FIT 5196 Assignment 3

- Name: Peiyu Liu
- Student Number: 31153291
- Data: 09/06/2021

Libraries used:

- pandas: handle data, load files, convert files and filter data.use pandas to read data files.
- numpy: handle array and narray data, process matrix oeprations.
- json: load json file. like School.json file.
- BeautifulSoup: pulling data out of HTML and XML files.
- shapefile: provides read and write support for the Shapefile format
- Point: drawing function on an image, take x,y arguments.
- shape: drawing shape on an image for data.
- math: contain mathematical functions for data calculation.
- os: load and handle files.
- datatime: manupulate dates and times to calculate stations schedule time.
- preprocessing: transfer classes to raw feature vector into downstream estimators.
- LinearRegression: use this function to observe target dataset and predict linear.
- pyplot: interactive drawing and procedural drawing.
- scipy: includes modules for optimization, linear algebra, integration, interpolation, and special functions
- seaborn: data visualization library based on matplotlib to generate visualization digram.
- matplotlib: graphical data and provide diversified output formats.
- warnings: ignore warning notification.

Reference:

- Point: https://www.geeksforgeeks.org/wand-point-function-in-python/ (https://www.geeksforgeeks.org/wand-point-function-in-python/) , https://shapely.readthedocs.io/en/stable/manual.html (https://shapely.readthedocs.io/en/stable/manual.html) , https://pypi.org/project/Shapely/ (https://pypi.org/project/Shapely/) , https://automating-gis-processes.github.io/site/notebooks/L1/geometric-objects.html (https://automating-gis-processes.github.io/site/notebooks/L1/geometric-objects.html)
- BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (https://www.crummy.com/software/BeautifulSoup/bs4/doc/)
- shapefile: https://pypi.org/project/pyshp/#overview (https://pypi.org/project/pyshp/#overview) , https://gis.stackexchange.com/questions/113799/how-to-read-a-shapefile-in-python (https://gis.stackexchange.com/questions/113799/how-to-read-a-shapefile-in-python)
- shape: https://pypi.org/project/Shapely/ (https://pypi.org/project/Shapely/)
- math: https://docs.python.org/3/library/math.html (https://docs.python.org/3/library/math.html)
- datatime: https://docs.python.org/3/library/datetime.html (https://docs.python.org/3/library/datetime.html)
- preprocessing: https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
- LinearRegression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-

learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

- pyplot: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html)
- scipy: https://docs.scipy.org/doc/scipy/reference/tutorial/special.html (https://docs.scipy.org/doc/scipy/reference/tutorial/special.html)
- seaborn: https://seaborn.pydata.org/introduction.html (https://seaborn.pydata.org/introduction.html)
- warnings: https://docs.python.org/3/library/warnings.html (https://docs.python.org/3/library/warnings.html)
- matplotlib: https://matplotlib.org/stable/tutorials/introductory/pyplot.html (https://matplotlib.org/stable/tutorials/introductory/pyplot.html)

1. Introduction

- For assignment 3, I need to handle some data files in different file types. The real estate data file is a CSV file, and it contains many data information like property id, latitude, longitude and so on. so I use the pandas' function to read the real estate file. Real estate data also have an XML file, so I open the XML file and use the beautifulsoup function to reformat the file and use the dictionary to store keys and values. After process data in these two files, I use the pandas function to combine two files and drop duplications. Other files like a hospital in xlsx type, recreation file in HTML type, school file in JSON type, boundary files in dbf, prj, shp, shx type. There are also transition files, and files have information about stops, stop times, trips and calendars. after process these files, all data in these files should convert to dataframe, it is a better method to handle data in these files. I concat each file in one new file and generate a linear model to predict the "price" using "Distance_to_school", "travel_min_to_CBD", and "Distance_to_Recreation _centre" attributes.
- important requirement: property id should be unique, travel time should be mean time (all times/ degree), trip should in 7-9.

Import libraries and packages

In [ ]:

```
# import packages and libraries to process data, details will explain in each section.
import pandas as pd
import numpy as np
import json
from bs4 import BeautifulSoup
import shapefile
from shapely.geometry import Point
from shapely.geometry import shape
from math import radians, cos, sin, asin, sqrt, acos
from os import listdir
from datetime import datetime
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot as plt
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt
import warnings

# ignore warning notificaiton when draw diagrams.
warnings.filterwarnings("ignore")
```

read data file and observe variables

In [ ]:

```
# read data file and observe variables information
# read csv file using pandas function
rsd_csv_file = pd.read_csv('31153291/real_estate_data.csv')
rsd_csv_file.head()
```

In [ ]:

```
# delete index
# because I need to concat each file, each file has own index, so better to delete index of each file.
rsd_csv_file = rsd_csv_file.drop(columns = 'index')
```

In [ ]:

```
# open data file
# open xml file and read data
rsd_xml_file = open('31153291/real_estate_data.xml')
rsd__xml_data = rsd_xml_file.read()

# read data and change data group format
# pulling relestate data to lxml format.
# Reference: BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/bs4/doc/
rsd_soup = BeautifulSoup(rsd__xml_data,'lxml')
```

- generate dictionary to store variables' keys and values
- store data like this format: easy to find key to value. {'property_id': [], 'lat': [], 'lng': [], 'addr_street': [], 'price': [], 'property_type': [], 'year': [], 'bedrooms': [], 'bathrooms': [], 'parking_space': []}

In [ ]:

```
rsd_columns = [[] for flag in range(len(rsd_csv_file.columns))]
rsd_dict = dict(zip(rsd_csv_file.columns, rsd_columns))
```

