# 31901611_ass3

February 21, 2021

## 1 FIT5196 Assessment 3

**Student Name: Prashasti Garg**

**Student ID: 31901611** Date: 18/02/2021

Version: 1.0

Environment: Python 3.7.9 and Jupyter notebook

Libraries used: please include the main libraries you used in your assignment here, e.g.,: * `pandas` (for data manipulation and interpretation) * `html5lib` (for reading .html file) * `tabula` (for reading .pdf file) * `PyPDF2` (for working with .pdf file) * `json` (for reading .json file) * `xmltodict` (for reading .xml file) * `lxml` (for parsing xml and html file) * `xml.etree.ElementTree` (for implementing a simple and efficient for parsing and creating xml data) * `math` (for solving mathematical functions) * `shapefile` (for providing reading and writing support to the shapefile) * `shapely.geometry` (for working with shape file) * `datetime` (for working with date and time)

### 1.1 TASK 01 - Data Integration

### 1.2 Introduction

Data Integration is the process of combining the data from all different sources in a single, unified view. In Task 01, we have been provided seven files. These file are in .json, .xml, .html, .xlsx, .pdf and two in .zip format. The task is to extract data from these seven file and to create a .csv file which would have 21 columns. These 21 columns are: 1. **Property_id**: A unique id for the property 2.**lat**: The property latitude. 3.**lng**: The property longitude. 4.**addr_street**: The property address 5.**suburb**: The property suburb. Default value: "not available". 6.**price**: The property price. 7.**property_type**: The type of the property. 8.**year**: Year of sold. 9.**bedrooms**: Number of bedrooms. 10.**bathrooms**: Number of bathrooms. 11.**parking_space**: The number of parking space of the property. 12.**Shopping_center_id**: The closest shopping centre to the property. Default value: "not available". 13.**Distance_to_sc**: The Haversine Distance (3 decimal places with +-0.001 tolerance) from the closest shopping centre to the property. Default value: 0. 14.**Train_station_id**: The closest train station to the property. Default value: 0. 15.**Distance_to_train_station**: The Haversine Distance (3 decimal places with +-0.001 tolerance) from the closest train station to the property. Default value: 0. 16.**travel_min_to_CBD**: The average travel time (minutes) from the closest train station to the "Flinders street" station on weekdays (i.e. Monday-Friday) departing between 7 to 9 am. For example, if there are 3 trip departing from the closest train station to the Flinders street station on weekdays between 7-9am and each take 6, 7, and 8 minutes respectively, then the value of this column for the property should

be (6+7+8)/3. If there are any direct transfers between the closest station and Flinders street station, only the average of direct transfers should be calculated. Default value: 0. 17.**Hospital_id**: The closest hospital to the property. Default value: "not available". 18.**Distance_to_hospital**: The Haversine Distance (3 decimal places with +-0.001 tolerance) from the closest hospital to the property. Default value: 0. 19.**Supermarket_id**: The closest supermarket to the property.Default value: "not available". 20.**Distance_to_supermaket**: The Haversine Distance (3 decimal places with +-0.001 tolerance) from the closest supermarket to the property. Default value: 0. 21.**Over_ave_price**: The Boolean value to the property. Write a True value if the price is higher than the average house price of its suburb. Otherwise, the value should be False. The average house price of each property can be calculated by averaging the total house price of its suburb. Default value: 0.

### 1.2.1 Importing Libraries

- The libraries which are required for extracting and integrating the data from the given files needs to be imported.

[ ]:
```
import pandas as pd
import json
import html5lib
import lxml
import xml.etree.ElementTree as ET
import math
import xmltodict
import PyPDF2
from shapely.geometry import shape,mapping, Point, asPoint
from tabula.io import read_pdf
import shapefile
from datetime import datetime, timedelta
```

### 1.2.2 Extracting attributes from real_state.json and real_state.xml file

- real_state.json and real_state.xml file contains 10 columns which are required.
- These both files contain these 10 columns, therefore a union of these files will be taken.
- Using pandas, real_state.json file is read which is later stored in 'df'.
- Using xmltodict and xm.etree.ElementTree real_state.xml file is read.
- Both these files are concatenated.

