FIT5196 Assessment 3

Student Name: DISHI JAIN

Student ID: 30759307

Libraries used:

- from **future** import division, unicode_literals
- import codecs
- from bs4 import BeautifulSoup
- import pandas as pd
- import datetime
- import json

- from math import sin, cos, sqrt, atan2, radians
- import tabula #!pip install tabula-py #conda install -c conda-forge tabula-py
- import xmltodict
- from tabula import read_pdf

- import numpy
- import matplotlib %matplotlib inline
- import shapefile
- import matplotlib.pyplot as plt
- import matplotlib.patches as patches
- from matplotlib.patches import Polygon
- from matplotlib.collections import PatchCollection
- import shapely
- from shapely.geometry import Point

1. Introduction

In this assignment we are required to complete two tasks. Here we are required to extract data from different file formats like pdf,xml,,etc. Hence handling of such files are needed in tis assignment. After this handling we are also required to perform certain mathematical and geographical calculations to determine nearest stops to a property. These stops could be shoppig centers, train stations, etc. In this assignment we are also required to reshape the data and perform certain normalisatons and transformtions on the data to prepare it in a manner such that it can fit into a Linear model more accurately.

2. Importing Libraries

In [ ]:

```
from __future__ import division, unicode_literals
import codecs
from bs4 import BeautifulSoup
import pandas as pd
import datetime
import json


from math import sin, cos, sqrt, atan2, radians
import tabula
#!pip install tabula-py
# conda install -c conda-forge tabula-py
import xmltodict
from tabula import read_pdf


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

3. Task - 1

4. Reading Files given to us

In [ ]:
```
##HOSPTALS.html

#creating html object
object_hos = codecs.open("hospitals.html", 'r', 'utf-8')
bsobj = BeautifulSoup(object_hos, "lxml")

#extrcating the dataframe class
table_hos = bsobj.find("table", attrs={"class": "dataframe"})
column_name_hospital = []

#extractng the column names
for value in table_hos.find_all("thead"):
    column_name_hospital.append(value.text)
column_name_hospital = column_name_hospital[0].replace("\n"," ").split()

table_data_hos = table_hos.tbody.find_all("tr")

#getting the values for each column and each row
final_hos_values = []
for td in table_data_hos:
    rows = []
    for x in td.find_all("td"):
        rows.append(x.text)
    final_hos_values.append(rows)

#creating a dataframe with extracted values nd column name
df_hospitals = pd.DataFrame(final_hos_values, columns = column_name_hospital)
df_hospitals
```

In [ ]:

In [ ]:
```
##REAL_STATE.JSON

#reading the .json file
with open('real_state.json') as object_json:
    real_state_values = json.load(object_json)

#converting the data to a dataframe
df_real_state_json = pd.DataFrame(real_state_values)
df_real_state_json
```

In [ ]:
```
##REAL_STATE.xml

#creating object for xml file
object_real_state_xml = open("real_state.xml" , "r")
real_state = object_real_state_xml.read()

#slicing the string to achieve parsing of data
real_state = real_state[2:-1]

#creating an xml to dict object
dictionary_real_state_xml = xmltodict.parse(real_state)

#column names extraction
colname_real_state_xml = list(dictionary_real_state_xml['root'].keys())
list_of_col_vals = []

#extracting the values for each column using #text parameter
for colname in colname_real_state_xml:
    individual_col_vals = []
    for i in dictionary_real_state_xml['root'][colname]:
        if dictionary_real_state_xml['root'][colname][i] != 'dict':
            individual_col_vals.append(dictionary_real_state_xml['root'][colname]
    list_of_col_vals.append(individual_col_vals)


#creating a dataframe and transposing to get the desired shape
df_real_state_xml = pd.DataFrame(list_of_col_vals)
df_real_state_xml = df_real_state_xml.transpose()

#giving the column name values
df_real_state_xml.columns = colname_real_state_xml
df_real_state_xml
```

In [ ]:

In [ ]:
```
##SHOPINGCENTERS.PDF

#reading pdf file with all pages
shopping_listdfs = read_pdf("shopingcenters.pdf",pages='all')

#concatenating all dataframes in the list created
df_shopping = pd.concat(shopping_listdfs)

