FIT5196 Assessment 3

Student Name: Sarath Gopinathan

Student ID: 30434904

Date: 13/11/2020

Version: 1.4

Environment: Python 3.8.5 and Jupyter notebook

Libraries used:

- pandas to read and perform actions on the given files
- numpy to perform calculations
- math to find distance between 2 lat longs
- zipfile to unzip the zip files
- json to convert json into a list
- xml.etree.ElementTree to read the xml
- tabula to read pdf
- geopandas to get read shp file
- shapely.geometry to create Point
- datetime to get time
- sklearn to perform preprocessing

Table of Contents

Importing packages

Importing all packages required to complete the tasks

In [ ]:

```
# pip install tabula-py
# pip install geopandas
# pip install Shapely==1.2b6


import pandas as pd
import math
import numpy as np
import zipfile
import json
import xml.etree.ElementTree as ET
from tabula import read_pdf
import geopandas as gpd
import shapefile
from shapely.geometry import Point
import datetime
from sklearn import preprocessing
```

Initializing DataFrame

Initializing a new Dataframe with all the column names.

In [ ]:

```
# column names of the dataframe
column_names = ['Property_id', 'lat', 'lng', 'addr_street', 'suburb', 'price', 'propert
y_type', 'year', 'bedrooms',
             'bathrooms', 'parking_space', 'Shopping_center_id', 'Distance_to_sc', 'Tra
in_station_id',
             'Distance_to_train_station', 'travel_min_to_CBD', 'Transfer_flag', 'Hospit
al_id', 'Distance_to_hospital',
             'Supermarket_id', 'Distance_to_supermaket']

# creating dataframe
init_df = pd.DataFrame()

init_df = pd.DataFrame(columns = column_names)

init_df['Distance_to_sc'] = init_df['Distance_to_sc'].astype('float64')
init_df['Distance_to_train_station'] = init_df['Distance_to_train_station'].astype('flo
at64')
init_df['travel_min_to_CBD'] = init_df['travel_min_to_CBD'].astype('float64')
init_df['Distance_to_hospital'] = init_df['Distance_to_hospital'].astype('float64')
init_df['Distance_to_supermaket'] = init_df['Distance_to_supermaket'].astype('float64')
# head of datafram with only column names
init_df.head()
```

Setting default values

Function to set all empty values to default values.

In [ ]:

```
def set_default_values(df):

    def_df = df

    def_df['suburb'] = "not available"
    def_df['Shopping_center_id'] = "not available"
    def_df['Distance_to_sc'] = 0
    def_df['Train_station_id'] = 0
    def_df['Distance_to_train_station'] = 0
    def_df['travel_min_to_CBD'] = 0
    def_df['Transfer_flag'] = -1
    def_df['Hospital_id'] = "not available"
    def_df['Distance_to_hospital'] = 0
    def_df['Supermarket_id'] = "not available"
    def_df['Distance_to_supermaket'] = 0

    return def_df
```

Unzipping the files

Unzipping all the input files.

In [ ]:

```
with zipfile.ZipFile("./30434904.zip", 'r') as zip_ref:
    zip_ref.extractall("./")

with zipfile.ZipFile("./GTFS_Melbourne_Train_Information.zip", 'r') as zip_ref:
    zip_ref.extractall("./")

with zipfile.ZipFile("./vic_suburb_boundary.zip", 'r') as zip_ref:
    zip_ref.extractall("./")
```

Process of combining all the data

1. Read the two realstate files, convert them into dataframes and then merge them and finally merge the new dataframe with a new dataframe with all the columns and set empty column values with default values.
2. Calculate the closest location(hospital / shopping center / supermarket / station) and fill in the id and distance columns respectively.
3. Find the suburb and fill the column.
4. Find travel_min_to_CBD, Transfer_flag column values and fill them.

Reading real_state files

Reading the input files.

Reading real_state.json

Initially the json file is read. Then it is loaded as a list using the json.loads() from the json package and then it is converted into a dataframe using pd.json_normalize() method.

In [ ]:

```
# reading the file
with open("./real_state.json",'r') as infile:
        file = infile.read()

#         file converted to a list
        data = json.loads(file)

# list converted to a dataframe
real_state_json_df = pd.json_normalize(data)

real_state_json_df.head()
```

Reading real_state.xml

Initially the xml file is read. It is identified that there exists a "b'" in the start of the file and a "'" in the end of the file. This is removed to convert the xml to a dataframe. As observed in the xml, the data fall under the root tag and each of the coulmns are created with the their id tag. Thus we read the xml using ET.XML() method from xml.etree.ElementTree package.

Now all the column names are read in a for loop and appended into a list. This is then converted into a dataframe.

We observe that the columns are the rows and the rows are the columns and so we take the transpose of this dataframe.