[ ]:
```
#real_state.json file is read
real_state = pd.read_json(r'real_state.json')
```

[ ]:
```
#len of real_state.json
len(real_state)
```

[ ]:
```
real_state.head()
```

- Checking null values in .json file.

[ ]:
```
real_state.isnull().sum()
```

[ ]:
```
#real_state.xml is read
xmlFile = open(r"D:\Jupyter Notebook\Wrangling\Assignment 03\real_state.xml")
```

[ ]:
```
#while reading \nb' is stripped
xmlData = xmlFile.read().strip("\nb'")
```

[ ]:
```
#xml file is parsed
root = ET.XML(xmlData)
data = []
cols = []
for i, child in enumerate(root):
    data.append([subchild.text for subchild in child])
    cols.append(child.tag)

xml_data = pd.DataFrame(data).T  # Write in DF and transpose it
xml_data.columns = cols  # Update column names
```

- Checking null values in .xml file

[ ]:
```
# len of real_state.xml
len(xml_data)
```

[ ]:
```
#checking whether there is any null in the file
xml_data.isnull().sum()
```

[ ]:
```
xml_data
```

- .json and .xml file is concatenated

[ ]:
```
merge_real = pd.concat([xml_data,real_state])
```

- Duplicates in merge_real, needs to be checked.

[ ]:
```
df = merge_real.drop_duplicates(keep = 'first')
```

- The indexes of column are not in order after merge, thus .reset_index() will reset the index of rows.

[ ]:
```
df = df.reset_index()
```

- Datatype of property_id from .json and .xml needs to be investigated, because there may be duplicated property_id in the 'df'. And making their datatype same will be helpful in removing duplicates.

[ ]:
```
print(type(df.loc[0, 'property_id']))
print(type(df.loc[1015, 'property_id']))
```

- The property_id in .xml file is in string data type.
- Thus, need to convert the datatype of property_id to numeric.

[ ]:
```
df['property_id'] = pd.to_numeric(df['property_id'], errors='coerce')
```

- Now that the datatype of property_id is int in 'df', the duplicates can be removed.

[ ]:
```
df.duplicated(subset='property_id')
```

- Using .duplicated(), I found about the rows which has duplicated values. thus using keep = 'first', I will keep only the first row of duplicate rows.

[ ]:
```
df[df.property_id == 24359]
```

[ ]:
```
df.drop_duplicates(subset=['property_id'], keep='first', inplace = True)
```

[ ]:
```
#again we have checked for duplicates
df[df.property_id == 24359]
```

- Now we don't have any duplicated rows.

[ ]:
```
df
```

- In task 01, we have 21 columns. We have extracted 10 columns above. We need to create other columns which will further be extracted, and need to assign default values to them.

[ ]:
```
df['suburb'] = 'not available'
df['Shopping_center_id'] = 'not available'
df['Distance_to_sc'] = 0
df['Train_station_id'] = 0
df['Distance_to_train_station'] = 0
df['travel_min_to_CBD'] = 0
df['Hospital_id'] = 'not available'
df['Distance_to_hospital'] = 0
df['Supermarket_id'] = 'not available'
df['Distance_to_supermaket'] = 0
df['Over_ave_price'] = 0
```

[ ]:
```
df = df[['property_id', 'lat', 'lng', 'addr_street',
         'suburb', 'price', 'property_type', 'year',
         'bedrooms', 'bathrooms', 'parking_space',
         'Shopping_center_id', 'Distance_to_sc',
         'Train_station_id', 'Distance_to_train_station',
         'travel_min_to_CBD', 'Hospital_id',
         'Distance_to_hospital', 'Supermarket_id', 'Distance_to_supermaket',␣
↪'Over_ave_price']]
```

- The numeric datatype of the following columns needs to be assigned.