#removing Unnamed columns
df_shopping = df_shopping.loc[:, ~df_shopping.columns.str.match('Unnamed')]
df_shopping.reset_index(drop=True,inplace=True)
df_shopping
```

In [ ]:

In [ ]:
```
## 5

##SUPERMARKETS.XLSX

#reading .xlsx file with sheet as 1
df_supermarket = pd.read_excel('supermarkets.xlsx', sheet_name='Sheet1')

#removing the unnamed columns
df_supermarket = df_supermarket.loc[:, ~df_supermarket.columns.str.match('Unnamed
df_supermarket.reset_index(drop=True,inplace=True)
df_supermarket
```

Finding repetitive ids in each dataframes created above.

In [ ]:
```
df_hospitals.id.value_counts()
```

In [ ]:
```
df_supermarket.id.value_counts()
```

In [ ]:
```
df_shopping.sc_id.value_counts()
```

In [ ]:
```
df_real_state_json.property_id.value_counts()
```

In [ ]:
```
df_real_state_xml.property_id.value_counts()
```

Hence all dataframes have unique ids except for df_real_state_json. Checking the duplicate values and removing them below

In [ ]:
```
df_real_state_json[df_real_state_json.duplicated()]
```

Hence only df_real_state_json file has repetitive property_ids. To remove them we can use drop duplicate()

In [ ]:
```
df_real_state_json.drop_duplicates(inplace=True)
```

In [ ]:
```
df_real_state_json.property_id.value_counts()
```

In [ ]:
```
df_real_state_json[df_real_state_json.duplicated()]
```

Now no duplicate values exist in the dataframe

In [ ]:
```

```

In [ ]:
```
## FINAL DATA FRAME ALL COMBINED

#creating the real_state_dataframe with concatenation
real_state_combined = pd.concat([df_real_state_xml,df_real_state_json]).drop_dupl
real_state_combined
```

In [ ]:
```
real_state_combined.property_id.value_counts()
```

Hence only unique ids exists in the dataframe now

Now to calculate the minimum distance of each property to the nearest shopping center, train station, supermarket and hospital we use the below function. We can also use the function to get the ids of each shopping center, train station, supermarket and hospital.

In [ ]:
```
#Function to find minimum distance and nearest id to given property


def distance_cal(lat1,lon1,df_new):
    final_dictionary_dis = {}

    #get column names
    column_names = list(df_new.columns)
    id_val = ''.join([s for s in column_names if "id" in s])
    lat_val = ''.join([s for s in column_names if "lat" in s])
    lng_val = ''.join([s for s in column_names if "lng" in s or "lon" in s])

    # given radius of earth in km
    R = 6378.0

    #converting lat long values to radian
    lat1 = radians(float(lat1))
    lon1 = radians(float(lon1))
    for index,i in enumerate(zip(df_new[lat_val],df_new[lng_val])):

        #converting lat long values to radian
        lat2 = radians(float(i[0]))
        lon2 = radians(float(i[1]))
        dlon = lon2 - lon1
        dlat = lat2 - lat1

        a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
        c = 2 * atan2(sqrt(a), sqrt(1 - a))

        #variable to store distance from given point to propert lat long
        distance = R * c

        #dictionary to store distance and ids
        final_dictionary_dis.update({distance:df_new[id_val][index]})

    #finding minimum distance and respective ids
    min_distance = min(final_dictionary_dis)
    min_hos_id = final_dictionary_dis[min_distance]
    return min_hos_id,min_distance


# def euclidean_dist_cal(x1,y1,x2,y2):
#     return(sqrt((float(x2)-float(x1))**2 + (float(y2)-float(y1))**2))
```

In [ ]:
```
#reading stops.txt file to get train stops

df_train_stations = pd.read_csv('stops.txt', sep=",")
df_train_stations.head()
```

In [ ]:
```
```

In [ ]:
```
#setting default values for the column that need to be created

real_state_combined['Hospital_id'] = 'not_available'
real_state_combined['Distance_to_hospital'] = 0
real_state_combined['Shopping_center_id'] = 'not_available'
real_state_combined['Distance_to_sc'] = 0
real_state_combined['Supermarket_id'] = 'not_available'
real_state_combined['Distance_to_supermaket'] = 0
real_state_combined['Train_station_id'] = 0
real_state_combined['Distance_to_train_station'] = 0
```

In [ ]:
```
#calling the function to get minimum distance and respective ids and filling the

