FIT5196-S2-2020 assessment 3

Student Name: Md. Saadman Hossain

Student ID: 31043313

Importing the required libraries

In [ ]:

```
import pandas as pd
import pandas_read_xml as pdx
from tabula import read_pdf
import re
from math import sin, cos, sqrt, atan2, radians
import datetime as dt
from sklearn import preprocessing
import xml.etree.ElementTree as et
from shapely.geometry import Point, Polygon
from sklearn.model_selection import train_test_split
```

In [ ]:

```
import numpy
import matplotlib
import shapefile
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from matplotlib.patches import Polygon
from matplotlib.collections import PatchCollection
%matplotlib inline
```

Task 1: Data Integration (60%)

reading hospitals file

In [ ]:

```
hospital_df = pd.read_excel('hospitals.xlsx').drop(columns=['Unnamed: 0'])
```

In [ ]:

```
hospital_df
```

reading supermarkets file

In [ ]:

```
supermarkets_df = read_pdf('supermarkets.pdf',pages = "all") # reading all pages at the
same time
# List of separate pages of the dataframe
comb_dfs = [supermarkets_df[0], supermarkets_df[1], supermarkets_df[2], supermarkets_df
[3], supermarkets_df[4]]
# combining all the pages dataframes into one; And removing unwanted columns
supermarkets_df = pd.concat(comb_dfs).reset_index().drop(columns=['Unnamed: 0','index'
])
supermarkets_df
```

reading real state json file

In [ ]:

```
real_state_json = pd.read_json(r'real_state.json')
real_state_json
```

I have noticed that some adresses are in full upper case, so im going to transform these as per output file and capitalize first letter of each word

In [ ]:

```
for i in range(len(real_state_json)):
    real_state_json.iloc[i,3] = real_state_json.iloc[i,3].title()
```

reading real state xml file

The xml file doesnt have proper formatting, thus i need to read the whole Xml file as a string then using regex only keep the relevant information. the steps are:

1. first dividing the xml string content into substrings using regex
2. for each of the substrings finding all the matching data (using regex) items and storing them into a list
3. combining the lists into a pandas dataframe

In [ ]:

```
# reading the xml file as string
xml_file = open('real_state.xml', 'r', encoding="UTF-8")
xml_file = xml_file.read()


text = xml_file # this is the xml file read as an entire string

# dividing the xml string content into substrings using regex (for each of the attributes)
try:
    property_id = re.search('<property_id type="dict">(.+?)</property_id>', text).group(1)
    lat = re.search('<lat type="dict">(.+?)</lat>', text).group(1)
    lng = re.search('<lng type="dict">(.+?)</lng>', text).group(1)
    addr_street = re.search('<addr_street type="dict">(.+?)</addr_street>', text).group(1)
    price = re.search('<price type="dict">(.+?)</price>', text).group(1)
    property_type = re.search('<property_type type="dict">(.+?)</property_type>', text).group(1)
    year = re.search('<year type="dict">(.+?)</year>', text).group(1)
    bedrooms = re.search('<bedrooms type="dict">(.+?)</bedrooms>', text).group(1)
    bathrooms = re.search('<bathrooms type="dict">(.+?)</bathrooms>', text).group(1)
    parking_space = re.search('<parking_space type="dict">(.+?)</parking_space>', text).group(1)

except AttributeError:
    # AAA, ZZZ not found in the original string
    property_id = '' # apply your error handling
    lat = ''
    lng = ''
    addr_street = ''
    price = ''
    property_type = ''
    year = ''
    bedrooms = ''
    bathrooms = ''
    parking_space = ''


# for each of the substrings finding all the matching data items and storing them into a list
property_id = re.findall('int">(.+?)</', property_id)
lat = re.findall('float">(.+?)</', lat)
lng = re.findall('float">(.+?)</', lng)
addr_street = re.findall('str">(.+?)</', addr_street)
price = re.findall('int">(.+?)</', price)
property_type = re.findall('str">(.+?)</', property_type)
year = re.findall('int">(.+?)</', year)
bedrooms = re.findall('int">(.+?)</', bedrooms)
bathrooms = re.findall('int">(.+?)</', bathrooms)
parking_space = re.findall('int">(.+?)</', parking_space)

# combining the lists into a pandas dataframe
xml_df = pd.DataFrame(list(zip(property_id,lat,lng,addr_street,price,property_type,year,bedrooms,bathrooms,parking_space)),
                columns =['property_id', 'lat','lng','addr_street','price','property_type','year','bedrooms','bathrooms','parking_space'])

xml_df
```

Merging real state json and xml dataframes

In [ ]:

```
real_state_json.info()
```

Changing xml data schema to match json data schema - this will help when removing duplicates from the combined dataframe

In [ ]:

```
# changing xml data schema to match json data
xml_df["property_id"] = xml_df['property_id'].astype('int64')
xml_df["lat"] = xml_df['lat'].astype('float')
xml_df["lng"] = xml_df['lng'].astype('float')
xml_df["addr_street"] = xml_df['addr_street'].astype('object')
xml_df["price"] = xml_df['price'].astype('int64')
xml_df["property_type"] = xml_df['property_type'].astype('object')
xml_df["year"] = xml_df['year'].astype('int64')
xml_df["bedrooms"] = xml_df['bedrooms'].astype('int64')
xml_df["bathrooms"] = xml_df['bathrooms'].astype('int64')
xml_df["parking_space"] = xml_df['parking_space'].astype('int64')
```