[ ]:
```
df['lat'] = pd.to_numeric(df['lat'], errors='coerce')
df['lng'] = pd.to_numeric(df['lng'], errors='coerce')
df['price'] = pd.to_numeric(df['price'], errors='coerce')
df['year'] = pd.to_numeric(df['year'], errors='coerce')
df['bedrooms'] = pd.to_numeric(df['bedrooms'], errors='coerce')
df['bathrooms'] = pd.to_numeric(df['bathrooms'], errors='coerce')
df['parking_space'] = pd.to_numeric(df['parking_space'], errors='coerce')
```

- The columns are needed to be renamed as per the requirement.

[ ]:
```
df.columns = ['Property_id', 'lat', 'lng', 'addr_street',
         'suburb', 'price', 'property_type', 'year',
         'bedrooms', 'bathrooms', 'parking_space',
         'Shopping_center_id', 'Distance_to_sc',
         'Train_station_id', 'Distance_to_train_station',
         'travel_min_to_CBD', 'Hospital_id',
         'Distance_to_hospital', 'Supermarket_id', 'Distance_to_supermaket',␣
 ↪'Over_ave_price']
```

- Index of df needs to reset again, as we removed the duplicates so indices are again non-uniform.

[ ]:
```
df = df.reset_index()
```

- The 'index' column needs to be dropped, therefore .drop(columns = [' '], inplace = True)

[ ]:
```
df.drop(columns=['index'], inplace = True)
```

[ ]:
```
df
```

### 1.2.3 Extracting Hospital_id, Distance_to_hospital from hospital.xlsx file

- Hospital_id needs to be the closest shopping center to the respective property_id and Distance_to_hospital is the distance from it. It is to be extracted from hospitals.xlsx file.
- .xlsx file is read using pd.read_excel, where the unwanted columns have been dropped using .drop().
- Using haversine distance formula is used to calculate disctance between two pair of latitude and longitude, therefore created a function ie, haversine_distance.
- Another function get_DistId is created to find the nearest the hospital from a property.
- Thus, Hospital_id, Distance_to_hospital is calculated using get_DistId.

[ ]:
```
#.xlsx file is read
hospital_data = pd.read_excel("./hospitals.xlsx",engine = 'openpyxl').
 ↪drop(columns = ['name', 'Unnamed: 0']) #https://stackoverflow.com/questions/
 ↪26063231/read-specific-columns-with-pandas-or-other-python-module
```

[ ]:
```
hospital_data
```

[ ]:
```
# shape of hospital_data
hospital_data.shape
```

[ ]:
```
def haversine_distance(lat1, lon1, lat2, lon2):
    radius_of_earth = 6378
    #lat1,lon1 = property's coordinates
    # lat2, lon2 = coordinates of other location
    prop_lat = math.radians(lat1)
    prop_long = math.radians(lon1)
    another_lat = math.radians(lat2)
    another_long = math.radians(lon2)

    dlat = prop_lat - another_lat
    dlon = prop_long - another_long

    a = math.sin(dlat / 2)**2 + math.cos(prop_lat) * math.cos(another_lat) *␣
 ↪math.sin(dlon / 2)**2
    distance = radius_of_earth * (2 * math.atan2(math.sqrt(a), math.sqrt(1 -␣
 ↪a)))

    return round(distance,4)
```

- get_DistId function here takes 4 arguments: df, hospital_data, id, lat, lng.
- id, lat, lng are the columns of hospital_data.
- In this function, I have used minDistance which has been assigned infinite value. Its been used because any value will be smaller than it. So the smallest of values calculated for each row is saved over minDistance.