real_state_combined['Hospital_id'],real_state_combined['Distance_to_hospital'] =
real_state_combined['Shopping_center_id'],real_state_combined['Distance_to_sc'] =
real_state_combined['Supermarket_id'],real_state_combined['Distance_to_supermaket
real_state_combined['Train_station_id'],real_state_combined['Distance_to_train_st
```

In [ ]:

In [ ]:

In [ ]:
```
#reading the shape files

sf = shapefile.Reader("./VIC_LOCALITY_POLYGON_shp") # note, no suffix, all 3 file
recs = sf.records()
shapes = sf.shapes()
```

In [ ]:
```
#fnding the length of recs and shapes
len(recs), len(shapes)
```

In [ ]:
```
#assigning default value to Suburb column

real_state_combined['Suburb'] = 'not available'
```

In [ ]:
```
cm = matplotlib.cm.get_cmap('Dark2')

Nshp = len(shapes)
cccol = cm(1.*numpy.arange(Nshp)/Nshp) # one colour for every contry...

list_of_poly = []
for nshp in range(Nshp):
    ptchs   = []
    pts     = numpy.array(shapes[nshp].points)
    prt     = shapes[nshp].parts
    par     = list(prt) + [pts.shape[0]]


    for pij in range(len(prt)):
        #polygons appended to a list for each suburb
         ptchs.append(Polygon(pts[par[pij]:par[pij+1]]))
    #list of polygons created where each element is a list
    list_of_poly.append(ptchs)

#finding the suburb for each property id
for i,j in zip(real_state_combined['lng'],real_state_combined['lat']):
    for x in range(len(list_of_poly)):

        #finding the suburb for each point and appending to column Suburb
        if list_of_poly[x][0].contains(Point(float(i),float(j)))[0] == True:
            real_state_combined.at[real_state_combined.index[(real_state_combine
```

In [ ]:

In [ ]:
```
#reading txt files for further processing

df_agencey = pd.read_csv('agency.txt', sep=",")
df_calendar = pd.read_csv('calendar.txt', sep=",")
df_calendar_dates = pd.read_csv('calendar_dates.txt', sep=",")
df_routes = pd.read_csv('routes.txt', sep=",")
df_shapes = pd.read_csv('shapes.txt', sep=",")
df_stop_times = pd.read_csv('stop_times.txt', sep=",")
df_trips = pd.read_csv('trips.txt', sep=",")
```

Checking for duplicate values in the files

In [ ]:
```
df_agencey[df_agencey.duplicated()]
```

In [ ]:
```
df_agencey.agency_id.value_counts()
```

In [ ]:
```
df_calendar[df_calendar.duplicated()]
```

In [ ]:
```
df_calendar.service_id.value_counts()
```

In [ ]:
```
df_calendar_dates[df_calendar_dates.duplicated()]
```

In [ ]:
```
df_calendar
```

In [ ]:
```
df_routes[df_routes.duplicated()]
```

In [ ]:
```
df_shapes[df_shapes.duplicated()]
```

In [ ]:
```
df_stop_times[df_stop_times.duplicated()]
```

In [ ]:
```
df_trips[df_trips.duplicated()]
```

Creating a joined df of the df_stop_times and df_trips dataframes on the trip_id column

In [ ]:
```
joined_df = df_stop_times.join(df_trips.set_index('trip_id'), on = 'trip_id')
```

Extracting the service id which tells us the service that are functioning on Monday,Tuesday,Wednesday,Thrusday, and Friday but not on Saturday and Sunday

In [ ]:
```
#extracting the relevant service id

service_id_valid = df_calendar['service_id'][(df_calendar['monday']==1) & (df_cal
```

In [ ]:
```
#filterng the joined_df
joined_df = joined_df[joined_df['service_id'].isin(service_id_valid)]
joined_df
```

In [ ]:
```
#extracting the valid trips where departure time isin '07' , '08' , '09'

correct_trips = joined_df['trip_id'][joined_df.departure_time.str[0:2].isin(['07'
joined_df = joined_df[joined_df.trip_id.isin(correct_trips)]
joined_df
```

In [ ]:
```
#checking for head sign to get trips going towards flinders street

joined_df = joined_df[joined_df['trip_headsign'] == 'City (Flinders Street)']
joined_df
```

In [ ]:
```

```

In [ ]:
```
#extracting the flinders id train station

flinders_id = df_train_stations['stop_id'][df_train_stations['stop_name'] == 'Fli
flinders_id = flinders_id[0]
flinders_id
```

In [ ]:

```

```

In [ ]:

```
joined_df.reset_index(inplace=True,drop=True)
joined_df
```

In [ ]:

```
#getting arrival and departure times and converting to datetime format

arrival_t = '07:00:00'
depart_t = '09:00:00'
arrival_dt = datetime.datetime.strptime(arrival_t, '%H:%M:%S')
depart_dt = datetime.datetime.strptime(depart_t, '%H:%M:%S')
```

In [ ]:
```
#function to determine the column travel_min_to_CBD

def find_travel_min_CBD(i):

    #creating df with stop ids only in flinder and id received in argument
    new_df = joined_df[(joined_df['stop_id'] == i) | (joined_df['stop_id'] == fli
    new_df.reset_index(inplace=True,drop=True)
    correct_trips = []
    #getting valid ids between 7 am to 9 am departure time for the stop id i  onl
    for val in new_df.index:
        if (new_df.loc[val,'stop_id'] == i) & (datetime.datetime.strptime(new_df.
            correct_trips.append(new_df.loc[val,'trip_id'])

    #correct_trips = new_df.trip_id.value_counts().index.to_list()
    new_valid_trips = []
    #creating subset df for each valid trip id
    for j in correct_trips:
        subset_df = new_df[new_df.trip_id == j]
        # each subset df has only two rows one for i and one for flinder id
        if len(subset_df) == 2:
            new_valid_trips.append(j)

    #final df with valid trips
    final_df = new_df[new_df['trip_id'].isin(new_valid_trips)]
    final_df.reset_index(inplace=True,drop=True)
    #finding the travel time
    final_time_list = []
    if len(new_valid_trips)>0:
        for new_trip in new_valid_trips:
            final_subset_df = final_df[final_df['trip_id'] == new_trip]
            if len(final_subset_df) > 0 :
                #departure time of 2 rows
                departure_time = final_subset_df['departure_time'].to_list()
                #arrival time of 2 rows
                arrival_time = final_subset_df['arrival_time'].to_list()
                #departure time of first row hence from stop i
                departure_time_val = departure_time[0]
                #arrival time of second row hence for flinders
                arrival_time_val = arrival_time[1]

                #conversion to datetime format
                d_time = datetime.datetime.strptime(departure_time_val, '%H:%M:%S
                a_time = datetime.datetime.strptime(arrival_time_val, '%H:%M:%S')
                final_time_list.append((a_time - d_time).seconds/60)

            return round(sum(final_time_list)/len(final_time_list)),0
    else:
        return 0,1

# new_valid_trips
# denominator = len(new_valid_trips)
```

In [ ]:

In [ ]:
```
#default value for flag
real_state_combined['Transfer_flag'] = -1
real_state_combined['travel_min_to_CBD'] = 0

#function to get respective flag and minute values
real_state_combined['travel_min_to_CBD'],real_state_combined['Transfer_flag'] = z
```

In [ ]:
```
#checking if we have duplicate records
real_state_combined.property_id.value_counts()
```

In [ ]:
```
final_write_output = real_state_combined.copy()
final_write_output.dtypes
```

In [ ]:

In [ ]:
```
#converting the column names as required in ouput file
final_write_output['suburb'] = final_write_output['Suburb']
final_write_output['Distance_to_supermarket'] = final_write_output['Distance_to_s
```

In [ ]:
```
#column ordering
columns_names = ['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',
 'Transfer_flag',
 'Hospital_id',
 'Distance_to_hospital',
 'Supermarket_id',
 'Distance_to_supermarket']
```

In [ ]:
```
final_write_output = final_write_output[columns_names]
```

In [ ]:
```
final_write_output.dtypes
```

In [ ]:

In [ ]:
```
#rounding off the values to 3 decimal places