Capitalize first letter of each word in adress

In [ ]:

```
for i in range(len(xml_df)):
    xml_df.iloc[i,3] = xml_df.iloc[i,3].title()
```

In [ ]:

```
xml_df
```

Merging real_state_json and xml_df dataframes and dropping duplicates

In [ ]:

```
# merging dataframes and dropping duplicates
real_state = pd.concat([real_state_json,xml_df]).drop_duplicates().reset_index().drop(c
olumns=['index'])
```

In [ ]:

```
real_state
```

Reading shopingcenters html file

In [ ]:

```
shopingcenters = pd.read_html('shopingcenters.html')[0]
shopingcenters=pd.DataFrame(shopingcenters)
```

In [ ]:

```
shopingcenters = shopingcenters.drop(columns=['Unnamed: 0']) # dropping unnecesary colu
mn
shopingcenters
```

This is a function to calculater the Euclidean distance between 2 coordinates using the latitude and longitude values of 2 coordinates.

In [ ]:

```
# https://gist.github.com/rochacbruno/2883505?fbclid=IwAR1YyJJdaF625MjH1YtymEdHVXw_kW5Z
k4fTMMeOKKBirS-vVx32XiBLPxk

import math

def distance(origin_location, destination_location): # takes lats and longs of 2 locati
ons
    lat1, lon1 = origin_location
    lat2, lon2 = destination_location
    radius = 6378 # earths radius in kilometers

    dlat = math.radians(lat2-lat1) # convert difference of lat to radians
    dlon = math.radians(lon2-lon1) # convert difference of lon to radians
    a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(math.radians(lat1)) \
        * math.cos(math.radians(lat2)) * math.sin(dlon/2) * math.sin(dlon/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    dist = radius * c # Euclidean distance between 2 locations

    return dist
```

In [ ]:

```
real_state
```

In [ ]:

```
# converting appropriate columns to numeric
num_list = ["lat", "lng",'price','year', 'bedrooms','bathrooms','parking_space']
real_state[num_list] = real_state[num_list].apply(pd.to_numeric)
real_state.dtypes
```

Working on the shopping centre distance and id

This is an empty nested dictionary with empty lists as values for storing all the shopping centre and distances for each property

In [ ]:

```
# creating a dictionary of 2009 empty lists for the corresponding rows in the real_stat
e dataframe
from collections import defaultdict

keys = ["sc_id", "Distance_to_sc"] # keys of the nested disctionary



my_dicts = defaultdict(dict) # nested dicts

for i in range(len(real_state)):
    my_dicts['dict_' + str(i)] = my_dicts['dict_' + str(i)].fromkeys(keys)
    my_dicts['dict_' + str(i)]['Distance_to_sc'] = []
    my_dicts['dict_' + str(i)]['sc_id'] = []
my_dicts
```

final nested dict to store shopping centre and closest distance

In [ ]:

```
closets_sc_dicts = defaultdict(dict)

for i in range(len(real_state)):
    closets_sc_dicts['dict_' + str(i)] = closets_sc_dicts['dict_' + str(i)].fromkeys(ke
ys)

closets_sc_dicts
```

Firstly entering all the shopping centres and their distance from property to shopping centre. Then finding the minimum distance, index of minimun distance as well as the respective shoppinfg centre id. then appending the shopping centre id and distance of the closest shopping centre from a property into the nested dictionary closets_sc_dicts

In [ ]:

```
for row in range(len(real_state)):
    lat = real_state.iloc[row,1] # latitude for each property
    lng = real_state.iloc[row,2] # longitude for each property
    for sc in range(len(shopingcenters)):
        sc_lat = shopingcenters.iloc[sc,1] # lat of each shopping centre
        sc_lng = shopingcenters.iloc[sc,2] # long of each shopping centre
        my_dicts['dict_' + str(row)]['sc_id'].append(shopingcenters.iloc[sc,0]) # appen
ding each of the sc_id into the list
        my_dicts['dict_' + str(row)]['Distance_to_sc'].append(distance((sc_lat,sc_lng),
(lat,lng))) # appending each distance into a list

    min_dist = min(my_dicts['dict_' + str(row)]['Distance_to_sc']) # minimum distance a
mongst all shopping centres
    min_index = my_dicts['dict_' + str(row)]['Distance_to_sc'].index(min(my_dicts['dict
_' + str(row)]['Distance_to_sc'])) # index
    closest_sc = my_dicts['dict_' + str(row)]['sc_id'][min_index] # using the index to
 get the sc_id of the closest sc
    closets_sc_dicts['dict_' + str(row)]['sc_id'] = closest_sc # inputting the sc_id of
the closest sc
    closets_sc_dicts['dict_' + str(row)]['Distance_to_sc'] = min_dist # inputting dista
nce to nearest shopping centre


closets_sc_dicts
```

In [ ]:

```
real_state
```

Appending Shopping_center_ids from the earlier dictionary

In [ ]:

```
real_state["Shopping_center_id"] = ""
real_state["Distance_to_sc"] = 1.333 # only stetting to this value to set as float type
will be replace by actual values
real_state['Distance_to_sc'] = real_state['Distance_to_sc'].astype(float)
```