[ ]:
```
def get_DistId(df1, df2, col_id, lat, lng):
    temp = []
    for i,j in df1.iterrows():
        minDistance = math.inf
        temp_id = ""
        for x,y in df2.iterrows():
            distance =  haversine_distance(df1.loc[i,'lat'], df1.
 ↪loc[i,'lng'],df2.loc[x,lat], df2.loc[x,lng])
            if distance < minDistance:
                minDistance = distance
                temp_id = df2.loc[x,col_id]
        temp.append((temp_id, minDistance))
    return temp
```

[ ]:
```
#passing arguments in get_DistId function
DistId = get_DistId(df, hospital_data, 'id','lat','lng')
```

```
[ ]: #values calculated are populated in the Hospital_id and Distance_to_hospital
for i, j in df.iterrows():
    df.loc[i,'Hospital_id'], df.loc[i,'Distance_to_hospital'] = DistId[i][0],␣
 ↪DistId[i][1]
```

### 1.2.4 Extracting Shopping_center_id, Distance_to_sc from shopingcenters.html

- Shopping_center_id needs to be the closest shopping center to the respective property_id and Distance_to_sc is the distance from it. It is to be extracted from shopingcenters.html file.
- .html file is read using pd.read_html.
- The unwanted columns has been dropped using .drop().
- Using get_DistId, the nearest Shopping_center_id and Distance_to_sc has been calculated.

```
[ ]: #file is read
shop_cen_data = pd.read_html("shopingcenters.html")[0].drop(columns=['Unnamed:␣
 ↪0'])
```

```
[ ]: shop_cen_data
```

```
[ ]: #shape of shop_cen_data
shop_cen_data.shape
```

- get_DistId function here takes : df, shop_cen_data, sc_id, lat,lng, where sc_id, lat,lng are the columns of shop_cen_data.

```
[ ]: #passing arguments in get_DistId function
DistId = get_DistId(df, shop_cen_data, 'sc_id', 'lat','lng')
```

```
[ ]: #values calculated are populated in Shopping_center_id and Distance_to_sc
for i, j in df.iterrows():
    df.loc[i,'Shopping_center_id'], df.loc[i,'Distance_to_sc'] = DistId[i][0],␣
 ↪DistId[i][1]
```

### 1.2.5 Extracting Supermarket_id, Distance_to_supermaket from supermarkets.pdf

- Supermarket_id needs to be the closest supermarket to the respective property_id, and Distance_to_supermaket is the distance to it, which is to be extracted from supermarkets.pdf file.
- For finding the number of pages in pdf file provided, PyPDF2 library is imported.
- Then tabula library is imported, for reading the file using read_pdf.
- Using get_DistId, Supermarket_id, Distance_to_supermaket is calculated, which is nearest to the property.

```
[ ]: #file is opened
pdfFileObj = open('supermarkets.pdf', 'rb')
#file is read using PyPDF2 library
pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
#number of pages is read
super_mkt_pages=pdfReader.numPages
```

```
[ ]: #file is again read using tabula library
mkt = read_pdf('supermarkets.pdf', pages = 'all')
#an empty dataframe is created
super_mkt = pd.DataFrame()
#extracting each page in pdf file and appending in created dataframe
for i in range(super_mkt_pages):
    super_mkt = super_mkt.append(mkt[i])
    #index is reset after appending
    super_mkt.reset_index(drop = True, inplace = True)
```

```
[ ]: #unwanted index id dropped
super_mkt = super_mkt.drop(columns=['Unnamed: 0', 'type'])
```

```
[ ]: super_mkt
```

```
[ ]: #shape of super_mkt
super_mkt.shape
```

```
[ ]: #passing arguments in get_DistId function
DistId = get_DistId(df, super_mkt, 'id', 'lat', 'lng')
```

```
[ ]: #values calculated are populated in Supermarket_id and Distance_to_supermarket
for i, j in df.iterrows():
    df.loc[i,'Supermarket_id'], df.loc[i,'Distance_to_supermaket'] =␣
 ↪DistId[i][0], DistId[i][1]
```