The index of the dataframe are the tags of each value and so we reset the index. This makes the existing index into a new column called 'index' and we get a new index with numbers starting from 0 which is what we need.

The index column is now deleted as it is of no use.

Finally the column names of the columns are renamed to match real_state_json_df inorder to merge them as they have the exact same columns.

In [ ]:

```
# reading xml file
with open("./real_state.xml",'r') as infile:
        file = infile.read()
# removing "b'"
file = file[2:]

# removing "'"
file = file[:-1]

#reading the xml file
root = ET.XML(file)

all_records = []

# extracting rows and columns as a list
for i, child in enumerate(root):
        record = {}
        for sub_child in child:
            record[sub_child.tag] = sub_child.text
        all_records.append(record)


# converting the list into dataframe
real_state_xml_df = pd.DataFrame(all_records)

# transpose the dataframe
real_state_xml_df = real_state_xml_df.transpose()

# new index with numbers
real_state_xml_df = real_state_xml_df.reset_index()

# removing index column
del real_state_xml_df['index']

dfcols = ['property_id', 'lat', 'lng', 'addr_street',
          'price', 'property_type', 'year', 'bedrooms', 'bathrooms', 'parking_space']

for i in range(0,10):

# renaming all the column names to match real_state_json_df columns
    real_state_xml_df.columns.values[i] = dfcols[i]

# change datatype of lan and lng from string to float
real_state_xml_df['lat'] = pd.to_numeric(real_state_xml_df['lat'])
real_state_xml_df['lng'] = pd.to_numeric(real_state_xml_df['lng'])

real_state_xml_df.head()
```

Merging the two dataframes

real_state_json_df and real_state_xml_df are merged here using concat as both of them have the same columns.

In [ ]:

```
# merging both dataframes to form a single dataframe
final_real_state = pd.concat([real_state_json_df, real_state_xml_df])

final_real_state.head()
```

Merging the new dataframe and the initial dataframe

Merging the init_df and the final_real_state dataframe. Since the property type column names are different, we first peprform outerjoin and then copy values from "property_id" to "Property_id" and then delete "property_id"

Next we call the set_default_values method to replace all the empty values with the default values.

At the end we get phase_one_df

In [ ]:

```
# merging both dataframes using outerjoin
phase_one_df = pd.concat([init_df, final_real_state], axis=0, join='outer', ignore_inde
x=False, keys=None,
          levels=None, names=None, verify_integrity=False, copy=True)

# copy values from column property_id to Property_id
phase_one_df['Property_id'] = phase_one_df['property_id']

# delete column property_id
del phase_one_df['property_id']

# new index with numbers
phase_one_df = phase_one_df.reset_index()

# delete index column
del phase_one_df['index']

# set default values to empty column values
phase_one_df = set_default_values(phase_one_df)

phase_one_df.head()
```

Find closest location(hospital / shopping center / supermarket / station)

Reading the location files and converting them into respective dataframes.

We rename sc_id to id in shopping_center_df to make sure all the dataframes have same column names. We rename and reposition the columns in stops_df to make sure all the four dataframes have same column name and column order.

Reading shopingcenters.pdf

Reading shopingcenters.pdf using the tabula package. In case tabula is not installed, we run the code "pip install tabula-py" to install it. Also, jre is required for this to work and so we must have jre in our environment variables after installing in case it does not exist.

In [ ]:

```
# converts the pdf into a list. Use pages = all to read all the pages
shopping_center_list = read_pdf("./shopingcenters.pdf", pages='all')

# new dataframe created
shopping_center_df = pd.DataFrame();

for i in range(0, len(shopping_center_list)):

# append the values in each page using for loop
    shopping_center_df = shopping_center_df.append(pd.DataFrame(shopping_center_list[i
]))

# sorting the dataframe based on the index values in case they are not sorted(chance of
paging issue)
shopping_center_df.sort_values('Unnamed: 0')

# resetting index and making sure it continues from the end of each page instead of sta
rting from 0 again
shopping_center_df.reset_index(drop=True, inplace=True)

# deleting the index column
del shopping_center_df['Unnamed: 0']

# renameing sc_id to id to make sure all the dataframes have same column names
shopping_center_df.columns.values[0] = "id"

shopping_center_df.head()
```

Reading supermarkets.xlsx

Reading the supermarkets.xlsx using the read_excel() method in pandas library. We then delete the index column(Unnamed: 0) as it is not required.

In [ ]:

```
supermarkets_df = pd.read_excel('./supermarkets.xlsx')

del supermarkets_df['Unnamed: 0']

supermarkets_df.head()
```

Reading hospitals.html

Reading the hospitals.html using the read_html() method in pandas library. A list is created from this and then we convert this into a dataframe. We then delete the index column(Unnamed: 0) as it is not required.