final_write_output['Distance_to_sc'] = final_write_output['Distance_to_sc'].apply
final_write_output['Distance_to_train_station'] = final_write_output['Distance_to
final_write_output['Distance_to_hospital'] = final_write_output['Distance_to_hosp
final_write_output['Distance_to_supermarket'] = final_write_output['Distance_to_s
```

In [ ]:

In [ ]:
```
#writing dataframe to csv file
final_write_output.to_csv('30759307_A3_solution.csv',index=False)
```

5. TASK 2

In this task we are asked to judge the effects of different normalization(or scaling) and transformation techniques on selected columns that are "Distance_to_sc" , "Distance_to_hospital" , "travel_min_to_CBD" and "price". The transformation and normalization techniques are applied in a manner assuming that we need to create a Linear model that will be used to predict the prices based on the other three columns.

For Transformation -

The two assumptions that we are focused upon are Linearity and Normality.

Linearity assumes that the relationship between the traget variable and the predictors is linear. That is if one increases then the other also increases and if one decreases then the other also decreases which is referred to as posiive correlation. And if one increases and other decreases or vice versa then it is called as negative correlation.

Normality means that the predictors and the target variable's data is normally distributed. This helps in better predictions of the data when the model is created. A normally ditributed column will be a better input for the model and hence will make better predictions.

We are required to apply different transformation techniques on all the 4 columns. These transformation techniques are applied to columns to remove any kind of skewness in the data. When the data in a column is right or left skewed then the model will be trained on more occurences of a particular value. Hence it would be like training the entire model on imbalanced data. Hence to remove this skewness we apply the different well known transformation techniques like Square Root, Logarithm, Reciprocal, Power,etc. These transformations will hence help us to achieve better linearity and normality of the columns.

For Normalisation/Scaling -

Scaling or normalizing technique means to scale the values of a column in a particular range. If our data contains values which are varying highly in range then we apply standardisation techniques. After applying the techniques in Scaling, the shape of the graph/distribution of data doesn't change as we are only changing the data by a range. Hence we standardise the values in all the 4 columns individually so that all the columns receive an equal weightage while creating the model.

The 2 well known techniques for Normalisation/Scaling are -

MinMax Scaling and ZScore Scaling

In MinMax Scaling the column's data is converted into a range such that the min value of that column is 0 and the maximum value of that column is 1. Each value is extrcated by subtracting it from the minimum value of the column and diving by the range. In this approach the outliers are not handled and they still remain in the data. For ZScore Scaling we use StandardScaler which standardies a column by subtracting the mean and scaling to unit variance. Unit variance is basically derived by dividing all the values by the SD(standard deviation). After this approach the mean of the column becomes 0 and the SD becomes 1. In this approach the outliers are hndled while creatng the model.

Now to check the Linearity and Normality for the given columns, we will plot scatter plots and histograms.

5.1 CHECKING LINEARITY

In [ ]:
```
pd.options.mode.chained_assignment = None

#extracting required column only
cols = ['Distance_to_sc','travel_min_to_CBD','Distance_to_hospital','price']

#creating dataframe
df_reshape = real_state_combined[cols]
df_reshape
```

In [ ]:
```