### 1.2.6 Extracting suburb from vic_suburb_boundary folder.

- vic_suburb_boundary folder is provided in .zip folder which has been extracted manually.
- This folder contains four files which are required to find the suburb under which a specific property_id falls.

```
[ ]: #shapefile is read
sf = shapefile.Reader("./vic_suburb_boundary/VIC_LOCALITY_POLYGON_shp")
#records of the shapefile
recs = sf.records()
#points of the polygon in the shape file
shapes = sf.shapes()
```

[ ]:
```
#len of shapes and recs in shapefile
len(shapes), len(recs)
```

[ ]:
```
#creating a list of tuples of longitude and latitude of property_id
lg_lt = []
for i in range(len(df)):
    lg_lt.append((df.loc[i,'lng'], df.loc[i,'lat']))
```

[ ]:
```
#an empty list is created to append all the shapes of the suburbs in shapes
suburb_shape_list = []
for i in shapes:
    suburb_shape_list.append(shape(i.__geo_interface__))
```

[ ]:
```
#finding the suburb of each property and populating it to 'suburb' column in df
for i in range(len(lg_lt)):
    for j in range(len(suburb_shape_list)):
        if suburb_shape_list[j].contains(Point(lg_lt[i])):
            df.loc[i,'suburb'] = recs[j][6]
            break
```

### 1.2.7 Calculating Over_ave_price with price and suburb columns.

- Over_ave_price includes boolean values.
- In Over_ave_price, if the value of a property is higher than the average property of its suburb then True needs to added to the column or else False.

[ ]:
```
#a function created to calculate the average values of each suburb
def suburb_average(sub):
    cost = df.loc[df['suburb'] == sub, 'price']
    total = 0
    for i in cost:
        total += i

    return total / len(cost)
```

- A dictionary is created which will store the average value of each suburb in df. This will help in not calculating the average price of each suburb of every row in 'df'.

[ ]:
```
average_dict = {}
for i in range(len(df)):
    if df.loc[i,'suburb'] not in average_dict:
        average_dict[df.loc[i,'suburb']] = suburb_average(df.loc[i,'suburb'])
```

[ ]:
```
#values populated in Over_ave_price column
for i in range(len(df)):
```

```
    df.loc[i,'Over_ave_price'] = df.loc[i,'price'] > average_dict[df.loc[i,␣
 ↪'suburb']]
```

### 1.2.8 Extracting Train_station_id and Distance_to_train_station from 1. GTFS - Melbourne Train Information - From PTV (9 Oct 2015) folder.

- Train_station_id needs to be the closest train station to the respective property_id, and Distance_to_train_station is the distance to it, which is to be extracted from stops.txt file in 1. GTFS - Melbourne Train Information - From PTV (9 Oct 2015) folder.
- The folder provided was in .zip format, whose files have been extracted manually.
- The folder contains eight .txt file.
- Among these eight files, stops.txt file is used to extract the Train_station_id and Distance_to_train_station, which are nearest to property.

[ ]:
```
#a text file is read
stops_data = pd.read_csv(r'./1. GTFS - Melbourne Train Information - From PTV␣
 ↪(9 Oct 2015)/GTFS - Melbourne Train Information/stops.txt')
```

[ ]:
```
#unwanted columns are dropped
stops_data = stops_data.drop(columns = ['stop_short_name', 'stop_name'])
```

[ ]:
```
stops_data
```

[ ]:
```
#shape of stops.txt file
stops_data.shape
```

[ ]:
```
#passing arguments in get_DistId function
DistId = get_DistId(df, stops_data, 'stop_id', 'stop_lat', 'stop_lon')
```

[ ]:
```
#values populated are populated in Train_station_id and␣
 ↪Distance_to_train_station
for i, j in df.iterrows():
    df.loc[i,'Train_station_id'], df.loc[i,'Distance_to_train_station'] =␣
 ↪DistId[i][0], DistId[i][1]
```

### 1.2.9 Extracting travel_min_to_CBD from GTFS - Melbourne Train Information - From PTV (9 Oct 2015) folder.

- travel_min_to_CBD is the time taken from the respective Train_station_id to the 'Flinders Street' on weekdays (Mon-Fri) between 7 am -9am.
- For this column, the average value needs to be calculated for the closest train station to Flinders Street.
- To calculate the value of travel_min_to_CBD, calender.txt, routes.txt, trips.txt, stops.txt, stops_time.txt needs to be used which are inside GTFS - Melbourne Train Information - From PTV (9 Oct 2015) zip folder which has been extracted manually.