In [ ]:

```
closets_sc_dicts['dict_' + str(0)]['sc_id']
```

In [ ]:

```
for i in range(len(real_state)):
    real_state.iloc[i,10] = closets_sc_dicts['dict_' + str(i)]['sc_id']
    real_state.iloc[i,11] = closets_sc_dicts['dict_' + str(i)]['Distance_to_sc']
```

Here i am inputting the sc_id and Distance_to_sc from the dictionary above into appropriate columns of the real_state dataframe

Working on the hospital id and distance to nearest hospital

In [ ]:

```
# creating a dictionary of 2009 empty lists for the corresponding rows in the real_stat
e dataframe
from collections import defaultdict

hos_keys = ["Hospital_id", "Distance_to_hospital"] # keys of the dict



my_dicts_h = defaultdict(dict) # nested dictionary

for i in range(len(real_state)):
    my_dicts_h['dict_' + str(i)] = my_dicts_h['dict_' + str(i)].fromkeys(hos_keys)
    my_dicts_h['dict_' + str(i)]['Distance_to_hospital'] = []
    my_dicts_h['dict_' + str(i)]['Hospital_id'] = []

# Final nested dict for hospitals
closets_hos_dicts = defaultdict(dict)

for i in range(len(real_state)):
    closets_hos_dicts['dict_' + str(i)] = closets_hos_dicts['dict_' + str(i)].fromkeys(
hos_keys)




for row in range(len(real_state)):
    lat = real_state.iloc[row,1] # latitude for each property
    lng = real_state.iloc[row,2] # longitude for each property
    for hos in range(len(hospital_df)):
        hos_lat = hospital_df.iloc[hos,1] # lat of each hospital
        hos_lng = hospital_df.iloc[hos,2] # long of each hospital
        my_dicts_h['dict_' + str(row)]['Hospital_id'].append(hospital_df.iloc[hos,0]) #
appending each of the Hospital_id into the list
        my_dicts_h['dict_' + str(row)]['Distance_to_hospital'].append(distance((hos_lat
,hos_lng),(lat,lng))) # appending each distance into a list

    min_dist = min(my_dicts_h['dict_' + str(row)]['Distance_to_hospital']) # minimum di
stance amongst all hospitals
    min_index = my_dicts_h['dict_' + str(row)]['Distance_to_hospital'].index(min(my_dic
ts_h['dict_' + str(row)]['Distance_to_hospital'])) # index
    closest_hos = my_dicts_h['dict_' + str(row)]['Hospital_id'][min_index] # using the
 index to get the Hospital_id of the closest hospital
    closets_hos_dicts['dict_' + str(row)]['Hospital_id'] = closest_hos # inputting the
 Hospital_id of the closest hospital
    closets_hos_dicts['dict_' + str(row)]['Distance_to_hospital'] = min_dist # inputtin
g distance to nearest hospital

#creating empty column

real_state["Hospital_id"] = ""
real_state["Distance_to_hospital"] = 1.333 # only stetting to this value to set as floa
t type will be replace by actual values
real_state['Distance_to_hospital'] = real_state['Distance_to_hospital'].astype(float)

# appending Hospital_id and Distance_to_hospital from the nested dictionary closets_hos
_dicts into the real_state dataframe

for i in range(len(real_state)):
```

```
    real_state.iloc[i,12] = closets_hos_dicts['dict_' + str(i)]['Hospital_id']
    real_state.iloc[i,13] = closets_hos_dicts['dict_' + str(i)]['Distance_to_hospital']
```

Working on the Supermarket_id and Distance_to_supermaket

In [ ]:

```
# creating a dictionary of 2009 empty lists for the corresponding rows in the real_stat
e dataframe
from collections import defaultdict

sup_keys = ["Supermarket_id", "Distance_to_supermaket"] #keys for the dict



my_dicts_s = defaultdict(dict) # nested dict

for i in range(len(real_state)):
    my_dicts_s['dict_' + str(i)] = my_dicts_s['dict_' + str(i)].fromkeys(sup_keys)
    my_dicts_s['dict_' + str(i)]['Distance_to_supermaket'] = []
    my_dicts_s['dict_' + str(i)]['Supermarket_id'] = []

# Final nested dict for hospitals
closets_supermarket_dicts = defaultdict(dict)

for i in range(len(real_state)):
    closets_supermarket_dicts['dict_' + str(i)] = closets_supermarket_dicts['dict_' + s
tr(i)].fromkeys(sup_keys)




for row in range(len(real_state)):
    lat = real_state.iloc[row,1] # latitude for each property
    lng = real_state.iloc[row,2] # longitude for each property
    for sup in range(len(supermarkets_df)):
        sup_lat = supermarkets_df.iloc[sup,1] # lat of each supermarket
        sup_lng = supermarkets_df.iloc[sup,2] # long of each supermarket
        my_dicts_s['dict_' + str(row)]['Supermarket_id'].append(supermarkets_df.iloc[su
p,0]) # appending each of the Supermarket_id into the list
        my_dicts_s['dict_' + str(row)]['Distance_to_supermaket'].append(distance((sup_l
at,sup_lng),(lat,lng))) # appending each distance into a list

    min_dist = min(my_dicts_s['dict_' + str(row)]['Distance_to_supermaket']) # minimum
 distance amongst all supermarkets
    min_index = my_dicts_s['dict_' + str(row)]['Distance_to_supermaket'].index(min(my_d
icts_s['dict_' + str(row)]['Distance_to_supermaket'])) # index
    closest_supermarket = my_dicts_s['dict_' + str(row)]['Supermarket_id'][min_index] #
using the index to get the Supermarket_id of the closest supermarket
    closets_supermarket_dicts['dict_' + str(row)]['Supermarket_id'] = closest_supermark
et # inputting the Supermarket_id of the closest supermarket
    closets_supermarket_dicts['dict_' + str(row)]['Distance_to_supermaket'] = min_dist
# inputting distance to nearest supermarket

#creating empty column

real_state["Supermarket_id"] = ""
real_state["Distance_to_supermaket"] = 1.333 # only stetting to this value to set as fl
oat type will be replace by actual values
real_state['Distance_to_supermaket'] = real_state['Distance_to_supermaket'].astype(floa
t)

# appending Supermarket_id and Distance_to_supermaket from the nested dictionary closet
s_supermarket_dicts into the real_state dataframe

for i in range(len(real_state)):
```

```
    real_state.iloc[i,14] = closets_supermarket_dicts['dict_' + str(i)]['Supermarket_id']
    real_state.iloc[i,15] = closets_supermarket_dicts['dict_' + str(i)]['Distance_to_supermaket']
```