In [ ]:

```
# read file
hospitals_list = pd.read_html('./hospitals.html')

# create new dataframe
hospitals_df = pd.DataFrame();

for i in range(0, len(hospitals_list)):

# append the values in the table using for loop
    hospitals_df = hospitals_df.append(pd.DataFrame(hospitals_list[i]))

# resetting the index with numbers starting from 0
hospitals_df.reset_index(drop=True, inplace=True)

# deleting the index column which is not required.
del hospitals_df['Unnamed: 0']

hospitals_df.head()
```

Reading stops.txt

Reading the stops.txt using the read_csv() method in pandas library.

In [ ]:

```
stops_df = pd.read_csv('./1. GTFS - Melbourne Train Information - From PTV (9 Oct 201
5)/GTFS - Melbourne Train Information/stops.txt', sep=',')

# renaming stop_id to id, stop_lat to lat and stop_lon to lng
stops_df.columns.values[0] = "id"
stops_df.columns.values[3] = "lat"
stops_df.columns.values[4] = "lng"

# repositioning the columns to make sure all the four dataframes have same column name
 and column order
stops_df = stops_df[['id', 'lat', 'lng', 'stop_name', 'stop_short_name']]

stops_df.head()
```

Find closest location(hospital / shopping center / supermarket / station)

Function to calculate and return closest location id and distance based on function call.

In [ ]:

```
# used to calculate and find the closest location(hospital / shopping center / statio
n). Returns the closest location
# and the distance as a dictionary.

def calc_distance(lat1, long1, location_df):

#     empty dictionary created
    closest_location = {}

    for i in range (0, location_df['id'].count()):

        lat2 = float(location_df.iloc[i][1])
        long2 = float(location_df.iloc[i][2])

        # Converts lat & long to spherical coordinates in radians.
        degrees_to_radians = math.pi/180.0

        # phi = 90 - latitude
        phi1 = (90.0 - float(lat1))*degrees_to_radians
        phi2 = (90.0 - float(lat2))*degrees_to_radians

        # theta = longitude
        theta1 = float(long1)*degrees_to_radians
        theta2 = float(long2)*degrees_to_radians

        cos = (math.sin(phi1)*math.sin(phi2)*math.cos(theta1 - theta2) + math.cos(phi1)
*math.cos(phi2))

        #computes distance value using formula
        distance = round(math.acos(cos)*6378,3) #radius of the earth in km

#         converts distance from km to m
        distance = distance * 1000

#       if it is the first value then write it into the dictionary else check if the ex
isting distance is lesser and then
#       write into the dictionary

        if(i == 0):
            closest_location = {"id": location_df.iloc[i][0], "distance": distance}
        else:
            if(closest_location.get("distance") > distance):
                closest_location = {"id": location_df.iloc[i][0], "distance": distance}

    return closest_location
```

In [ ]:

```
for i in range(0, len(phase_one_df)):

#     find closest shopping center
    closest_shopping_center = calc_distance(phase_one_df.iloc[i][1], phase_one_df.iloc[
i][2], shopping_center_df)
    phase_one_df.at[i,"Shopping_center_id"] = closest_shopping_center.get("id")
    phase_one_df.at[i,"Distance_to_sc"] = closest_shopping_center.get("distance")

#     find closest railway stop
    closest_stop = calc_distance(phase_one_df.iloc[i][1], phase_one_df.iloc[i][2], stop
s_df)
    phase_one_df.at[i,"Train_station_id"] = closest_stop.get("id")
    phase_one_df.at[i,"Distance_to_train_station"] = closest_stop.get("distance")

#     find closest hospital
    closest_hospital = calc_distance(phase_one_df.iloc[i][1], phase_one_df.iloc[i][2],
hospitals_df)
    phase_one_df.at[i,"Hospital_id"] = closest_hospital.get("id")
    phase_one_df.at[i,"Distance_to_hospital"] = closest_hospital.get("distance")

#     find closest supermarket
    closest_supermarket = calc_distance(phase_one_df.iloc[i][1], phase_one_df.iloc[i][2
], supermarkets_df)
    phase_one_df.at[i,"Supermarket_id"] = closest_supermarket.get("id")
    phase_one_df.at[i,"Distance_to_supermaket"] = closest_supermarket.get("distance")

phase_two_df = phase_one_df

phase_two_df.head()
```

Find suburb of the location

The suburb of the location is identified using package geopandas and shapely.

