA SELECTION

#Split data in test and train data

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=4)
```

In [95]:

```
#MODEL TRAINING AND EVALUATION

#Fit K-neighbors classifier model and predict for a range of K values from 1 to 10
#Initialise maximum K, and mean and SD matrices
Ks = 7
mean_ac = np.zeros((Ks-1))
SD_ac = np.zeros((Ks-1))

#Loop through K while predicting y_hat, and calculate mean and standard deviation of accuracy scor
e to determine best K
for k in range(1,Ks):

    kneigh = KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)

    y_hat = kneigh.predict(X_test)
    mean_ac[k-1] = metrics.accuracy_score(y_hat,y_test)
    SD_ac[k-1] = np.std(y_hat==y_test)/np.sqrt(y_hat.shape[0])

K_best = mean_ac.argmax()+1
```

In [96]:

```
#MODEL OPTIMUM FITTING
kneigh_best = KNeighborsClassifier(n_neighbors=K_best).fit(X_train,y_train)

y_hat_kneigh  = kneigh.predict(X_test)
```

3.4.2 Decision Tree

In [97]:

```
#TRAINING DATA SELECTION

#Split data in test and train data

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=4)
```

In [98]:

```
#MODEL TRAINING & EVALUATION

#Fit decision tree classifier model and predict for a range of K values from 1 to 10
#Initialise maximum K, and mean and SD matrices
Ks = 30
mean_ac = np.zeros((Ks-1))
```

```
SD_ac = np.zeros((Ks-1))

for k in range(1,Ks):

    tree = DecisionTreeClassifier(criterion='entropy',max_depth=k).fit(X_train,y_train)
    y_hat = tree.predict(X_test)
    mean_ac[k-1] = metrics.accuracy_score(y_hat,y_test)
    SD_ac[k-1] = np.std(y_hat==y_test)/np.sqrt(y_hat.shape[0])

Ks_best = mean_ac.argmax()+1
```

In [99]:

```
#MODEL OPTIMUM FITTING

tree = DecisionTreeClassifier(criterion='entropy',max_depth=Ks_best).fit(X_train,y_train)
y_hat_tree  = tree.predict(X_test)
```

3.4.3 Support Vector Machine (SVM)

In [100]:

```
#TRAINING DATA SELECTION
#Split data in test and train datasets
#Split data in test and train data

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=4)
```

In [101]:

```
#MODEL FITTING AND EVALUATION

#Loop through each kernel type and calculate accuracy scores to evaluate the best kernel to use
#Initialise kernel type array and mean
kernel_types = ['linear','poly','rbf','sigmoid']
accuracy_scores = np.zeros((len(kernel_types)))

for n in kernel_types:

    clf = svm.SVC(probability=True,kernel=n)
    clf.fit(X_train, y_train)
    y_hat = clf.predict(X_test)
    accuracy_scores[kernel_types.index(n)] = metrics.accuracy_score(y_hat,y_test)

kernel_type_best = kernel_types[accuracy_scores.argmax()]
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to
account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warn
ing.
  "avoid this warning.", FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to
account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warn
ing.
  "avoid this warning.", FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/sklearn/svm/base.py:196:
FutureWarning: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to
account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warn
ing.
  "avoid this warning.", FutureWarning)
```

In [102]:

```
#MODEL OPTIMUM FITTING

clf = svm.SVC(probability=True,kernel=kernel_type_best).fit(X_train,y_train)
y_hat_clf  = clf.predict(X_test)
```

3.4.4 Logistic Regression

In [103]:

```
#TRAINING DATA SELECTION
#Split data in test and train datasets
#Split data in test and train data

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=4)
```

In [104]:

```
#MODEL FITTING & EVALUATION

#Loop through a range of C, the Regularization parameter, values to determine the optimal value fo
r fitting
#Initiliase range of C values
Cs_inverse = 100
Cs_inverse_range = range(10,Cs_inverse,10)
mean_ac = np.zeros(10)
i = 0

for c_inverse in Cs_inverse_range:

    LR = LogisticRegression(C=(1/c_inverse),solver='liblinear').fit(X_train,y_train)
    y_hat = LR.predict(X_test)
    mean_ac[i] = metrics.accuracy_score(y_hat,y_test)
    i = i + 1

C_inverse_best = Cs_inverse_range[mean_ac.argmax()]
C_best = 1/C_inverse_best
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/sklearn/linear_model/logistic.py:460: FutureWarning: Default multi_class will be changed
to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/sklearn/linear_model/logistic.py:460: FutureWarning: Default multi_class will be changed
to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/sklearn/linear_model/logistic.py:460: FutureWarning: Default multi_class will be changed
to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/sklearn/linear_model/logistic.py:460: FutureWarning: Default multi_class will be changed
to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/sklearn/linear_model/logistic.py:460: FutureWarning: Default multi_class will be changed
to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/sklearn/linear_model/logistic.py:460: FutureWarning: Default multi_class will be changed
to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/sklearn/linear_model/logistic.py:460: FutureWarning: Default multi_class will be changed
to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/sklearn/linear_model/logistic.py:460: FutureWarning: Default multi_class will be changed
to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/sklearn/linear_model/logistic.py:460: FutureWarning: Default multi_class will be changed
to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

In [105]:

```
#MODEL OPTIMUM FITTING

LR = LogisticRegression(C=C_best,solver='liblinear').fit(X_train,y_train)
y_hat_LR  = LR.predict(X_test)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/sklearn/linear_model/logistic.py:460: FutureWarning: Default multi_class will be changed
to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

3.4.5 Model Comparison & Evaluation

In [106]:

```
y_hat_ = [y_hat_kneigh,y_hat_tree,y_hat_clf,y_hat_LR]

#Predict probability values
y_hat_prob_kneigh = kneigh.predict_proba(X_test)
y_hat_prob_tree = tree.predict_proba(X_test)
y_hat_prob_clf = clf.predict_proba(X_test)
y_hat_prob_LR = LR.predict_proba(X_test)

y_hat_prob = [y_hat_prob_kneigh,y_hat_prob_tree,y_hat_prob_clf,y_hat_prob_LR]
```

In [107]:

```
#Calculate Jaccard Scores
j_y_hat_kneigh = jaccard_similarity_score(y_hat_kneigh,y_test)
j_y_hat_tree = jaccard_similarity_score(y_hat_tree,y_test)
j_y_hat_clf = jaccard_similarity_score(y_hat_clf,y_test)
j_y_hat_LR = jaccard_similarity_score(y_hat_LR,y_test)

j_y_hat = [j_y_hat_kneigh,j_y_hat_tree,j_y_hat_clf,j_y_hat_LR]
```

In [108]:

```
#Calculate F1 Scores
f_y_hat_kneigh = f1_score(y_hat_kneigh,y_test,average='weighted')
f_y_hat_tree = f1_score(y_hat_tree,y_test,average='weighted')
f_y_hat_clf = f1_score(y_hat_clf,y_test,average='weighted')
f_y_hat_LR = f1_score(y_hat_LR,y_test,average='weighted')

f_y_hat = [f_y_hat_kneigh,f_y_hat_tree,f_y_hat_clf,f_y_hat_LR]
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/sklearn/metrics/classification.py:1145: UndefinedMetricWarning: F-score is ill-defined an
d being set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
```

In [109]:

```
#Calculate Log Loss

l_y_hat_kneigh = log_loss(y_test,y_hat_prob_kneigh,labels=['Okay','Great','Excellent'])
l_y_hat_tree = log_loss(y_test,y_hat_prob_tree,labels=['Okay','Great','Excellent'])
l_y_hat_clf = log_loss(y_test,y_hat_prob_clf,labels=['Okay','Great','Excellent'])
l_y_hat_LR = log_loss(y_test,y_hat_prob_LR,labels=['Okay','Great','Excellent'])

l_y_hat = [l_y_hat_kneigh,l_y_hat_tree,l_y_hat_clf,l_y_hat_LR]
```

In [110]:

```
#Create accuracy results dataframe
acc_results = {'Algorithm': ['KNN','Decision Tree','SVM','Logistic Regression'],'Jaccard': j_y_hat
,'F1-score': f_y_hat,'Log Loss': l_y_hat}
acc_results_df = pd.DataFrame(acc_results)

acc_results_df
```

Out[110]:

	Algorithm	Jaccard	F1-score	Log Loss
0	KNN	0.8	0.888889	7.462273
1	Decision Tree	0.6	0.600000	7.463247
2	SVM	0.8	0.888889	0.869513
3	Logistic Regression	0.8	0.888889	0.879728

4. Results

The prediction results (ie. predicted venue ratings), using the test subset of data from *df_venue_details*, are as follows for each classifier model:

In [111]:

```
data = df_venues_details[['Venue Name','Venue Rating']].iloc[y_test.index.values].values
test_results = pd.DataFrame(data,columns=['Venue Name','Actual Venue Rating'])
pred_results = pd.DataFrame({'KNN Predicted Rating':y_hat_[0],'Decision Tree Predicted Rating':y_h
at_[1],'SVM Predicted Rating':y_hat_[2],'Logistic Regression Predicted Rating':y_hat_[3]})

results = pd.merge(test_results,pred_results,on=test_results.index.values)
results.drop('key_0',axis=1,inplace=True)

results
```

Out[111]:

	Venue Name	Actual Venue Rating	KNN Predicted Rating	Decision Tree Predicted Rating	SVM Predicted Rating	Logistic Regression Predicted Rating
0	JOE & THE JUICE	Great	Great	Great	Great	Great
1	Steersons Steakhouse	Great	Great	Great	Great	Great
2	Macquarie Group (1SS)	Excellent	Great	Great	Great	Great
3	Simplicity Cafe	Great	Great	Great	Great	Great
4	Beer Deluxe	Great	Great	Excellent	Great	Great

The Jaccard Similarity Index score, F1 score, and Logarithmic Loss score, for each classifier model, is as follows:

In [112]:

```
acc_results_df
```

Out[112]:

	Algorithm	Jaccard	F1-score	Log Loss
0	KNN	0.8	0.888889	7.462273
1	Decision Tree	0.6	0.600000	7.463247
2	SVM	0.8	0.888889	0.869513
3	Logistic Regression	0.8	0.888889	0.879728

The classifier model with the highest accuracy, therefore, is the K-Nearest Neighbors model. The addresses of tested venue locations with a predicted rating of either *Great* or *Excellent*, are as follows:

In [113]:

```
df_venues_details['Venue Address'].iloc[y_test.index.values]
```

Out[113]:

```
12    400 Barangaroo Avenue
0            King St. Wharf
6             1 Shelley St.
3      Lot 207, 19 Lime St.
4             9 Lime Street
Name: Venue Address, dtype: object
```

5. Discussion

The data analysis, and machine learning modelling, was successful in determining the prime locations for a new late-night eatery to set up in Sydney CBD, and have a high probability of achieving a rating of higher than 6.5/10 on FourSquare. Indeed, it was assumed that a newly set-up late-night eatery on a particular street would enjoy the same level of rating given to an already existing restaurant on that street. This is because it was hypothesised that private parking or public transport accessibility is the primary factor to determining the success of a restaurant. Therefore, if a late-night eatery is to be set up in Sydney CBD, the following streets would be most ideal to set up shop:

- Lime street, Sydney
- King Street Wharf, Sydney
- The Promenade, Sydney

These streets offer relative accessibility scores ranging from 40-75%, slightly implying the accessibility of the street may contribute heavily to the eatery success. However, interestingly the Pearson's correlation coefficient between a venue's rating and it's transport accessibility, is -0.304. If anything, there is a negative correlation between the 2 variables. Other factors may have been overlooked in this study that have a greater impact to the eatery success. They may include proximity to popular commercial buildings, such as Darling Park, and other points of interest such as malls. Another influence may be seasonal trends. Wintertime tends to favour warm-food late-night eateries, rather than ice-cream cafes. This would suppress quite a few eateries during various months, and lower their popularity and number of ratings. A future study may explore scoring each eatery based on these features instead. Granted, certain features such as seasonal trends will increase complexity of analysis, through requiring exploration of time-sensitive data, rather than a 'snapshot' or venue details and ratings as done for this study.

5.1 Data Exploration

During data pre-processing, it became obvious that very few venue records are available for study given our constraints. A total of 15 late-night eateries in Sydney CBD were identified. This is a low number of samples to be used for training classification models, introducing issues of validity to the accuracy scores and rating predictions. There may be several reasons for the low availability of data samples. Some include:

- The Foursquare database is used most heavily in the US, and hence will naturally offer more local venue details in that country. Many restaurants in Australia may simply not be captured in the database yet, either because owners/guests don't see the value of the tool yet, or the software itself has not adapted to Australia's foodie market
- The radius limit specified to the Foursquare API was too constricting. This limit was determined from the geographical definition of Sydney CBD as an area, however may be too small to illustrate the size of the eatery market in the city of Sydney.
- The conditions defining an eatery as 'late-night' may be too constricting. If an eatery stays open past 9PM, it was considered to be late-night. Many restaurants may not have their opening times updated to reflect seasonal changes (some restaurants stay open later during Christmas holidays). Moreover, many people may rather consider anything open after 8PM to be considered late-night. After all, late-night shopping on Thursdays in Sydney is defined as any shop open *up to 9PM*.
- The Standard Access to Foursquare API limited the number of results to 50.

The value of the Foursquare response data was substantial, offering many details about each venue beyond ratings and opening times, such as types of features and item price range. Many details could have been used as independent variables to understand the success of a restaurant. However, for the sake of simplicity, the hypothesis was focused upon the accessibility to private and public transport. Moreover, the types of features available (eg. reservations) was simplified to a boolean variable, and coupled with a boolean value on the availability of wi-fi.

The private parking data was also very useful in understanding not simply where each parking space is, but how available is it (ie. Total number of bays). This factor was then used to weigh each parking space and determine a final accessibility score. Similarly, the public transport data offered various modes of transport available in each location. For example, Circular Quay station provides bus, train, and ferry facilities. These modes could be one-hot encoded and further explored and used as weightings across different areas of Sydney (eg. buses are more popular around a particular street than trains). Alternatively, a tally of modes available at each location could simply be applied to each location to add weighting to it. For the sake of simplicity in this study, the influence of these modes was disregarded.

5.2 Model Exploration

4 classification models were trained for this study, and compared using various accuracy scores. It must be noted the low number of sampled available (15) severely affected the accuracy of all models. This would decrease their robustness and increase their sensitivity to new data, diminishing the validity of predicted ratings of a particular location that an eatery exists at. Given a train-test split of 0.7:0.3, the models only has 10 venues available to train on and 5 to be tested on. Furthermore, nearly all venues provided in the dataset had a numerical rating higher than 5, proving it extremely difficult to generate categorical bins that were spaced equally yet could illustrate patterns. Indeed, all 5 venues had a predicted rating of *Great*, and only the Decision Tree model predicted a rating of *Excellent* for an eatery that actually had a rating of *Great*.

In terms of accuracy, the Jaccard Simplicity Index, measuring similarity between the predicted ratings and actual ratings, all models had a score of either 0.6 or 0.8, with KNN and SVM scoring 0.8. The F1 score, conveying the balance between the precision and recall of predicted values, was also either 0.6 or 0.89; very suspiciously high scores given the low number of samples. Again, the KNN and SVM had the highest scores of 0.89 each. The Logarithmic loss, depicting the probability of a model predicting the correct value, were either 0.87 or a relatively high value of 7.46. The KNN had a value of 7.46, whilst SVM had a value of 0.87.

Given the SVM model had the highest Jaccard and F1 scores, and lowest logarithmic loss, this is evaluated to relatively be the best fitted classifier model. This model predicted having an eatery on the 3 streets, listed above in Section 5, would mean this eatery will most likely have a rating of greater than 6.5.

Many assumptions were taken into account during model generation, such as initialising K values for the KNN model, solver types used for logistic regression, and kernel types used for the SVM model. These selections would affect the predictions output, and further the model's accuracy. Regardless, the limited number of samples available would have had a much more negative impact on the accuracy of prediction.

Another approach to selecting models for this study could center around altering the business problem to finding similarities between successful restaurants, rather than classifying which have a higher rating. As discussed in Section 5.1, a venue rating retrieved from Foursquare's database may not fully paint the picture of a successful late-night eatery. Changing the business problem to find similarities could shift the type of models used to clustering instead of classifying. Each cluster group could then be further studied to understand what groups particular eateries together. An average of ratings in each group could also be calculated to identify which cluster is the 'successful group'. The answer to the business problem could then come from analysing the locations of restaurants in this group.

6. Conclusion

Following the implementation of the Sydney Lockout Laws, the night-time economy in Australia has started to shift to popularising late-night eateries. Understanding factors that impact this economy, and more particularly factors that impact the success of a late-night eatery have become vital. This study determined, using classification modeling, that 3 streets exist in Sydney CBD that would grant a higher eatery ratings. Indeed, the Support Vector Machine (SVM) model determined these streets would provide ratings higher than 6.5/10, namely:

- Lime street, Sydney
- King Street Wharf, Sydney
- The Promenade, Sydney

It must be noted a surprisingly low number of samples (15) were available for this study on venues that are both in Sydney CBD and open late-night. This further impacted the accuracy and validity of classifier models trained, and hence the streets selected as prime locations to set up a successful late-night eatery. Future studies could venture to explore other eatery databases beyond Foursquare, as well as a greater geographical area and wider range of opening times.