In [ ]:

```
real_state
```

melbourne train info files

In [ ]:

```
agency = pd.read_csv('agency.txt')
calendar = pd.read_csv('calendar.txt')
calendar_dates = pd.read_csv('calendar_dates.txt')
routes = pd.read_csv('routes.txt')
shapes = pd.read_csv('shapes.txt')
stop_times = pd.read_csv('stop_times.txt')
stops = pd.read_csv('stops.txt')
trips = pd.read_csv('trips.txt')
```

In [ ]:

```
len(stops)
```

In [ ]:

```
stops
```

Working on the Train_station_id and Distance_to_train_station

In [ ]:

```
# creating a dictionary of 2009 empty lists for the corresponding rows in the real_stat
e dataframe
from collections import defaultdict

train_keys = ["Train_station_id", "Distance_to_train_station"] # keys of the dict



my_dicts_t = defaultdict(dict) # nested dict

for i in range(len(real_state)):
    my_dicts_t['dict_' + str(i)] = my_dicts_t['dict_' + str(i)].fromkeys(train_keys)
    my_dicts_t['dict_' + str(i)]['Distance_to_train_station'] = []
    my_dicts_t['dict_' + str(i)]['Train_station_id'] = []

# Final nested dict for hospitals
closest_train_dicts = defaultdict(dict)

for i in range(len(real_state)):
    closest_train_dicts['dict_' + str(i)] = closest_train_dicts['dict_' + str(i)].fromk
eys(train_keys)




for row in range(len(real_state)):
    lat = real_state.iloc[row,1] # latitude for each property
    lng = real_state.iloc[row,2] # longitude for each property
    for train in range(len(stops)):
        train_lat = stops.iloc[train,3] # lat of each train stop
        train_lng = stops.iloc[train,4] # long of each train stop
        my_dicts_t['dict_' + str(row)]['Train_station_id'].append(stops.iloc[train,0])
# appending each of the Train_station_id into the list
        my_dicts_t['dict_' + str(row)]['Distance_to_train_station'].append(distance((tr
ain_lat,train_lng),(lat,lng))) # appending each distance into a list

    min_dist = min(my_dicts_t['dict_' + str(row)]['Distance_to_train_station']) # minim
um distance amongst all train stations
    min_index = my_dicts_t['dict_' + str(row)]['Distance_to_train_station'].index(min(m
y_dicts_t['dict_' + str(row)]['Distance_to_train_station'])) # index
    closest_train = my_dicts_t['dict_' + str(row)]['Train_station_id'][min_index] # usi
ng the index to get the Train_station_id of the closest train station
    closest_train_dicts['dict_' + str(row)]['Train_station_id'] = closest_train # input
ting the Train_station_id of the closest train station
    closest_train_dicts['dict_' + str(row)]['Distance_to_train_station'] = min_dist # i
nputting distance to nearest train station

#creating empty column

real_state["Train_station_id"] = ""
real_state["Distance_to_train_station"] = 1.333 # only stetting to this value to set as
float type will be replace by actual values
real_state['Distance_to_train_station'] = real_state['Distance_to_train_station'].astyp
e(float)

# appending Train_station_id and Distance_to_train_station from the nested dictionary c
losest_train_dicts into the real_state dataframe

for i in range(len(real_state)):
```

```
    real_state.iloc[i,16] = closest_train_dicts['dict_' + str(i)]['Train_station_id']
    real_state.iloc[i,17] = closest_train_dicts['dict_' + str(i)]['Distance_to_train_st
ation']
```

In [ ]:

```
real_state
```

In [ ]:

```

```

Finding suburb

In [ ]:

```
sf = shapefile.Reader("./VIC_LOCALITY_POLYGON_shp") # reading from the shapefiles after
unzipping
recs = sf.records() # the records of shape data info stored as lists
shapes = sf.shapes() # shapefile objects
```