In [ ]:

```
# create new dataframe with lat and long values from phase one dataframe
latLong_df = pd.DataFrame(columns = ['lat', 'lng'])

latLong_df['lat'] = phase_two_df['lat']
latLong_df['lng'] = phase_two_df['lng']

# create new geopandas dataframe with the latLong_df
gdf = gpd.GeoDataFrame(latLong_df, geometry=gpd.points_from_xy(latLong_df.lng, latLong_
df.lat))

# read the shape file
areas = gpd.read_file('./VIC_LOCALITY_POLYGON_shp.shp')

# initialise the crs for the new geopandas dataframe
gdf.crs = {'init' : areas.crs}

# replace the crs with the crs of the areas crs
gdf.to_crs(areas.crs, inplace = True)

# join both the dataframes using the operation within. This identifies the name of the
 suburb as both the dataframes merge
name = gpd.sjoin(gdf, areas, how = 'inner', op = 'within')

# write the suburb value to the main dataframe
phase_two_df['suburb'] = name['VIC_LOCA_2']

phase_three_df = phase_two_df

phase_three_df.head()
```

Find average time to flinders station from closest stop and transfer flag

Initially we read all the required txt files. Flinders station id is identified and stored. From calendar we notice that only "T0" service runs from Monday to Friday. Thus we filter out all the stop times that contain "T0" in the trip id.

Next extract all the rows that contain stop id of flinders station stop id. Next, we remove all the values which have departure time less than "07:00:00" and greater than "13:00:00" as that data is useless. Now we merge this dataframe with the stop_times_df as this will give us all the trips which contain flinders station in them. This is used as a base dataframe.

Now we pass each station_id from the phase_three_df to find all the trips that start from the respective stop_id between "07:00:00" and "13:00:00" and find the average time.

If the average time is 0 then we change the Transfer_flag to 1 else we keep it as 0.

In [ ]:

```
# read trips data
trips_df = pd.read_csv('./1. GTFS - Melbourne Train Information - From PTV (9 Oct 201
5)/GTFS - Melbourne Train Information/trips.txt', sep=',')

trips_df
```

In [ ]:

```
# read stop times data
stop_times_df = pd.read_csv('./1. GTFS - Melbourne Train Information - From PTV (9 Oct
 2015)/GTFS - Melbourne Train Information/stop_times.txt', sep=',')

stop_times_df
```

In [ ]:

```
# identify flinders station stop id from  stops dataframe
flinders_df = stops_df[stops_df.isin(["Flinders Street Railway Station"]).any(axis=1)]
flinders_stop_id = str(flinders_df.iloc[0][0])

flinders_stop_id
```

In [ ]:

```
# read calendar data
calendar_df = pd.read_csv('./1. GTFS - Melbourne Train Information - From PTV (9 Oct 20
15)/GTFS - Melbourne Train Information/calendar.txt', sep=',')

calendar_df
```

In [ ]:

```
# Since only T0 runs on all week days, we filter out rows that contain "T0" in the trip
_id
req_stop_times_df = stop_times_df[stop_times_df['trip_id'].str.contains("T0")]

req_stop_times_df
```

In [ ]:

```
# filter out all the rows that have flinders street station id in the stop_id
flinders_stop = req_stop_times_df[req_stop_times_df.stop_id == int(flinders_stop_id)]

# flinders_stop

flinders_stop
```

In [ ]:

```
# new dataframe which will be used to merge late
updated_flinders_stop = pd.DataFrame()

updated_flinders_stop = pd.DataFrame(columns=flinders_stop.columns)

# identify and filter out row if time is less that "07:00:00" am or greater than "13:0
0:00"

for i in range(0, len(flinders_stop)):

    time = flinders_stop.iloc[i][2]

    hours, minutes, seconds = map(int, time.split(':'))

#     getting seconds from datetime.timedelta
    cal_seconds = datetime.timedelta(hours=hours, minutes=minutes, seconds=seconds).sec
onds

#     converting to minutes
    cal_mins = cal_seconds/60

#     480 = "07:00:00" and 780 = "13:00:00" - filter the others out
    if((cal_mins > 420) and (cal_mins < 780)):

        updated_flinders_stop = updated_flinders_stop.append(flinders_stop.iloc[i].to_f
rame().transpose())

# resetting the index with numbers starting from 0
updated_flinders_stop.reset_index(drop=True, inplace=True)

# # deleting the index column which is not required.
# del updated_flinders_stop['index_col']

updated_flinders_stop
```