[ ]:
```
#files required are read
calender_data = pd.read_csv(r'D:\Jupyter Notebook\Wrangling\Assignment 03\1.␣
↪GTFS - Melbourne Train Information - From PTV (9 Oct 2015)\GTFS - Melbourne␣
↪Train Information\calendar.txt')
routes_data = pd.read_csv(r'D:\Jupyter Notebook\Wrangling\Assignment 03\1. GTFS␣
↪- Melbourne Train Information - From PTV (9 Oct 2015)\GTFS - Melbourne Train␣
↪Information\routes.txt')
trips_data = pd.read_csv(r'D:\Jupyter Notebook\Wrangling\Assignment 03\1. GTFS␣
↪- Melbourne Train Information - From PTV (9 Oct 2015)\GTFS - Melbourne Train␣
↪Information\trips.txt')
stop_times_data = pd.read_csv(r'D:\Jupyter Notebook\Wrangling\Assignment 03\1.␣
↪GTFS - Melbourne Train Information - From PTV (9 Oct 2015)\GTFS - Melbourne␣
↪Train Information\stop_times.txt')
```

- For travel_min_to_CBD column, service_id needs to be found out which run from Monday to Friday.

[ ]:
```
calender_data
```

[ ]:
```
daily_service = calender_data.loc[(calender_data['monday'] == 1) &␣
↪(calender_data['tuesday'] == 1) & (calender_data['wednesday'] == 1) &␣
↪(calender_data['thursday'] == 1) & (calender_data['friday'] == 1),␣
↪'service_id']
```

[ ]:
```
daily_service.values[0]
```

- Now the stop_id fro Flinders Street needs to to find from stops_data.

[ ]:
```
stops_data
```

[ ]:
```
stops_data.shape
```

[ ]:
```
fs_id = stops_data[stops_data["stop_name"].str.contains("Flinders Street")].
↪values[0][0]
```

[ ]:
```
fs_id #stop_id for flinders street
```

- Now I found the route_id from routes_data which run from another station to 'Flinders Station'. Therefore, I have splitted route_long_name with respect to "-" and its [1] index is appended.

[ ]:
```
routes_data
```

[ ]:
```
fs = pd.DataFrame()
for i,j in routes_data.iterrows():
    x = j.route_long_name.split("-")
    if x[1] == ' City (Flinders Street)':
        fs = fs.append(j)
```

[ ]:
```
#it includes table which goes to Flinders Street
fs.shape
```

[ ]:
```
fs
```

- In trips_data, it includes the route_id, service_id which has been extracted above.In this dataframe, direction_id is '0' when it goes towards the Flinders Street and '1' when it come from Flinders Street.
- Thus, I extracted the route_id which goes towards flinders street (extracted above), and service_id = 'T0' (extracted above) and direction_id = '0'.

[ ]:
```
trips_data
```

[ ]:
```
fs_trip = trips_data[(trips_data['service_id'] == 'T0') &␣
↪(trips_data['route_id'].isin(fs.route_id)) & (trips_data['direction_id'] ==␣
↪0)]
```

[ ]:
```
fs_trip.shape #shape of fs_trip
```

[ ]:
```
fs_trip
```

- In stop_times_data, need to extract the trip_id from the condition defined above, departure_time and arrival_time between 7am - 9am.