In [ ]:

```
real_state['suburb'] = "" # empty column for suburb
real_state
```

Code for finding the matching suburbs from the shapefile given to us. steps:

1. Iterate through the shapefile records and turn the points into polygons for e
ach record
2. For each of the polygons in the shapefile, iterate through the real_state dat
aframe (where we have all the properties)
3. Take the lng and lat for each property and check if its within the boundary o
f the popygon(i.e. suburb)
4. If the location of the property is within the polygon of the suburb, extract
 the suburb name from the records (6th index of the list) and using the capitali
ze function turn first alphabet capital and rest lower case (to match output fil
e)
5. Input the suburb name in the suburn column for the appropritate location usin
g the index

In [ ]:

```
from shapely.geometry import Point, Polygon
```

In [ ]:

```
for i in range(len(sf)):
    pointss = shapes[i].points
    ploygons = Polygon(pointss)


    for k in range(len(real_state)):
        if Point(real_state.iloc[k,2],real_state.iloc[k,1]).within(ploygons):
            real_state.iloc[k,18] = recs[i][6].capitalize()
```

In [ ]:

```
real_state
```

In [ ]:

```
stop_times
```

travel_min_to_CBD (15%)

filtering out departure time from 7 to 9 am for stop_times dataframe

Converting departure_time and arrival_time into string format

In [ ]:

```
stop_times['departure_time']= stop_times['departure_time'].astype(str)
stop_times['arrival_time']=stop_times['arrival_time'].astype(str)
```

Filter out the stop_times dataframe where departure time is between 7 and 9 am inclusive

In [ ]:

```
filtered_stop_times = stop_times[(stop_times['departure_time']>="07:00:00") & (stop_tim
es['departure_time']<="09:00:00")]
```

Sorting by trip_id and stop_sequence

In [ ]:

```
filtered_stop_times = filtered_stop_times.sort_values(by = ['trip_id','stop_sequence'])
filtered_stop_times
```

In [ ]:

```
trips
```

only taking weekdays calender data

In [ ]:

```
filtered_calendar = calendar
filtered_calendar['count'] = ""
filtered_calendar['count'] = filtered_calendar['monday']+filtered_calendar['tuesday']+f
iltered_calendar['wednesday']+filtered_calendar['thursday']+filtered_calendar['friday']
```

Keeping only the services which operate from monday to friday as per requirements

In [ ]:

```
filtered_calendar = filtered_calendar[filtered_calendar['count']!=0] # removing 0 count
s
filtered_calendar
```

Filtering the trips dataframe and only keeping rows with match service_id as per the filtered_calendar dataframe service_id (i.e. services that run from monday to friday only)

Filtering further to keep direction_id =0 rows, this is done as direction_id = 0 means the trip is going towards flinders street, which is what we are after

In [ ]:

```
filtered_trips = trips
filtered_trips = filtered_trips[filtered_trips["service_id"].isin(filtered_calendar['se
rvice_id'].tolist())]
filtered_trips = filtered_trips[filtered_trips['direction_id'] == 0] #trips to flinder
 street
filtered_trips
```

Filtering the filtered_stop_times dataframe to keep only the rows which matches the trip_id of filtered_trips dataframe

In [ ]:

```
filtered_stop_times = filtered_stop_times[filtered_stop_times['trip_id'].isin(filtered_
trips["trip_id"].tolist())]
```

In [ ]:

```
filtered_stop_times
```

In [ ]:

```
stop_times[stop_times['trip_id'] == '17067231.T0.2-HBG-F-mjp-1.2.H']
```

Here i am calculating the arrival time to flinders street for each stop_id. Steps:

1. create empty list for storing the times
2. Iterate thorugh filtered_stop_times dataframe
3. store trip_id for each row in a variable
4. stopss is the rows in stop_times where the trip_id matches the trip_id form filtered_stop_times as well as the stop_id is filders street (which is 19854)
5. if length of stopss greater then 0, then append the second column from stopss which is the arrival time at flinders street into the list.
6. create a new column named arriving_in_flinders in filtered_stop_times, and input all the values form the list.
7. now we will have all the arriving times at flinders street for each of the stop_ids in the filtered_stop_times dataframe

In [ ]:

```
arriving_in_flinders = []

for i in range(len(filtered_stop_times)):
    trip = filtered_stop_times.iloc[i,0]
    stopss = stop_times[(stop_times['trip_id'] == trip) & (stop_times['stop_id']==19854
)]

    if len(stopss)>0:
        arriving_in_flinders.append(stopss.iloc[0,1])
    else:
        arriving_in_flinders.append('0')
```

In [ ]:

```
filtered_stop_times['arriving_in_flinders']=""
filtered_stop_times['arriving_in_flinders'] = arriving_in_flinders
```

In [ ]:

```
filtered_stop_times
```

In [ ]:

```
filtered_stop_times["travel_min_to_CBD"]=""
```

Here i am defining a function that calculates the time difference between 2 times in strings and returns the time difference in minutes.