In [ ]:

```
cols = ['trip_id','arrival_time','departure_time','stop_id','stop_sequence',
        'stop_headsign','pickup_type','drop_off_type','shape_dist_traveled']

new_cols = ['trip_id','arrival_time','departure_time','stop_id','stop_sequence',
            'stop_headsign','pickup_type','drop_off_type','shape_dist_traveled', 'time_
from_flinders']

final_time_diff_df = pd.DataFrame(columns=new_cols)

# for each item in updated_flinders_stop, we get the entire trip from the stop_times_df
by merging both. After merging
# a new clolumn is created where the time difference in minutes is stored(time differen
ce between respective stop
# and flinders stop). If the time difference is negative, it means it is a return trip
 and so we ignore those values.
for i in range (0, len(updated_flinders_stop)):

#     since the iloc.to_frame gives us a transposed dataframe, we take a transpose of i
t
    transposed_df = updated_flinders_stop.iloc[i].to_frame().transpose()

#     merge the two dataframes
    merged_df = pd.merge(stop_times_df, transposed_df,left_on='trip_id', right_on='trip
_id')

#     remove all the y values as they are not required
    merged_df = merged_df.iloc[:, :-8]

#     rename all columns with old column names
    merged_df.columns = cols

#     find the departure time of flinders station from this dataframe
    flinders_stop_time_df = merged_df[merged_df.stop_id == int(flinders_stop_id)]
    flinders_stop_time = str(flinders_stop_time_df.iloc[0][2])

    flinders_stop_time

    hours, minutes, seconds = map(int, flinders_stop_time.split(':'))

    flinders_cal_seconds = datetime.timedelta(hours=hours, minutes=minutes, seconds=sec
onds).seconds

#     calculate the minutes
    flinders_cal_mins = flinders_cal_seconds/60

#     new list created to store all the time difference values(time difference between
 station and flinders station)
    time_diff = []

    for j in range(0, len(merged_df)):

        time = merged_df.iloc[j][2]

        hours, minutes, seconds = map(int, time.split(':'))

        cal_seconds = datetime.timedelta(hours=hours, minutes=minutes, seconds=seconds)
.seconds

        cal_mins = cal_seconds/60
```

```
#         append values to list
        time_diff.append(flinders_cal_mins - cal_mins)

#     add the values to the column
    merged_df['time_from_flinders'] = time_diff

#     remove return trips(if time difference is less than 0 it means that the trip is a
return trip)
    merged_df = merged_df[(merged_df['time_from_flinders'] > 0)]

#     append this dataframe into the final data frame which will be used to return avg
 time based on stop it
    final_time_diff_df = final_time_diff_df.append(merged_df)

final_time_diff_df.head()
```

In [ ]:

```
def avg_time_to_cbd(input_stop_id):

    avg = 0

    count = 0

    total_time = 0

#     subset or required trip
    required_stops_df = final_time_diff_df[final_time_diff_df.stop_id == int(input_stop
_id)]

    for i in range (0, len(required_stops_df)):

#         calculate time
        time = required_stops_df.iloc[i][2]

        hours, minutes, seconds = map(int, time.split(':'))

        cal_seconds = datetime.timedelta(hours=hours, minutes=minutes, seconds=seconds)
.seconds

#     calculate minutes
        cal_mins = cal_seconds/60

#     check if time is greater than or equal to 7 am and less than or equal to 9 am.
# If so add the avg time and add count value by 1
        if((cal_mins >= 420) and (cal_mins <= 540)):

            total_time = total_time + required_stops_df.iloc[i][9]
            count = count+1

    if(count == 0):
        avg = 0
    else:
        avg = total_time/count

#     return the average value
    return avg
```

In [ ]:

```
# write the average time to flinders street and the transfer flag
for i in range(0, len(phase_three_df)):

    avg_time = avg_time_to_cbd(phase_three_df.iloc[i][13])
    phase_three_df.at[i,'travel_min_to_CBD'] = avg_time

#     if average time is 0 it means that there are no direct trains or the stop id is f
linders
#     street stop and so assign flag to 1 else to 0
    if(avg_time == 0):

        phase_three_df.at[i,'Transfer_flag'] = 1

    else:

        phase_three_df.at[i,'Transfer_flag'] = 0

phase_four_df = phase_three_df

#writing to new file
phase_four_df.to_csv(r'30434904_A3_solution.csv', index = False)
```

Data Reshaping

The data reshapping process is done below.

In [ ]:

```
phase_four_df.head()
```

In [ ]:

```
phase_four_df.describe()
```

Z-Score Normalisation (standardisation)

Z-Score normalisation is done below.

In [ ]:

```
std_scale = preprocessing.StandardScaler().fit(phase_four_df[['price', 'Distance_to_sc'
,
                                                              'travel_min_to_CBD', 'Dis
tance_to_hospital']])
df_std = std_scale.transform(phase_four_df[['price', 'Distance_to_sc', 'travel_min_to_C
BD', 'Distance_to_hospital']]) # an array not a df
df_std[0:5]
```

In [ ]:

```
phase_four_df['Pscaled'] = df_std[:,0] # so 'Pscaled' is Price scaled
phase_four_df['DTSCscaled'] = df_std[:,1] # and 'DTSCscaled' is Distance_to_sc scaled
phase_four_df['Tscaled'] = df_std[:,2] # and 'Tscaled' is travel_min_to_CBD scaled
phase_four_df['DTHscaled'] = df_std[:,3] # and 'DTHscaled' is Distance_to_hospital scal
ed
phase_four_df.head()
```