df_reshape['price'] = df_reshape['price'].astype(float)
```

In [ ]:
```
#finding correlation amongst the columns
df_reshape.corr()
```

In [ ]:
```
import seaborn as sns
# scatter plot with regression line
sns.regplot(df_reshape["price"], df_reshape["Distance_to_sc"])
```

In [ ]:
```
import seaborn as sns
# scatter plot with regression line
sns.regplot(df_reshape["price"], df_reshape["Distance_to_hospital"])
```

In [ ]:
```
import seaborn as sns
# scatter plot with regression line
sns.regplot(df_reshape["price"], df_reshape["travel_min_to_CBD"])
```

From above graphs we get to see that the three columns have a negative correlation with the target column "price". This can be analysed from the regression line plotted in each graphs which tells us that while increasing the value of the predictor the value of price decreases. We can also see that the data points in the graphs are not evenly distributed across the regression line which causes an issue in the data. This would also cause problems while predicting new prices from the Linear Model.

Now to check the Normality of all 4 columns we plot a histogram

5.2 CHECKING NORMALITY

In [ ]:
```
df_reshape["Distance_to_hospital"].hist(bins=30)
```

In [ ]:
```
df_reshape["Distance_to_sc"].hist(bins=30)
```

In [ ]:
```
df_reshape["travel_min_to_CBD"].hist(bins=30)
```

In [ ]:
```
df_reshape["price"].hist(bins=30)
```

The above histograms tells us about the skewness of the data. If the tail is towards the right then the column is right skewed. If the tail iis towards the left then the column is left skewed. And if the histogram forms a bell shape then we can say that the data is not skewed.

From the above graohs we can see that Distance_to_hospital, Distance_to_sc and price are right skewed while the travel_to_min_CBD is unevenly skewed.

Now to remove this skewness, we apply ceratin transformation techniques on the respective columns.

We will apply the Power(or Square Power) Transformation on the columns -

Distance_to_hospital

Distance_to_sc

price

This is beacuse when the data is right skewed we apply transformations such that the power is in decimals like square roots and less than 1. This is because it allows the data to be extracted such that the skeness decreases and hence the data becomes more normally distributed.

For the column travel_min_to_CBD we achieve a left skewed distribution. For such distributions we can apply either log or power trnsformations where power could be a number greater than or close to 1. This is because it allows the data to be extracted such that the skeness decreases and hence the data becomes more normally distributed.

Hence below we apply power transformations like 1/4,1/2,1/5 on the 3 columns and apply power transformtion like 1.2 on the travel_to_min_CBD column. This allows the skewness to decrease and hene the data becomes more normally distributed.

In [ ]:
```
import math

#transforming column values
df_reshape['new_Distance_to_hospital'] = None
i = 0
for row in df_reshape.iterrows():
    df_reshape['new_Distance_to_hospital'].at[i] = math.pow(df_reshape['Distance_
    i += 1

df_reshape['new_Distance_to_sc'] = None
i = 0
for row in df_reshape.iterrows():
    df_reshape['new_Distance_to_sc'].at[i] = math.pow(df_reshape['Distance_to_sc'
    i += 1

df_reshape['new_travel_min_to_CBD'] = None
j = 0
for row in df_reshape.iterrows():
    df_reshape['new_travel_min_to_CBD'].at[j] = math.pow(df_reshape['travel_min_t
    j += 1