In [ ]:

```
def time_diff(start,end): # takes 2 string inputs for times
    # using the datetime module converting the strings into datatime objects
    start_dt = dt.datetime.strptime(start, '%H:%M:%S')
    end_dt = dt.datetime.strptime(end, '%H:%M:%S')
    diff = (end_dt - start_dt) # time difference
    diff_in_minutes = diff.seconds/60 # seconds in the time diff divided by 60 gives us
minutes
    return diff_in_minutes
```

for each row in filtered_stop_times if arriving_in_flinders is not '0' then calculate the time diff and append into the new column travel_min_to_CBD, else set the value to 0.

In [ ]:

```
for i in range(len(filtered_stop_times)):
    if filtered_stop_times.iloc[i,9] != '0':
        filtered_stop_times.iloc[i,10] = time_diff(filtered_stop_times.iloc[i,2], filte
red_stop_times.iloc[i,9])
    else:
        filtered_stop_times.iloc[i,10] = 0
```

In [ ]:

```
filtered_stop_times
```

Removing indirect train routes

In [ ]:

```
## Removing indirect train routes
filtered_stop_times = filtered_stop_times[filtered_stop_times['arriving_in_flinders']>=
filtered_stop_times['departure_time']]
```

travel_min_to_CBD to numeric

In [ ]:

```
filtered_stop_times["travel_min_to_CBD"] = filtered_stop_times["travel_min_to_CBD"].app
ly(pd.to_numeric)
```

Here i am grouping filtered_stop_times by stop_id and getting the mean

Thus for each stop id i will have the average travel time (for example if there are 3 stop ids it will add the travel_min_to_CBD of the 3 stops and divide by 3 as .mean() function gets the average)

In [ ]:

```
dummy_df = filtered_stop_times.groupby('stop_id',as_index=False).mean() # avg duration
 for each stop
dummy_df
```

Turning the dummy_df dataframe into a dictionary taking the stop_id as key and travel_min_to_CBD as value

In [ ]:

```
dummy_dict = dummy_df.set_index('stop_id')['travel_min_to_CBD'].to_dict() #dictionary o
f stops and avg travel duration
dummy_dict
```

In [ ]:

```
dummy_dict[15351]
```

Creating an empty column to to store the information from the above dictionary into the real_state dataframe

In [ ]:

```
real_state['travel_min_to_CBD'] = ""
```

In [ ]:

```
real_state
```

For each of the rows in the real_state dataframe; if the Train_station_id is in the keys of the dummy_dict (i.e. matches the stop_id), append the corresponding value from the dummy_dict into the travel_min_to_CBD attribute of the real_state dataframe. Else just append 0.

In [ ]:

```
for i in range(len(real_state)):
    if real_state.iloc[i,16] in dummy_dict.keys():
        real_state.iloc[i,19] = dummy_dict[real_state.iloc[i,16]]
    else:
        real_state.iloc[i,19] = 0
```

In [ ]:

```
real_state
```

In [ ]:

```
real_state[real_state['travel_min_to_CBD']==0]
```

Transfer flag

In [ ]:

```
real_state['Transfer_flag']="" # empty Transfer_flag column in the real_state dataframe
```

Setting transfer flag to 0 if there is a direct trip (travel time not 0) or set to 1 if there is no direct trip.

In [ ]:

```
# setting transfer flag to 0 if there is a direct trip (travel time not 0) or set to 1
 if there is no direct trip.
for i in range(len(real_state)):
    if real_state.iloc[i,19] != 0:
        real_state.iloc[i,20] = 0
    else:
        real_state.iloc[i,20] = 1
```

In [ ]:

```
real_state
```

In [ ]:

```
real_state.columns
```

Rearranging the dataframe columns according to the sample output

In [ ]:

```
real_state = real_state[['property_id','lat', 'lng', 'addr_street','suburb','price', 'p
roperty_type',
       'year', 'bedrooms', 'bathrooms', 'parking_space', 'Shopping_center_id',
       'Distance_to_sc','Train_station_id',
       'Distance_to_train_station','travel_min_to_CBD',
       'Transfer_flag','Hospital_id', 'Distance_to_hospital','Supermarket_id', 'Distanc
e_to_supermaket']]
```

In [ ]:

```
real_state
```

Writing the final dataframe into a csv file

In [ ]:

```
real_state.to_csv('31043313_A3_solution.csv',index=False)
```

Task 2: data reshaping (20%)

Data reshaping can be a valuable tool to ensure raw data can be reshaped and suitable for various data mining techniques. in our case we are looking to reshape the data (trying out various reshaping methods) to make the data suitable for a linear regression model which would be capable of predicting the price of a house by using the distance to nearest shoppoing centre and hospital as well as travel time to the city.

2 ways data can be reshaped.

1. normalization
2. transformation

Data normalization refers to the process of transforming raw data into a different form which is more suitable of the purpose of modelling and analysis. scaling normalization and standardization are the 2 main ways data can be normalized.

1. Scaling normalization refers to rescaling the data into a specific range. common methods are min-max normalizaiton and scaling normalizaiton.
2. Standardization focusses on shifting the distribution of the data. the raw data becomes z-scores with a mean of 0 and standard deviation of 1.