In [ ]:

```
phase_four_df.describe()
```

In [ ]:

```
print('Mean after standardisation:\nprice = {:.2f}, Distance_to_sc = {:.2f}, travel_min
_to_CBD = {:.2f}, Distance_to_hospital = {:.2f}'
      .format(df_std[:,0].mean(), df_std[:,1].mean(), df_std[:,2].mean(), df_std[:,3].m
ean()))
print('\nStandard deviation after standardisation:\nprice = {:.2f}, Distance_to_sc =
{:.2f}, travel_min_to_CBD = {:.2f}, Distance_to_hospital = {:.2f}'
      .format(df_std[:,0].std(), df_std[:,1].std(), df_std[:,2].std(), df_std[:,3].std
()))
```

In [ ]:

```
%matplotlib inline
```

In [ ]:

```
phase_four_df["price"].astype(float).plot(), phase_four_df["Distance_to_sc"].astype(flo
at).plot(),
phase_four_df["travel_min_to_CBD"].astype(float).plot(), phase_four_df["Distance_to_hos
pital"].astype(float).plot()
```

In [ ]:

```
phase_four_df["price"].astype(float).hist(), phase_four_df["Distance_to_sc"].astype(flo
at).hist(),
phase_four_df["travel_min_to_CBD"].astype(float).hist(), phase_four_df["Distance_to_hos
pital"].astype(float).hist()
```

In [ ]:

```
phase_four_df["Pscaled"].plot(), phase_four_df["DTSCscaled"].plot(), phase_four_df["Tsc
aled"].plot(), phase_four_df["DTHscaled"].plot()
```

In [ ]:

```
phase_four_df["Pscaled"].hist(), phase_four_df["DTSCscaled"].hist(), phase_four_df["Tsc
aled"].hist(), phase_four_df["DTHscaled"].hist()
```

In [ ]:

```
phase_four_df["Pscaled"].plot(), phase_four_df["price"].astype(float).plot()
```

In [ ]:

```
phase_four_df["Pscaled"].hist(), phase_four_df["price"].astype(float).hist()
```

In [ ]:

```
phase_four_df["DTSCscaled"].plot(), phase_four_df["Distance_to_sc"].astype(float).plot
()
```

In [ ]:

```
phase_four_df["DTSCscaled"].hist(), phase_four_df["Distance_to_sc"].astype(float).hist
()
```

In [ ]:

```
phase_four_df["Tscaled"].plot(), phase_four_df["travel_min_to_CBD"].astype(float).plot
()
```

In [ ]:

```
phase_four_df["Tscaled"].hist(), phase_four_df["travel_min_to_CBD"].astype(float).hist
()
```

In [ ]:

```
phase_four_df["DTHscaled"].plot(), phase_four_df["Distance_to_hospital"].astype(float).
plot()
```

In [ ]:

```
phase_four_df["DTHscaled"].hist(), phase_four_df["Distance_to_hospital"].astype(float).
hist()
```

It is observed that the shape is more or less the same for all the above plots

MinMax Noramlisation

MinMax normalisation is done below.

Using scikit-learn

Calculated using scikit-learn

In [ ]:

```
minmax_scale = preprocessing.MinMaxScaler().fit(phase_four_df[['price', 'Distance_to_s
c',
                                                              'travel_min_to_CBD', 'Dis
tance_to_hospital']])
df_minmax = minmax_scale.transform(phase_four_df[['price', 'Distance_to_sc',
                                                              'travel_min_to_CBD', 'Dis
tance_to_hospital']])
df_minmax[0:5]
```

Manual

Calculated manually

In [ ]:

```
minP = phase_four_df.price.astype(float).min()
maxP = phase_four_df.price.astype(float).max()
minP, maxP
```

In [ ]:

```
minDTSC = phase_four_df.Distance_to_sc.astype(float).min()
maxDTSC = phase_four_df.Distance_to_sc.astype(float).max()
minDTSC, maxDTSC
```

In [ ]:

```
minT = phase_four_df.travel_min_to_CBD.astype(float).min()
maxT = phase_four_df.travel_min_to_CBD.astype(float).max()
minT, maxT
```

In [ ]:

```
minDTH = phase_four_df.Distance_to_hospital.astype(float).min()
maxDTH = phase_four_df.Distance_to_hospital.astype(float).max()
minDTH, maxDTH
```

In [ ]:

```
p = phase_four_df.price[0]
mmp = (p - minP) / (maxP - minP)
mmp
```

In [ ]:

```
df_minmax[0][0]
```

In [ ]:

```
p = phase_four_df[phase_four_df.price == phase_four_df.price.astype(float).max()].price
mmp = (p - minP) / (maxP - minP)
mmp
```

In [ ]:

```
dtsc = phase_four_df.Distance_to_sc[0]
mmdtsc = (dtsc - minDTSC) / (maxDTSC - minDTSC)
mmdtsc
```