[ ]:
```
stop_times_data
```

[ ]:
```
fs_stop = stop_times_data[stop_times_data['trip_id'].isin(fs_trip.trip_id) &␣
↪(stop_times_data['departure_time'] >= '07:00:00') &␣
↪(stop_times_data['arrival_time'] <= '09:00:00')]
```

- I have created a column to store the timetaken to reach Flinders Street from a particular Train_station_id, and removed the columns which are of no use right now. Also the index has been resetted.

[ ]:
```
fs_stop['time_taken'] = 0
```

[ ]:
```
fs_stop = fs_stop.drop(columns=["shape_dist_traveled", "stop_headsign",␣
↪"pickup_type", "drop_off_type"])
```

[ ]:
```
fs_stop = fs_stop.reset_index().drop(columns=["index"])
```

- I have calculated the time taken to flinders street in opposite direction. If we go from one station to flinders street or come back to this station, time taken must be same.
- I have started calculating time when flinders street come across to every station above it, and that time has been added in every row as I move from down to above. This method will save the time as its time complexity will be low. Also I have keep the trip-id in check while calculating time. Thats why i have created a boolean.

[ ]:
```
tripId = ""
isFs = False
tformat = "%H:%M:%S"
for i in range(len(fs_stop) - 1, 0, -1):
    #
    if tripId != fs_stop.loc[i, 'trip_id']:
        isFs = False

    if fs_stop.loc[i, 'stop_id'] == 19854:
        isFs = True
        tripId = fs_stop.loc[i, 'trip_id']

    elif isFs:
        s1 = fs_stop.loc[i, "arrival_time"]
        s2 = fs_stop.loc[(i + 1), "arrival_time"]
        sdelta = datetime.strptime(s2, tformat) - datetime.strptime(s1, tformat)
        fs_stop.loc[i, 'time_taken'] = fs_stop.loc[(i + 1), "time_taken"] +␣
↪sdelta.seconds
```

[ ]:
```
#last 20 rows of fs_stop
fs_stop.tail(20)
```

- A function is created to iterate through rows in 'df' to find the Train_station_id of each row, which will take 2 arguments: stop_id of station that are stored in 'df' as Train_station_id and 'df'.

[ ]:
```
def time_taken(stopId, df):
    temp = df[(df["stop_id"] == stopId) & (df["time_taken"] != 0)]
    total = 0
    for i, j in temp.iterrows():
        total += j.time_taken

    if len(temp) != 0:
        return (total / len(temp)) // 60
    else: return 0
```

[ ]:
```
#iterated in df
for i,j in df.iterrows():
     df.loc[i,'travel_min_to_CBD'] = time_taken(df.loc[i, 'Train_station_id'],␣
↪fs_stop)
```

- I have created a .csv for the information extracted above.

```
[ ]: df.to_csv("31901611_A3_solution.csv",  index = False)
```

### 1.3 References


- For defining data integration. [https://www.talend.com/resources/what-is-data-integration/]
- For understanding xml.etree.ElementTree library. [https://docs.python.org/3/library/xml.etree.elementtree.
- For extracting the data of xml file. [https://medium.com/@robertopreste/from-xml-to-pandas-dataframes-9292980b1c1c]
- For renaming the columns. [https://stackoverflow.com/questions/11346283/renaming-columns-in-pandas]
- For understanding and using tabula library. [https://tabula-py.readthedocs.io/en/latest/tabula.html]
- For solving a doubt over extraction of values in file. [https://stackoverflow.com/questions/36684013/extract-column-value-based-on-another-column-pandas-dataframe]
- For mapping table. [https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.contains.html] [https://kanoki.org/2020/01/21/pandas-dataframe-filter-with-multiple-conditions/]
- For opening the shapefile. [https://pythonhosted.org/Python%20Shapefile%20Library/]
- For calculating time difference between two time strings. [https://stackoverflow.com/questions/3096953/how-to-calculate-the-time-interval-between-two-time-strings]