Taking only the required columns for data reshaping

In [ ]:

```
reshape_data = real_state[['price', 'Distance_to_sc','Distance_to_hospital','travel_min_to_CBD']]
```

In [ ]:

```
reshape_data
```

In [ ]:

```
reshape_data["travel_min_to_CBD"]=reshape_data["travel_min_to_CBD"].apply(pd.to_numeric)
```

In [ ]:

```
reshape_data.dtypes
```

In [ ]:

```
reshape_data.describe()
```

In [ ]:

```
from pylab import rcParams
rcParams['figure.figsize'] = 5, 5
reshape_data['price'].hist()
```

In [ ]:

```
rcParams['figure.figsize'] = 5, 5
reshape_data['Distance_to_sc'].hist()
```

In [ ]:

```
rcParams['figure.figsize'] = 5, 5
reshape_data['Distance_to_hospital'].hist()
```

In [ ]:

```
rcParams['figure.figsize'] = 5, 5
reshape_data['travel_min_to_CBD'].hist()
```

From the histograms we can see that the attributes are right skewed

Sqrt transformation on continous right skewed data ('Distance_to_sc','Distance_to_hospital','travel_min_to_CBD')

In [ ]:

```
reshape_data['sc_root'] = None
i = 0
for row in reshape_data.iterrows():
    reshape_data['sc_root'].at[i] = math.sqrt(reshape_data["Distance_to_sc"][i])
    i += 1


reshape_data['hos_root'] = None
i = 0
for row in reshape_data.iterrows():
    reshape_data['hos_root'].at[i] = math.sqrt(reshape_data["Distance_to_hospital"][i])
    i += 1


reshape_data['cbd_root'] = None
i = 0
for row in reshape_data.iterrows():
    reshape_data['cbd_root'].at[i] = math.sqrt(reshape_data["travel_min_to_CBD"][i])
    i += 1
```

In [ ]:

```
reshape_data
```

In [ ]:

```
rcParams['figure.figsize'] = 5, 5
reshape_data['sc_root'].hist()
```

Improvement after using root transformation

In [ ]:

```
rcParams['figure.figsize'] = 5, 5
reshape_data['hos_root'].hist()
```

Improvement after using root transformation

In [ ]:

```
rcParams['figure.figsize'] = 5, 5
reshape_data['cbd_root'].hist()
```

No Improvement after using root transformation

trying log transformation on travel_min_to_CBD

In [ ]:

```
reshape_data['cbd_log'] = None
i = 0
for row in reshape_data.iterrows():
    reshape_data['cbd_log'].at[i] = math.log(reshape_data["travel_min_to_CBD"][i]+1) #
 adding 1 for the 0 values
    i += 1
```

In [ ]:

```
reshape_data['cbd_log'].hist()
```

Not improved at all again

Trying out root transformation on price

In [ ]:

```
reshape_data['price_root'] = None
i = 0
for row in reshape_data.iterrows():
    reshape_data['price_root'].at[i] = math.sqrt(reshape_data["price"][i])
    i += 1
```

In [ ]:

```
rcParams['figure.figsize'] = 5, 5
reshape_data['price_root'].hist()
```

not a huge improvement

Trying out log transformation on price

In [ ]:

```
reshape_data['price_log'] = None
i = 0
for row in reshape_data.iterrows():
    reshape_data['price_log'].at[i] = math.log(reshape_data["price"][i])
    i += 1
```

In [ ]:

```
rcParams['figure.figsize'] = 5, 5
reshape_data['price_log'].hist()
```

Here we can see much improvement as the data for price is close to normal distribution now which is good for linear regression

In [ ]:

```
plt.scatter(reshape_data['sc_root'],reshape_data['price_root'])
```

In [ ]:

```
reshape_data.columns
```

Linear regression example

In [ ]:

```
X = reshape_data[['sc_root','hos_root','cbd_root']]
Y = reshape_data['price_log']

x_train, x_test,y_train,y_test = train_test_split(X,Y,test_size =0.1,random_state=1995)
# print the data
from sklearn.linear_model import LinearRegression
clf = LinearRegression()
clf.fit(x_train,y_train)
clf.predict(x_test)
clf.score(x_test,y_test)
```

Z-Score Normalisation (standardisation)

In [ ]:

```
standard_scale = preprocessing.StandardScaler().fit(reshape_data[['price', 'Distance_to
_sc','Distance_to_hospital','travel_min_to_CBD']])
df_std = standard_scale.transform(reshape_data[['price', 'Distance_to_sc','Distance_to_
hospital','travel_min_to_CBD']]) # an array not a df
df_std
```

In [ ]:

```
# put it alongside data... to view
reshape_data['price_scaled'] = df_std[:,0]
reshape_data['Distance_to_sc_scaled'] = df_std[:,1]
reshape_data['Distance_to_hospital_scaled'] = df_std[:,2]
reshape_data['travel_min_to_CBD_scaled'] = df_std[:,3]
reshape_data
```

In [ ]:

```
reshape_data.describe() # check that μ = 0 and σ = 1... approx
```

In [ ]:

```
%matplotlib inline
from pylab import rcParams
rcParams['figure.figsize'] = 20, 10
```

In [ ]:

```
reshape_data["Distance_to_hospital"].plot(), reshape_data["Distance_to_sc"].plot()
```

In [ ]:

```
reshape_data["Distance_to_hospital_scaled"].plot(), reshape_data["Distance_to_sc_scale
d"].plot()
```