In [ ]:

```
df_minmax[0][1]
```

In [ ]:

```
dtsc = phase_four_df[phase_four_df.Distance_to_sc == phase_four_df.Distance_to_sc.astyp
e(float).max()].Distance_to_sc
mmdtsc = (dtsc - minDTSC) / (maxDTSC - minDTSC)
mmdtsc
```

In [ ]:

```
t = phase_four_df.travel_min_to_CBD[0] # the first value, for practice
mmt = (t - minT) / (maxT - minT)
mmt
```

In [ ]:

```
df_minmax[0][2]
```

In [ ]:

```
t = phase_four_df[phase_four_df.travel_min_to_CBD == phase_four_df.travel_min_to_CBD.as
type(float).max()].travel_min_to_CBD
mmt = (t - minT) / (maxT - minT)
mmt
```

In [ ]:

```
dth = phase_four_df.Distance_to_hospital[0]
mmdth = (dth - minDTH) / (maxDTH - minDTH)
mmdth
```

In [ ]:

```
df_minmax[0][3]
```

In [ ]:

```
dth = phase_four_df[phase_four_df.Distance_to_hospital == phase_four_df.Distance_to_hos
pital.astype(float).max()].Distance_to_hospital
mmdth = (dth - minDTH) / (maxDTH - minDTH)
mmdth
```

In [ ]:

```
print('Mean after standardisation:\nprice = {:.2f}, Distance_to_sc = {:.2f}, travel_min
_to_CBD = {:.2f}, Distance_to_hospital = {:.2f}'
      .format(df_minmax[:,0].min(), df_minmax[:,1].min(), df_minmax[:,2].min(), df_minm
ax[:,3].min()))
print('\nStandard deviation after standardisation:\nprice = {:.2f}, Distance_to_sc =
{:.2f}, travel_min_to_CBD = {:.2f}, Distance_to_hospital = {:.2f}'
      .format(df_minmax[:,0].max(), df_minmax[:,1].max(), df_minmax[:,2].max(), df_minm
ax[:,3].max()))
```

Plot the original, standardised and normalised data values

In [ ]:

```
%matplotlib inline

from matplotlib import pyplot as plt

def plot():
    f = plt.figure(figsize=(8,6))

    plt.scatter(phase_four_df['price'], phase_four_df['Distance_to_sc'],
            color='green', label='input scale', alpha=0.5)

    plt.scatter(df_std[:,0], df_std[:,1], color='red',
             label='Standardized u=0, s=1', alpha=0.3)

    plt.scatter(df_minmax[:,0], df_minmax[:,1],
            color='blue', label='min-max scaled [min=0, max=1]', alpha=0.3)

    plt.title('price and Distance_to_sc content of the realstate dataset')
    plt.xlabel('price')
    plt.ylabel('Distance_to_sc')
    plt.legend(loc='upper right')
    plt.grid()
    plt.tight_layout()


plot()
plt.show()
```

In [ ]:

```
%matplotlib inline

from matplotlib import pyplot as plt

def plot():
    f = plt.figure(figsize=(8,6))

    plt.scatter(phase_four_df['price'], phase_four_df['travel_min_to_CBD'],
            color='green', label='input scale', alpha=0.5)

    plt.scatter(df_std[:,0], df_std[:,2], color='red',
             label='Standardized u=0, s=1', alpha=0.3)

    plt.scatter(df_minmax[:,0], df_minmax[:,2],
            color='blue', label='min-max scaled [min=0, max=1]', alpha=0.3)

    plt.title('price and travel_min_to_CBD content of the realstate dataset')
    plt.xlabel('price')
    plt.ylabel('travel_min_to_CBD')
    plt.legend(loc='upper right')
    plt.grid()
    plt.tight_layout()

plot()
plt.show()
```

In [ ]:

```
%matplotlib inline

from matplotlib import pyplot as plt

def plot():
    f = plt.figure(figsize=(8,6))

    plt.scatter(phase_four_df['price'], phase_four_df['Distance_to_hospital'],
            color='green', label='input scale', alpha=0.5)

    plt.scatter(df_std[:,0], df_std[:,3], color='red',
             label='Standardized u=0, s=1', alpha=0.3)

    plt.scatter(df_minmax[:,0], df_minmax[:,3],
            color='blue', label='min-max scaled [min=0, max=1]', alpha=0.3)

    plt.title('price and Distance_to_hospital content of the realstate dataset')
    plt.xlabel('price')
    plt.ylabel('Distance_to_hospital')
    plt.legend(loc='upper right')
    plt.grid()
    plt.tight_layout()

plot()
plt.show()
```

It is observerved that the shape is not changed in any of the above plots

Data Transformation

The data transformation process is done below(log, power and boxcox).