In [ ]:

```
# Search for all type child nodes of the current type to determine whether the filter conditions are met
# Reference: (beautifulsoup find_all(): https://blog.csdn.net/depers15/article/details/51934210)
for flag in rsd_dict.keys():
    type_list = rsd_soup.find(flag).find_all(type)

# Get all types and concatenate them using the given delimiter.
# Referece: (get_text(): https://blog.csdn.net/f156207495/article/details/78074240)
    for flag_ in type_list:
        rsd_dict[flag].append(flag_.get_text())
```

In [ ]:

```
# transfer dictionary to dataframe format
# all files need to be dataframe format. easy to handle
rsd_df = pd.DataFrame(rsd_dict)
```

In [ ]:

```
# use lambda to traversal relestate dataframe to reshape dataframe by filter add and property type.
# Reference: lambda:https://www.guru99.com/python-lambda-function.html
for flag in rsd_df.columns:
    if flag != 'addr_street' and flag != 'property_type':
        # flag should be: property_id,lat,lng,price,year,bedrooms,bathrooms,parking_space
        rsd_df[flag] = rsd_df[flag].apply(lambda flag: float(flag))
```

In [ ]:

```
# combine realestate xml file and html file, delete index. after combine two files then delete copy data.
# Reference: concat: https://jakevdp.github.io/PythonDataScienceHandbook/03.06-concat-and-appen
d.html
# drop_duplications: https://blog.csdn.net/csefrfvdv/article/details/100770965
combine_rsd = pd.concat([rsd_df, rsd_csv_file], ignore_index = True)
combine_rsd = combine_rsd.drop_duplicates()
```

- add lists to new realestate dataframe. lists sizes are equal to index length. uniform each variables numbers.
- Create a list and add a header to the list and the number of rows in each header must be the same.
- Reference: https://www.codenong.com/10712002/ (https://www.codenong.com/10712002/), https://stackoverflow.com/questions/10712002/create-an-empty-list-in-python-with-certain-size (https://stackoverflow.com/questions/10712002/create-an-empty-list-in-python-with-certain-size)

In [ ]:

```
# add lists  to new realestate dataframe. lists sizes are equal to index length. uniform each variables nu
mbers.
# Create a list and add a header to the list and the number of rows in each header must be the same.
# Reference: https://www.codenong.com/10712002/, https://stackoverflow.com/questions/10712002/cr
eate-an-empty-list-in-python-with-certain-size
new_rsd_size = len(combine_rsd.index)
combine_rsd['suburb'] = ['no meaning'] * new_rsd_size
combine_rsd['School_id'] = ['no meaning'] * new_rsd_size
combine_rsd['Distance_to_school'] = ['no meaning'] * new_rsd_size
combine_rsd['Train_station_id'] = ['no meaning'] * new_rsd_size
combine_rsd['Distance_to_train_station'] = ['no meaning'] * new_rsd_size
combine_rsd['travel_min_to_CBD'] = ['no meaning'] * new_rsd_size
combine_rsd['Transfer_flag'] = ['no meaning'] * new_rsd_size
combine_rsd['Hospital_id'] = ['no meaning'] * new_rsd_size
combine_rsd['Distance_to_hospital'] = ['no meaning'] * new_rsd_size
combine_rsd['Recreation_centre_id'] = ['no meaning'] * new_rsd_size
combine_rsd['Distance_to_Recreation_centre'] = ['no meaning'] * new_rsd_size
```

- add all variables to list: 'property_id', 'lat', 'lng', 'addr_street', 'suburb', 'price', 'property_type', 'year', 'bedrooms', 'bathrooms','parking_space', 'School_id', 'Distance_to_school', 'Train_station_id', 'Distance_to_train_station','travel_min_to_CBD', 'Transfer_flag', 'Hospital_id', 'Distance_to_hospital', 'Recreation_centre_id','Distance_to_Recreation_centre'

In [ ]:

```
combine_rsd = combine_rsd[
    ['property_id', 'lat', 'lng', 'addr_street', 'suburb', 'price', 'property_type', 'year', 'bedrooms', 'bathrooms',
     'parking_space', 'School_id', 'Distance_to_school', 'Train_station_id', 'Distance_to_train_station',
     'travel_min_to_CBD', 'Transfer_flag', 'Hospital_id', 'Distance_to_hospital', 'Recreation_centre_id',
     'Distance_to_Recreation_centre']]
```

suburb

- use shapefile reader function to read shapefile file and recrod attribute from file.
- read supplementary_data/vic_suburb_boundary/VIC_LOCALITY_POLYGON_shp.shp
- Reference: read shapefile: https://blog.csdn.net/sgcc_zhs/article/details/75142599 (https://blog.csdn.net/sgcc_zhs/article/details/75142599)
- compare matching latitude and longitude to find correct suburb and add suburb to file.
- using a function called .within() that checks if a point is within.
- Reference: check within point in suburb boundary: https://automating-gis-processes.github.io/CSC18/lessons/L4/point-in-polygon.html (https://automating-gis-processes.github.io/CSC18/lessons/L4/point-in-polygon.html)

In [ ]:

```
combine_rsd.loc[0, ['lat', 'lng']]
```

In [ ]:

```
# use shapefile reader function to read shapefile file and recrod attribute from file.
suburb_file = shapefile.Reader('supplementary_data/vic_suburb_boundary/VIC_LOCALITY_POLYGON_sh
p.shp')
suburb_shape = suburb_file.shapes()
suburb_records = suburb_file.records()
```

In [ ]:

```
# set required parameters
def sub_find(lat, lng, suburb_shape=suburb_shape, suburb_records=suburb_records):
    for flag in range(len(suburb_shape)):
        sub_bound = suburb_shape[flag]
        # use Point within() function to check point in suburb limitation.
        if Point((lng, lat)).within(shape(sub_bound)):
            return suburb_records[flag][-6]
```

In [ ]:

```
# traversal all each index and row in rsd container file to compare suburb latitude and longitude.
for flag_index, flag_row in combine_rsd.iterrows():
    match_suburb = sub_find(flag_row.lat, flag_row.lng)
    # once find matching geographical data, add correct suburb to rsd file.
    combine_rsd.loc[flag_index, 'suburb'] = match_suburb
    #break
```

school data

- read school file,json file can read directly.
- group school latitude and longitude data.
- generate dictionary and zip school information to new format.
- transfer school data to dict format: school id: school geographic data -- {12: array([-37.74, 145.21])}
- use math function to calculate radians distance.
- Calculate the distance between two latitude and longitude
- Longitude and latitude converted to radians
- Use the average radius of the earth
- return result in Three decimal places
- Reference: math distance: https://www.codegrepper.com/code-examples/python/python+calculate+distance+between+two+points (https://www.codegrepper.com/code-examples/python/python+calculate+distance+between+two+points) , https://www.w3resource.com/python-exercises/python-basic-exercise-40.php (https://www.w3resource.com/python-exercises/python-basic-exercise-40.php)

In [ ]:

```
# read json file and show header.
school_file = pd.read_json('31153291/School.json')
school_file.head()
```

In [ ]:

```
# generate dictionary and zip school information to new format.
# transfer school data to dict format: school id: school geographic data -- {12: array([-37.74, 145.21])}
school_dic = dict(zip(school_file.School_ID.to_list(), school_file[['Lat', 'Long']].values))
```

In [ ]:

```
# Calculate the distance between two latitude and longitude
# Reference: math distance: https://www.codegrepper.com/code-examples/python/python+calculate+di
stance+between+two+points , https://www.w3resource.com/python-exercises/python-basic-exercise-
40.php
def calculator(lat1, lon1, lat2, lon2):
    lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    # Longitude and latitude converted to radians
    a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ** 2
    c = 2 * asin(sqrt(a))
    # Use the average radius of the earth
    r = 6378
    # return result in Three decimal places
    return round(c * r, 3)
```

In [ ]:

```
school_names = list(school_dic.keys())
for flag_index, flag_row in combine_rsd.iterrows():
    cal_container = []
    for key in school_dic.keys():
        # key: 0 index is latitude, 1 index is longitude
        cal_dice = calculator(flag_row.lat, flag_row.lng, school_dic[key][0], school_dic[key][1])
        cal_container.append(cal_dice)
        # calculate minimum
    min_dist = min(cal_container)
    school_indx = cal_container.index(min_dist)
    combine_rsd.loc[flag_index, 'School_id'] = school_names[school_indx]
    # add minimum distance to matching school(using school id to pair)
    combine_rsd.loc[flag_index, 'Distance_to_school'] = min_dist
```

In [ ]:

```
combine_rsd.loc[0, ['School_id', 'Distance_to_school']]
```

hospital

- read hospital file: xlsx file should transfer to dataframe then easy to process.
- read xlsx file pages, read pages' name, only one page: Sheet1
- Parameter analysis： parse hospital data.
- reference： Parameter analysis： https://stackoverflow.com/questions/7372716/parsing-excel-documents-with-python (https://stackoverflow.com/questions/7372716/parsing-excel-documents-with-python) , https://yangfangs.github.io/2016/12/13/python-argpaarse-usage/ (https://yangfangs.github.io/2016/12/13/python-argpaarse-usage/)
- generate dictionary and zip hospital data into group: {id: latigude, longitude}.
- use math function to calculate radians distance.
- Calculate the distance between two latitude and longitude
- Longitude and latitude converted to radians
- Use the average radius of the earth
- return result in Three decimal places
- Reference: math distance: https://www.codegrepper.com/code-examples/python/python+calculate+distance+between+two+points (https://www.codegrepper.com/code-examples/python/python+calculate+distance+between+two+points) , https://www.w3resource.com/python-exercises/python-basic-exercise-40.php (https://www.w3resource.com/python-exercises/python-basic-exercise-40.php)

In [ ]:

```
# read xlsx file and read page's name
hospital_xlsx_read = pd.ExcelFile('31153291/hospital.xlsx')
hospital_xlsx_read.sheet_names
```

In [ ]:

```
# Parameter analysis：  https://stackoverflow.com/questions/7372716/parsing-excel-documents-with-python
hospital_data = hospital_xlsx_read.parse('Sheet1')
```

In [ ]:

```
# generate dictionary and zip hospital data into group: {id: [latigude, longitude]}.
hospital_dic = dict(zip(hospital_data.id.to_list(), hospital_data[['lat', 'lng']].values))
```

In [ ]:

```
# same manipuation like school.
# Calculate the distance between two latitude and longitude
# Reference: math distance: https://www.codegrepper.com/code-examples/python/python+calculate+distance+between+two+points
hospital_names = list(hospital_dic.keys())
for flag_index, flag_row in combine_rsd.iterrows():
    cal_container = []
    for key in hospital_dic.keys():
        # key: 0 index is latitude, 1 index is longitude
        cal_dice = calculator(flag_row.lat, flag_row.lng, hospital_dic[key][0], hospital_dic[key][1])
        cal_container.append(cal_dice)
        # calculate minumum distance as hospital distance.
    min_dist = min(cal_container)
    index_hosp = cal_container.index(min_dist)
    combine_rsd.loc[flag_index, 'Hospital_id'] = hospital_names[index_hosp]
      # add minimum distance to matching hospital(using hospital id to pair)
    combine_rsd.loc[flag_index, 'Distance_to_hospital'] = min_dist
```

In [ ]:

```
combine_rsd.loc[0]
```

recreation centre id

- open and read html file
- read data and change data group format
- pulling relestate data to lxml format.
- Reference: BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (https://www.crummy.com/software/BeautifulSoup/bs4/doc/)
- each part format: index 2 is latitude, index 3 is longitude.
- use math function to calculate radians distance.
- Calculate the distance between two latitude and longitude
- Longitude and latitude converted to radians
- Use the average radius of the earth
- return result in Three decimal places
- Reference: math distance: https://www.codegrepper.com/code-examples/python/python+calculate+distance+between+two+points (https://www.codegrepper.com/code-examples/python/python+calculate+distance+between+two+points) , https://www.w3resource.com/python-exercises/python-basic-exercise-40.php (https://www.w3resource.com/python-exercises/python-basic-exercise-40.php)

In [ ]:

```
# recreation_centre_id
rc_html_read = open('31153291/Recreation_centres.html')
rc_data = rc_html_read.read()
```

In [ ]:

```
# read data and change data group format
# pulling relestate data to lxml format.
# Reference: BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/bs4/doc/
rc_format = BeautifulSoup(rc_data, 'lxml')
rc_format
```

In [ ]:

```
# data format is <tr><td>MITCHE3674</td><td>Vermont South Club</td><td>-37.851683</td><td>14
5.180228</td></tr>
rc_format.find_all('td')
```

In [ ]:

```
# index 2 is latitude, index 3 is longitude
rc_format.find_all('td')[2].get_text()
```

In [ ]:

```
rc_info = rc_format.find_all('td')
rc_container = {}
# data format is <tr><td>MITCHE3674</td><td>Vermont South Club</td><td>-37.851683</td><td>14
5.180228</td></tr>
for flag in range(0, len(rc_info), 4):
    rc_id = rc_info[flag].get_text()
    # index 2 is latitude, index 3 is longitude
    rc_lat = rc_info[flag + 2].get_text()
    rc_lng = rc_info[flag + 3].get_text()
    # get latutide and longitude, then transfer them to float format.
    # store float lantitude and longitude in containder.
    rc_container[rc_id] = (float(rc_lat), float(rc_lng))
```

In [ ]:

```
# same function like school and hospital
# Calculate the distance between two latitude and longitude
# Reference: math distance: https://www.codegrepper.com/code-examples/python/python+calculate+di
stance+between+two+points
recreat_names = list(rc_container.keys())
for flag_index, flag_row in combine_rsd.iterrows():
    cal_container = []
    for key in rc_container.keys():
        #  key: 0 index is latitude, 1 index is longitude
        cal_dice = calculator(flag_row.lat, flag_row.lng, rc_container[key][0], rc_container[key][1])
        cal_container.append(cal_dice)
        # calculate minumum distance as hospital distance.
    min_dist = min(cal_container)
    index_rc = cal_container.index(min_dist)
    combine_rsd.loc[flag_index,'Recreation_centre_id'] = recreat_names[index_rc]
     # add minimum distance to matching hospital(using hospital id to pair)
    combine_rsd.loc[flag_index,'Distance_to_Recreation_centre'] = min_dist
```

In [ ]:

```
combine_rsd.loc[0]
```

Train

- use os function to read each folder in gtfs document. #### notice: because my laptop is Mac, so there is empty file: DS_store in my gtfs file, so I use for loop to ignore this empty file.
- read folder file in gtfs, each folder has stops.txt.
- read stops.txt
- traversal each stop stations' name.
- find train stop stations.
- transfer stop station data to dict format: stop name: stop geographic data -- {name: array([latitude, longitude])}
- generate dictionary and zip stop information to new format.
- transfer stop data to dict format: stop name: stop geographic data -- {name: array([-37.74, 145.21])}

In [ ]:

```
# show folders in gtfs folder.
for data_file in listdir('supplementary_data/gtfs/'):
    print(data_file)
```

In [ ]:

```
# use os function to read each folder in gtfs document.
def file_group(filename, path='supplementary_data/gtfs/'):
    file_flag = None
    for file_name in listdir(path):
# because my laptop is Mac, so there is empty file: DS_store in my gtfs file, so I use for loop to ignore t
his empty file.
        if file_name == '.DS_Store':
        # if file name is .DS_Store, ignore this file.
            continue
        file_path = pd.read_csv(path + '/' + file_name + '/google_transit/' + filename, sep=',')
        if file_flag:
            files_temp = pd.concat([files_temp, file_path], ignore_index=True)
            print(files_temp)
        else:
            files_temp = file_path
            file_flag = 1
            # drop duplicated file.
        files_temp = files_temp.drop_duplicates()
        # reset file index.
        files_temp = files_temp.reset_index(drop=True)
    return files_temp
```

In [ ]:

```
# gather stop stations data.
gtfs_stop_file = file_group('stops.txt')
gtfs_stop_file.head()
```

In [ ]:

```
# traversal and find stop name is Southern Cross.
gtfs_stop_file[gtfs_stop_file.stop_name.apply(lambda flag: 'Southern Cross' in flag)]
```

In [ ]:

```
# filter Railway Station/ because I only need train stop stations.
gtfs_stop_file = gtfs_stop_file[gtfs_stop_file.stop_name.apply(lambda flag: 'Railway Station' in flag and
'Railway Station/' not in flag)]
```

In [ ]:

```
# check stop stations' name is unique.
gtfs_stop_file.stop_name.unique()
```

In [ ]:

```
# generate dictionary and zip stop information to new format.
# transfer school data to dict format: school id: school geographic data -- {12: array([-37.74, 145.21])}
stop_container = dict(zip(gtfs_stop_file.stop_id.to_list(),gtfs_stop_file[['stop_lat','stop_lon']].values))
```

In [ ]:

```
# same function like school and hospital
# Calculate the distance between two latitude and longitude
# Reference: math distance: https://www.codegrepper.com/code-examples/python/python+calculate+di
stance+between+two+points
stop_names = list(stop_container.keys())
for flag_index, flag_row in combine_rsd.iterrows():
    cal_container = []
    for key in stop_container.keys():
        #  key: 0 index is latitude, 1 index is longitude
        cal_dice = calculator(flag_row.lat, flag_row.lng, stop_container[key][0], stop_container[key][1])
        cal_container.append(cal_dice)
        # calculate minumum distance as hospital distance.
    min_dist = min(cal_container)
    stop_indx = cal_container.index(min_dist)
    combine_rsd.loc[flag_index, 'Train_station_id'] = stop_names[stop_indx]
     # add minimum distance to matching train station(using station id to pair)
    combine_rsd.loc[flag_index, 'Distance_to_train_station'] = min_dist
```

In [ ]:

```
combine_rsd.loc[0]
```

In [ ]:

```
# travel_min_to_CBD
route_data = file_group('routes.txt')
schedule_stop = file_group('stop_times.txt')
calendar_data = file_group('calendar.txt')
trip_data = file_group('trips.txt')
```

In [ ]:

```
# only need schedule >= 7am and <= 9am
# use 2pm can get all 9am.
schedule_stop = schedule_stop[(schedule_stop.departure_time >= '07:00:00') & (schedule_stop.departur
e_time <= '14:00:00')]
schedule_stop = schedule_stop.reset_index(drop=True)
schedule_stop.head()
```

In [ ]:

```
schedule_stop.tail()
```

In [ ]:

```
trip_data.head()
```

- get array:array(['T0', 'T0_1', 'T0_2', 'T0_3', 'T0_4', 'T0_5', 'T0_6', 'T0_7',

```
      'T0_8', 'T0_9', 'T0', 'T0_1', 'T0_2', 'T0_1', 'T0_10', 'T0_11',
      'T0_12', 'T0_13', 'T0_14', 'T0_15', 'T0_16', 'T0_17', 'T0_18',
      'T0_19', 'T0_20', 'T0_21', 'T0_22', 'T0_23', 'T0_3', 'T0_4',
      'T0_5', 'T0_6', 'T0_7', 'T0_8', 'T0_9', 'T0'], dtype=object)
```

In [ ]:

```
calendar_service = calendar_data[(calendar_data.monday == 1) & (calendar_data.tuesday == 1) & (calend
ar_data.wednesday == 1) & (
          calendar_data.thursday == 1) & (calendar_data.friday == 1)].service_id.values
```

In [ ]:

```
# Reference: isin() https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.isi
n.html
# check the element is included in the value
trip_container = trip_data[trip_data.service_id.isin(calendar_service)].trip_id.values
```

In [ ]:

```
len(trip_container), len(set(trip_container)) # check length is same to ensure data is correct.
```

In [ ]:

```
# find southern cross station
gtfs_stop_file.head()
for flag in gtfs_stop_file.stop_name:
    if 'Southern Cross' in flag:
        print(flag)
```

In [ ]:

```
# get stop stations' id
gtfs_stop_file[gtfs_stop_file.stop_name == 'Southern Cross Railway Station (Melbourne City)'].stop_id.val
ues
```

In [ ]:

```
gtfs_stop_file[gtfs_stop_file.stop_id == 20043]
```

In [ ]:

```
scrsmc = gtfs_stop_file[gtfs_stop_file.stop_name == 'Southern Cross Railway Station (Melbourne City)'].s
top_id.values
scrsmc
```

In [ ]:

```
schedule_stop[schedule_stop.stop_id == 20043].head()
```

In [ ]:

```
schedule_stop[schedule_stop.trip_id == '1022.UJ.1-V48-H-mjp-1.6.R']
```

In [ ]:

```
schedule_stop[schedule_stop.stop_id == 22180].head()
```

In [ ]:

```
schedule_stop[schedule_stop.trip_id == '1.T5.2-WMN-C-mjp-1.1.H']
```

In [ ]:

```
trip_filter = []
# check the element is included in the value
matched_trips = schedule_stop[schedule_stop.stop_id.isin(scrsmc)].trip_id.unique()
for flag in matched_trips:
    if flag in trip_container:
        trip_filter.append(flag)
schedule_stop = schedule_stop[schedule_stop.trip_id.isin(trip_filter)]
trip_filter
```

In [ ]:

```
stations_container = schedule_stop[schedule_stop.trip_id.isin(trip_filter)].stop_id.unique()
stations_df = gtfs_stop_file[gtfs_stop_file.stop_id.isin(stations_container)]

# generate dictionary and zip stop information to new format.
# transfer school data to dict format: stop id: stop geographic data -- {id: array([-37.74, 145.21])}
stations_dict = dict(zip(stations_df.stop_id.tolist(),stations_df[['stop_lat','stop_lon']].values))
stations_dict # all related station
```

In [ ]:

```
# use departure time and arrival time to calculate spend time
def calculate_time(dep_time, arr_time):
    # set time format.
    time_format = '%H:%M:%S'
    time_diff = datetime.strptime(arr_time, time_format) - datetime.strptime(dep_time, time_format)
    # transfer spend time to minutes
    time_min_format = round(time_diff.seconds / 60)
    return time_min_format
```

In [ ]:

```
combine_rsd.Train_station_id.unique()
```

In [ ]:

```
schedule_stop[schedule_stop.stop_id == 20022].trip_id.values
```

In [ ]:

```
tsid_unique = combine_rsd.Train_station_id.unique()

# stop should in 7am-9am
# stop station should be id 20043,22180
def cal_stop(tsid_unique, trip_filter=trip_filter, schedule_stop=schedule_stop):
    st_container = {}
    for id_flag in tsid_unique:
        st_container[id_flag] = []
        matched_trips = schedule_stop[schedule_stop.stop_id == id_flag].trip_id.values
        for trips in matched_trips:
            dep_time = schedule_stop[(schedule_stop.trip_id == trips) & (schedule_stop.stop_id == id_flag)].
departure_time.values[0]
            try:
                arr_time = schedule_stop[(schedule_stop.trip_id == trips) & (schedule_stop.stop_id == 20043
)].arrival_time.values[0]
            except:
                 arr_time = schedule_stop[(schedule_stop.trip_id == trips) & (schedule_stop.stop_id == 22180
)].arrival_time.values[0]
            if dep_time <= arr_time and dep_time <= '09:00:00':
                spend_time = calculate_time(dep_time, arr_time)
                st_container[id_flag].append(spend_time)
        if st_container[id_flag]:
            st_container[id_flag] = np.mean(st_container[id_flag])
        else:
            st_container[id_flag] = 0
    return st_container
```

In [ ]:

```
# check the element is included in the value
combine_rsd[combine_rsd.Train_station_id.isin(scrsmc)]
```

In [ ]:

```
st_container = cal_stop(stations_container, trip_filter=trip_filter)
st_container # including all stations in schedule_stop
```

In [ ]:

```
new_stop_times = {}
for key in st_container.keys():
    if st_container[key]:
        new_stop_times[key] = st_container[key]
st_container = new_stop_times
st_container
```

In [ ]:

```
# flag and cbd
for flag_index, flag_row in combine_rsd.iterrows():
    if flag_row.Train_station_id in st_container.keys():
        combine_rsd.loc[flag_index, 'Transfer_flag'] = 0
        combine_rsd.loc[flag_index, 'travel_min_to_CBD'] = st_container[flag_row.Train_station_id]
    else:
        combine_rsd.loc[flag_index, 'Transfer_flag'] = 1
```

In [ ]:

```
# same function like school and hospital
# Calculate the distance between two latitude and longitude
# Reference: math distance: https://www.codegrepper.com/code-examples/python/python+calculate+di
stance+between+two+points
flag1_transfer = combine_rsd[combine_rsd.Transfer_flag == 1]
for flag_index,flag_row in flag1_transfer.iterrows():
    cal_container = []
    for key in st_container.keys():
        # key index 0 is latitude, 1 is longitude
        cal_dice = calculator(flag_row.lat,flag_row.lng, stop_container[key][0],stop_container[key][1])
        cal_container.append((cal_dice,key))
        # calculate minimum distance
    min_dist = min(cal_container)
    station_new = min_dist[1]
         # add minimum distance to matching travel(using to pair)
    combine_rsd.loc[flag_index,'travel_min_to_CBD'] = st_container[station_new]
```

In [ ]:

```
combine_rsd[combine_rsd.Transfer_flag == 1].travel_min_to_CBD.unique()
```

In [ ]:

```
combine_rsd.loc[0]
```

In [ ]:

```
# change each variables to dataframe foramt and easy to combine each variables into one df file.
combine_rsd.School_id = combine_rsd.School_id.apply(lambda flag: float(flag))
combine_rsd.Distance_to_school = combine_rsd.Distance_to_school.apply(lambda flag: float(flag))
combine_rsd.Distance_to_train_station = combine_rsd.Distance_to_train_station.apply(lambda flag: float(
flag))
combine_rsd.travel_min_to_CBD = combine_rsd.travel_min_to_CBD.apply(lambda flag: float(flag))
combine_rsd.Transfer_flag = combine_rsd.Transfer_flag.apply(lambda flag: float(flag))
combine_rsd.Hospital_id = combine_rsd.Hospital_id.apply(lambda flag: flag)
combine_rsd.Distance_to_hospital = combine_rsd.Distance_to_hospital.apply(lambda flag: float(flag))
combine_rsd.Recreation_centre_id = combine_rsd.Recreation_centre_id.apply(lambda flag: flag)
combine_rsd.Distance_to_Recreation_centre = combine_rsd.Distance_to_Recreation_centre.apply(lambd
a flag: float(flag))
```

In [ ]:

```
# modify property id name
combine_rsd = combine_rsd.rename(columns = {'property_id':'Property_id'})
```

In [ ]:

```
# output result
combine_rsd.to_csv('31153291_A3_solution.csv', index = False)
```

Data Reshaping

- The influence of different normalization/conversion methods (ie normalization, min-max normalization, logarithm, power, box-cox conversion) on the properties of "price", "Distance_to_school", "travel_min_to_CBD" and "Distance_to_Recreation_centre".
- data preprocessing methods: StandardScaler function
- Adjust the distribution of the feature data to the standard normal distribution, also called the Gaussian distribution, which means that the mean dimension of the data is 0 and the variance is 1. After the standard conversion operation is performed on the training data set, the same conversion is applied to the test training set, Use fit function to match this training set.
- reference: https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/ (https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/) , https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)
- Load data set
- Import model parameters
- Training model
- sklearn.linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=1), fit trains on the training set X, y. score returns the predicted coefficient of determination R^2.
- Reference: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html) , https://scikit-learn.org/stable/modules/linear_model.html (https://scikit-learn.org/stable/modules/linear_model.html)

notice: diagram methods' code are reference to lecture and tutorial code.

In [ ]:

```
# Adjust the distribution of the feature data to the standard normal distribution, also called the Gaussian distribution,
# which means that the mean dimension of the data is 0 and the variance is 1.
# After the standard conversion operation is performed on the training data set,
# the same conversion is applied to the test training set, Use fit function to match this training set.
standard_model = preprocessing.StandardScaler().fit(combine_rsd[['price', 'Distance_to_school',
                                         'travel_min_to_CBD',
                                         'Distance_to_Recreation_centre']])
model_format = standard_model.transform(combine_rsd[['price', 'Distance_to_school',
                           'travel_min_to_CBD',
                           'Distance_to_Recreation_centre']])

model_format[0:5]
```

In [ ]:

```
# copy list to forbid modifying raw list.
duplication_df = combine_rsd[['price', 'Distance_to_school',
            'travel_min_to_CBD',
            'Distance_to_Recreation_centre']].copy()
# four variables: price, distance_to_school, travel_min_to_CBD,distance_to_rc.
# index is 0,1,2,3
duplication_df['priced'] = model_format[:, 0]
duplication_df['Distance_to_schooled'] = model_format[:, 1]
duplication_df['travel_min_to_CBDed'] = model_format[:, 2]
duplication_df['Distance_to_Recreation_centred'] = model_format[:, 3]
```

In [ ]:

```
duplication_df['price'].plot(), duplication_df['priced'].plot()
```

In [ ]:

```
parameter_a = duplication_df[['Distance_to_schooled',
            'travel_min_to_CBDed',
            'Distance_to_Recreation_centred']]
parameter_b = duplication_df['priced']
#  Load data set
# Import model parameters
# Training model
# sklearn.linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=1),
# fit trains on the training set X, y.
# score returns the predicted coefficient of determination R^2.
# Reference: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.h
tml , https://scikit-learn.org/stable/modules/linear_model.html

linear_regression = LinearRegression().fit(parameter_a,parameter_b)
linear_regression.score(parameter_a,parameter_b)
```

In [ ]:

```
duplication_df = combine_rsd[['price', 'Distance_to_school',
                                'travel_min_to_CBD',
                                'Distance_to_Recreation_centre']].copy()
```

In [ ]:

```
# fit trains on the training set.
limit_value = preprocessing.MinMaxScaler().fit(duplication_df)
limit_value_format = limit_value.transform(duplication_df)
limit_value_format[0:5]
```

In [ ]:

```
duplication_df = combine_rsd[['price', 'Distance_to_school',
            'travel_min_to_CBD',
            'Distance_to_Recreation_centre']].copy()
duplication_df['priced'] = limit_value_format[:, 0]
duplication_df['Distance_to_schooled'] = limit_value_format[:, 1]
duplication_df['travel_min_to_CBDed'] = limit_value_format[:, 2]
duplication_df['Distance_to_Recreation_centred'] = limit_value_format[:, 3]

# min - max[0-1]
duplication_df[['priced', 'Distance_to_schooled',
            'travel_min_to_CBDed',
            'Distance_to_Recreation_centred']].describe()
```

In [ ]:

```
# plot

%matplotlib inline
```

In [ ]:

```
# scatter reference: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html
# Drawing of scatter plot
# Data mapping to the visual interface
def plot():
    f= plt.figure(figsize = (8,6))
    plt.scatter(model_format[:,0], model_format[:,1], color='red', label='Standardized u=0, s=1 price', alpha
=0.3)
    plt.scatter(model_format[:,1], model_format[:,1], color='green', label='Standardized u=0, s=1 Dist_schoo
l', alpha=0.3)
    plt.scatter(limit_value_format[:,0], limit_value_format[:,1], color='blue', label='min-max scaled [min=0,
 max=1] Dist_school', alpha=0.3)
    plt.title('plot digram')
    plt.xlabel('x')  # Set the label of the x-axis
    plt.ylabel('y') # Set the label of the y-axis
    plt.grid()
    plt.tight_layout()
# Whether there is a numerical or quantitative correlation trend between features, and whether the trend
is linear or non-linear;
# Observe whether there is noise in the data, and intuitively judge whether the noise will have a great im
pact on the model.
# reference: https://blog.csdn.net/weixin_40683253/article/details/87367437
plot()
plt.show()
```

In [ ]:

```
parameter_a = duplication_df[['Distance_to_schooled',
           'travel_min_to_CBDed',
           'Distance_to_Recreation_centred']]
parameter_b = duplication_df['priced']
linear_regression = LinearRegression().fit(parameter_a,parameter_b)
linear_regression.score(parameter_a,parameter_b)
```

In [ ]:

```
#log
duplication_df = combine_rsd[['price', 'Distance_to_school', 'travel_min_to_CBD', 'Distance_to_Recreation
_centre']].copy()
duplication_df['priced'] = np.log2(combine_rsd.price)
duplication_df['Distance_to_schooled'] = np.log2(combine_rsd.Distance_to_school) #
duplication_df['travel_min_to_CBDed'] = np.log2(combine_rsd.travel_min_to_CBD) #
duplication_df['Distance_to_Recreation_centred'] = np.log2(combine_rsd.Distance_to_Recreation_centre)
```

In [ ]:

```
duplication_df.describe()
```

In [ ]:

```
plt.scatter(duplication_df.price, duplication_df.priced)
```

In [ ]:

```
parameter_a = duplication_df[['Distance_to_schooled',
            'travel_min_to_CBDed',
            'Distance_to_Recreation_centred']]
parameter_b = duplication_df['priced']
linear_regression = LinearRegression().fit(parameter_a,parameter_b)
linear_regression.score(parameter_a,parameter_b)
```

In [ ]:

```
#power
duplication_df = combine_rsd[['price', 'Distance_to_school', 'travel_min_to_CBD', 'Distance_to_Recreation
_centre']].copy()
duplication_df['priced'] = combine_rsd['price']**2
duplication_df['Distance_to_schooled'] = combine_rsd.Distance_to_school**2 #
duplication_df['travel_min_to_CBDed'] = combine_rsd.Distance_to_Recreation_centre**2 #
duplication_df['Distance_to_Recreation_centred'] = combine_rsd.travel_min_to_CBD**2 #
# min-max [0-1]
duplication_df.describe()
```

In [ ]:

```
# scatter reference: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html
plt.scatter(duplication_df.price, duplication_df.priced)
```

In [ ]:

```
parameter_a = duplication_df[['Distance_to_schooled',
          'travel_min_to_CBDed',
          'Distance_to_Recreation_centred']]
parameter_b = duplication_df['priced']
#  Load data set
# Import model parameters
# Training model
# sklearn.linear_model.LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=1),
# fit trains on the training set X, y.
# score returns the predicted coefficient of determination R^2.
# Reference: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.h
tml , https://scikit-learn.org/stable/modules/linear_model.html
linear_regression = LinearRegression().fit(parameter_a,parameter_b)
linear_regression.score(parameter_a,parameter_b)
```

In [ ]:

```
# boxcox:
# Reference: scipy.stats.boxcox: http://blog.17baishi.com/7230/ ,
# https://www.geeksforgeeks.org/box-cox-transformation-using-python/, https://docs.scipy.org/doc/
scipy/reference/generated/scipy.stats.boxcox.html
def plot_generation(raw_data):
    original_data = raw_data
    matched_data, matched_lambda = stats.boxcox(original_data)
    fig, ax = plt.subplots(1, 2)
    #  # Use Box-Cox transformation to convert the data to normal, reducing the unobservable error and
 the correlation between the predictor variables. The main feature of Box-Cox transformation is to intro
duce a parameter,
    sns.distplot(original_data, hist=False, kde=True, kde_kws={'shade': True, 'linewidth': 2},
                label="Non-Normal", color="green", ax=ax[0])
    sns.distplot(matched_data, hist=False, kde=True, kde_kws={'shade': True, 'linewidth': 2},
                label="Normal", color="green", ax=ax[1])
    plt.legend(loc = 'upper right')
    fig.set_figheight(5)
    fig.set_figwidth(10)
    # # and estimate the parameter through the data itself to determine the data transformation form tha
t should be adopted.
    print(f"Transformation: {matched_lambda}")
```

In [ ]:

```
duplication_df = combine_rsd[['price', 'Distance_to_school', 'travel_min_to_CBD', 'Distance_to_Recreation
_centre']].copy()
```

In [ ]:

```
for flag in duplication_df.columns:
    plot_generation(duplication_df[flag])
```

Summary

- It took me a lot of energy and time to complete assignment 3. I need to process separate files. The file formats are different. There are Excel, json, html, and shapefiles. I need to convert the data in each file The format is converted into a unified dataframe, and each file has associated corresponding data. Through these data, I can combine each file to form a large file. In the large file, the data in each file can be integrated, and each file can be processed. The data of each file corresponds to the data between the files through the corresponding latitude, longitude, id, and name, and fills them into the new file. By completing this process, my ability to analyze the data and match the data It has been greatly improved. In task2, I used multiple data analysis and drawing methods to visualize my data, allowing me to observe the distribution of my data more intuitively and conveniently. Through the training of assignment 3, I have mastered the visualization methods that are not proficient in teaching.

Reference:

- Point: https://www.geeksforgeeks.org/wand-point-function-in-python/ (https://www.geeksforgeeks.org/wand-point-function-in-python/) , https://shapely.readthedocs.io/en/stable/manual.html (https://shapely.readthedocs.io/en/stable/manual.html) , https://pypi.org/project/Shapely/ (https://pypi.org/project/Shapely/) , https://automating-gis-processes.github.io/site/notebooks/L1/geometric-objects.html (https://automating-gis-processes.github.io/site/notebooks/L1/geometric-objects.html)
- BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (https://www.crummy.com/software/BeautifulSoup/bs4/doc/)
- shapefile: https://pypi.org/project/pyshp/#overview (https://pypi.org/project/pyshp/#overview) , https://gis.stackexchange.com/questions/113799/how-to-read-a-shapefile-in-python (https://gis.stackexchange.com/questions/113799/how-to-read-a-shapefile-in-python)
- shape: https://pypi.org/project/Shapely/ (https://pypi.org/project/Shapely/)
- math: https://docs.python.org/3/library/math.html (https://docs.python.org/3/library/math.html)
- datatime: https://docs.python.org/3/library/datetime.html (https://docs.python.org/3/library/datetime.html)
- preprocessing: https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
- LinearRegression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)
- pyplot: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html)
- scipy: https://docs.scipy.org/doc/scipy/reference/tutorial/special.html (https://docs.scipy.org/doc/scipy/reference/tutorial/special.html)
- seaborn: https://seaborn.pydata.org/introduction.html (https://seaborn.pydata.org/introduction.html)
- warnings: https://docs.python.org/3/library/warnings.html (https://docs.python.org/3/library/warnings.html)
- matplotlib: https://matplotlib.org/stable/tutorials/introductory/pyplot.html (https://matplotlib.org/stable/tutorials/introductory/pyplot.html)
- list: https://www.codenong.com/10712002/ (https://www.codenong.com/10712002/), https://stackoverflow.com/questions/10712002/create-an-empty-list-in-python-with-certain-size (https://stackoverflow.com/questions/10712002/create-an-empty-list-in-python-with-certain-size)
- linearregression： https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html) , https://scikit-learn.org/stable/modules/linear_model.html (https://scikit-learn.org/stable/modules/linear_model.html)
- standardscale: https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/ (https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/) , https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)
- scipy.stats.boxcox: http://blog.17baishi.com/7230/ (http://blog.17baishi.com/7230/) , https://www.geeksforgeeks.org/box-cox-transformation-using-python/ (https://www.geeksforgeeks.org/box-cox-transformation-using-python/), https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.boxcox.html (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.boxcox.html)

- math distance: https://www.codegrepper.com/code-examples/python/python+calculate+distance+between+two+points (https://www.codegrepper.com/code-examples/python/python+calculate+distance+between+two+points)
- Parameter analysis： https://stackoverflow.com/questions/7372716/parsing-excel-documents-with-python (https://stackoverflow.com/questions/7372716/parsing-excel-documents-with-python)