In [ ]:

```
reshape_data["Distance_to_sc"].plot(), reshape_data["Distance_to_sc_scaled"].plot()
```

In [ ]:

```
reshape_data
```

In [ ]:

```
X = reshape_data[['Distance_to_sc_scaled','Distance_to_hospital_scaled','travel_min_to_
CBD_scaled']]
Y = reshape_data['price_scaled']

x_train, x_test,y_train,y_test = train_test_split(X,Y,test_size =0.1,random_state=1995)
# print the data
from sklearn.linear_model import LinearRegression
clf = LinearRegression()
clf.fit(x_train,y_train)
clf.predict(x_test)
clf.score(x_test,y_test)
```

MinMax Noramlisation

In [ ]:

```
minmax_scale = preprocessing.MinMaxScaler().fit(reshape_data[['price', 'Distance_to_sc'
,'Distance_to_hospital','travel_min_to_CBD']])
df_minmax = minmax_scale.transform(reshape_data[['price', 'Distance_to_sc','Distance_to
_hospital','travel_min_to_CBD']])
df_minmax
```

In [ ]:

```
reshape_data['price_scaled'] = df_minmax[:,0]
reshape_data['Distance_to_sc_scaled'] = df_minmax[:,1]
reshape_data['Distance_to_hospital_scaled'] = df_minmax[:,2]
reshape_data['travel_min_to_CBD_scaled'] = df_minmax[:,3]
reshape_data
```

Linear regression

In [ ]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
```

In [ ]:

```
plt.scatter(reshape_data['travel_min_to_CBD'],reshape_data['price'])
```

In [ ]:

```
X = reshape_data[['Distance_to_sc_scaled','Distance_to_hospital_scaled','travel_min_to_
CBD_scaled']]
Y = reshape_data['price_scaled']
```

In [ ]:

```
x_train, x_test,y_train,y_test = train_test_split(X,Y,test_size =0.1)
# print the data
x_train
```

In [ ]:

```
y_train
```

In [ ]:

```
from sklearn.linear_model import LinearRegression
clf = LinearRegression()
clf.fit(x_train,y_train)
clf.predict(x_test)
clf.score(x_test,y_test)
```

In [ ]:

```
### not very good accuracy score
```

Data Transformation

Log transformation

In [ ]:

```
reshape_data = real_state[['price', 'Distance_to_sc','Distance_to_hospital','travel_min
_to_CBD']]
reshape_data = reshape_data[reshape_data['travel_min_to_CBD']!=0]
reshape_data = reshape_data.reset_index()
reshape_data
```

In [ ]:

```
import math
reshape_data['log_sc'] = None
i = 0
for row in reshape_data.iterrows():
    reshape_data['log_sc'].at[i] = math.log(reshape_data["Distance_to_sc"][i])
    i += 1
```

In [ ]:

```
reshape_data['log_hos'] = None
i = 0
for row in reshape_data.iterrows():
    reshape_data['log_hos'].at[i] = math.log(reshape_data["Distance_to_hospital"][i])
    i += 1
```

In [ ]:

```
reshape_data['log_cbd'] = None
i = 0
for row in reshape_data.iterrows():
    reshape_data['log_cbd'].at[i] = math.log(reshape_data["travel_min_to_CBD"][i]+1)
    i += 1
```

In [ ]:

```
reshape_data['log_price'] = None
i = 0
for row in reshape_data.iterrows():
    reshape_data['log_price'].at[i] = math.log(reshape_data["price"][i])
    i += 1
```

In [ ]:

```
plt.scatter(reshape_data['log_hos'],reshape_data['log_price'])
```

In [ ]:

```
plt.scatter(reshape_data['log_cbd'],reshape_data['log_price'])
```

In [ ]:

```
plt.scatter(reshape_data['log_sc'],reshape_data['log_price'])
```

there is no apparant linear relationship between predictor and target variable

In [ ]:

```
X = reshape_data[['log_sc','log_hos','log_cbd']]
Y = reshape_data['log_price']

x_train, x_test,y_train,y_test = train_test_split(X,Y,test_size =0.1,random_state=1995)
# print the data
from sklearn.linear_model import LinearRegression
clf = LinearRegression()
clf.fit(x_train,y_train)
clf.predict(x_test)
clf.score(x_test,y_test)
```

Conclusions:

I have tried a few different methods such as root transformation, log transformation, standardization, min_max normalization etc. The data we have is right skewed for all the attributes. Although, applying transformation on the attributes was somewhat successful in terms of satisfying the normality assumption of linear regression, linearity is not achievable as seen above. This is due to the fact that the predictor variables are not suitable to predict the target variable (no linear correlation between predictors and target variable what so ever). Even a train test split to build a linear regression model yielded very low accuracy score for each of the transformations i did. Although in theory, distance to nearest hospital, distance to shopping centre and travel time to cbd should be good factors affecting the price of a property, with the data we have it is hard to see any kind of relationship between target and predictors.

Almost all suburbs in melbourne (in the data we have) have close shopping centres and hospitals, thus distances from shopping centres and hospitals are not significantly different for each property which means there is no apparant correlation. Also, in the case of travel min to cbd, people have other method of transportation if travelling to cbd by train takes too long.

In conclusion, using the data we have it is difficult to build a linear regression model using the predictors we have (having little to no correlation with the target variable).