In [ ]:

```
plt.scatter(phase_four_df['price'], phase_four_df['Distance_to_sc'], phase_four_df['tra
vel_min_to_CBD'],
                phase_four_df['Distance_to_hospital'])
```

Log transformation

Log transformation is done below.

In [ ]:

```
import math
phase_four_df['lp'] = None

phase_four_df["price"] = phase_four_df["price"].astype('float')

for i in range(0, len(phase_four_df["price"])):

    phase_four_df['lp'][i] = math.log(phase_four_df["price"][i])

phase_four_df.head()
```

In [ ]:

```
import math
phase_four_df['ldtsc'] = None

phase_four_df["Distance_to_sc"] = phase_four_df["Distance_to_sc"].astype('float')

for i in range(0, len(phase_four_df["Distance_to_sc"])):

    phase_four_df['ldtsc'][i] = math.log(phase_four_df["Distance_to_sc"][i])

phase_four_df.head()
```

In [ ]:

```
import math
phase_four_df['ldth'] = None

phase_four_df["Distance_to_hospital"] = phase_four_df["Distance_to_hospital"].astype('f
loat')

for i in range(0, len(phase_four_df["Distance_to_hospital"])):

    phase_four_df['ldth'][i] = math.log(phase_four_df["Distance_to_hospital"][i])

phase_four_df.head()
```

In [ ]:

```
plt.scatter(phase_four_df.lp, phase_four_df.ldtsc) # and after
```

In [ ]:

```
plt.scatter(phase_four_df.lp, phase_four_df.ldth) # and after
```

It is observerved that the shape is more or less the same in the above plots.

Power transformation

Power transformation is done below.

In [ ]:

```
import math
phase_four_df['lp'] = None

phase_four_df["price"] = phase_four_df["price"].astype('float')

for i in range(0, len(phase_four_df["price"])):

    phase_four_df['lp'][i] = math.pow(phase_four_df["price"][i],2)

phase_four_df.head()
```

In [ ]:

```
import math
phase_four_df['ldtsc'] = None

phase_four_df["Distance_to_sc"] = phase_four_df["Distance_to_sc"].astype('float')

for i in range(0, len(phase_four_df["Distance_to_sc"])):

    phase_four_df['ldtsc'][i] = math.pow(phase_four_df["Distance_to_sc"][i],2)

phase_four_df.head()
```

In [ ]:

```
import math
phase_four_df['ldth'] = None

phase_four_df["Distance_to_hospital"] = phase_four_df["Distance_to_hospital"].astype('f
loat')

for i in range(0, len(phase_four_df["Distance_to_hospital"])):

    phase_four_df['ldth'][i] = math.pow(phase_four_df["Distance_to_hospital"][i],2)

phase_four_df.head()
```

In [ ]:

```
plt.scatter(phase_four_df.lp, phase_four_df.ldtsc) # and after
```

In [ ]:

```
plt.scatter(phase_four_df.lp, phase_four_df.ldth) # and after
```

It is observerved that the shape is more or less the same in the above plots.

Boxcox transformation

Boxcox transformation is done below.

In [ ]:

```
from scipy import stats
phase_four_df['bcp'] = None

phase_four_df["price"] = phase_four_df["price"].astype('float')

boxcox_data = stats.boxcox(phase_four_df["price"])


for i in range(0, len(boxcox_data[0])):

    phase_four_df['bcp'][i] = boxcox_data[0][i]

phase_four_df.head()
```

In [ ]:

```
from scipy import stats
phase_four_df['bcdtsc'] = None

phase_four_df["Distance_to_sc"] = phase_four_df["Distance_to_sc"].astype('float')

boxcox_data = stats.boxcox(phase_four_df["Distance_to_sc"])


for i in range(0, len(boxcox_data[0])):

    phase_four_df['bcdtsc'][i] = boxcox_data[0][i]

phase_four_df.head()
```

In [ ]:

```
from scipy import stats
phase_four_df['bcdth'] = None

phase_four_df["Distance_to_hospital"] = phase_four_df["Distance_to_hospital"].astype('f
loat')

boxcox_data = stats.boxcox(phase_four_df["Distance_to_hospital"])


for i in range(0, len(boxcox_data[0])):

    phase_four_df['bcdth'][i] = boxcox_data[0][i]

phase_four_df.head()
```

In [ ]:

```
plt.scatter(phase_four_df.bcp, phase_four_df.bcdtsc) # and after
```

In [ ]:

```
plt.scatter(phase_four_df.bcp, phase_four_df.bcdth) # and after
```

It is observerved that the shape is more or less the same in the above plots.

References

1. scipy.stats.boxcox — SciPy v1.5.4 Reference Guide. (2020). Retrieved 18 November 2020, from https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.boxcox.html (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.boxcox.html)

In [ ]:

```

```