df_reshape['new_price'] = None
j = 0
for row in df_reshape.iterrows():
    df_reshape['new_price'].at[j] = math.pow(df_reshape['price'][j],1/5)
    j += 1
```

In [ ]:

In [ ]:
```
#Converting each column type to float below
df_reshape['new_Distance_to_hospital'] = df_reshape['new_Distance_to_hospital'].a
df_reshape['new_Distance_to_sc'] = df_reshape['new_Distance_to_sc'].astype(float)
df_reshape['new_travel_min_to_CBD'] = df_reshape['new_travel_min_to_CBD'].astype(
df_reshape['new_price'] = df_reshape['new_price'].astype(float)
```

Now we will plot the histograms again after we have applied the transformations.

In [ ]:
```
df_reshape["new_price"].hist(bins=30)
```

In [ ]:
```
df_reshape["new_travel_min_to_CBD"].hist(bins=30)
```

In [ ]:
```
df_reshape["new_Distance_to_hospital"].hist(bins=30)
```

In [ ]:
```
df_reshape["new_Distance_to_sc"].hist(bins=30)
```

Hence from the above graphs we can see that the histograms are now forming a bell shaped structure and hence the data have become more normally distributed. Hence after applying the ceratin power transformations we achieve good results in terms of the assumption Normalty. As a result now each column gets a fair chance and is used appropriately while creating the Linear Model.

Now taking a look at the linearity we can plot the scatter plots and regression line as shown below

In [ ]:
```
sns.regplot(df_reshape["new_price"], df_reshape["new_Distance_to_sc"])
```

In [ ]:
```
sns.regplot(df_reshape["new_price"], df_reshape["new_Distance_to_hospital"])
```

In [ ]:
```
sns.regplot(df_reshape["new_price"], df_reshape["new_travel_min_to_CBD"])
```

For each graph above we can find that there is a negative corelation that exists between the preidctor and the target columns. After applying the transformations, the data points are more evenly spread across the regrssion line. As a result the data points are more towards the regression line and can be plotted across it. Hence an even distribution of the data points is seen. Hence this regression line best fits the data. This line is represented by

y = ax1 + bx2 + cx3 + d,

where y is the target variable and x represents the different predictors and the a,b,c,d are the coefficient values. The more the points lie across this line the better the model wiill be. Hence after the above transformations we can see that the datapoints are now closely related to each other as well as closely located near the regression line.

Hence after applying these transformation techniques we have achieved better results of the predictors. Hence the normality and linearity are not better as can be seen from the histogram and scatter plots above.

In [ ]:
```
df_reshape.corr()
```

Hence the columns new_price, new_Distance_to_hospital ,new_Distance_to_sc and new_travel_min_to_CBD are now better in terms of linearity and normality and we can see the correlation amongst them from the table above.

5.3 CHECKING SCALING

Now to scale the data to a definite range we can choose ZScore Scaling or MinMax Scaling. Here I have used ZScore Normalisation as when the data is uniformally distributed and forms a bell shape histogram then we usually go for this approach. This is because after applying this approach the data points are centered across 0 with a standard deviation of 1. Hence to apply StandardScaler() we first fit the data columns into the fit function and then use the transform function on them. The resultant values rae the scaled version of the given columns and hence they can be used to create the final columns.

Z SCORE NORMALISATION

In [ ]:
```
df_reshape.describe()
```

In [ ]:
```

```

As we are preparing the data for a linear model hence we can do scaling of the predictors and target. It is not mandatory as the Linear model handles the scales by taking different values of the coeeficient variable. But to give the predictors a fair chance we do scaling.

In [ ]:
```
from sklearn import preprocessing
```

In [ ]:
```
#fitting and transforming on StandardScaler()
std_scale = preprocessing.StandardScaler().fit(df_reshape[['new_Distance_to_hospi
df_std = std_scale.transform(df_reshape[['new_Distance_to_hospital','new_Distance
df_std
```

In [ ]:
```
df_reshape['Scaled_Distance_to_sc'] = df_std[:,0] # so 'Ascaled' is Alcohol scale
df_reshape['Scaled_travel_min_to_CBD'] = df_std[:,1]
df_reshape['Scaled_Distance_to_hospital'] = df_std[:,2]
df_reshape['Scaled_price'] = df_std[:,3]# and 'MAscaled' is Malic acid scaled
df_reshape.head()
```

In [ ]:
```
df_reshape.describe()
```

Hence the new scaled columns now have mean = 0 and SD = 1

In [ ]:
```
%matplotlib inline
```

In [ ]:
```
df_reshape["Distance_to_hospital"].plot(), df_reshape["Distance_to_sc"].plot(),df
```

As all the columns have different scales hence it becomes difficult to plot all the 2013 points on the same graph. Hence to have a better visulisation we plot each individually.

In [ ]:
```
df_reshape["Distance_to_hospital"].plot()
```

In [ ]:
```
df_reshape["Distance_to_sc"].plot()
```

In [ ]:
```
df_reshape["travel_min_to_CBD"].plot()
```

In [ ]:
```
df_reshape["price"].plot()
```

Now after performing standard scaling we can see all the columns together in a single graph as they have been standardised and can be plotted on a similar y axis range

In [ ]:
```
df_reshape["Scaled_Distance_to_hospital"].plot(), df_reshape["Scaled_Distance_to_
```

Z score scaling handles outliers as well which is not seen in MINMAX scaling. Hence the final columns become

"Scaled_Distance_to_sc"

"Scaled_Distance_to_hospital"

"Scaled_travel_min_to_CBD"

"Scaled_price"

6. Summary

In this assignment we have learned how to handle and write files of different formats like pdf, json, xml, etc. We learned how to extract the data values from these files and create dataframes out of them. We also played around with the distance calculations to determine nearest shopping centers, train stations, etc to a particular property id based on the latitude and longitude points. We also learned how to determine direct trips between two train stations which also helped me to learn more about the Dataset. We also learned how to deal with shape files and determine whether a particular point lies within a suburb or not by creating polygons of the suburb. In this assignment we also learned how to reshape and transform data using log, power, sqrt techniques. Using these we transformed and scaled our data to present it well to the Linear model.

In [ ]:

In [ ]:

In [ ]: