

Property Price Prediction

Alan Cahill

C00174533

Lecturer:

Dr. Lei Shi

Technical Manual

**Code**

**Cork Data Maker**

import csv

import googlemaps

import itertools

import os

os.environ["GOOGLE\_API\_KEY"] = 'AIzaSyC6c29GBWSO3xkQhMPgPuYhdMpn3XgCMhA'

import geocoder

import re

api\_key = 'AIzaSyB\_YV5ioa-B8wQf83dC5hpOslDeXlAumLE'

def open\_propertyregister():

with open('..\\..\\Database\\propregister.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

###Import propregister file

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'DateOfSale': '',

'address': '',

'price': '',

'LatLng': ''

}

if i and a > 0:

#add appropriate data to the dictionary.

hinfo['DateOfSale'] = i[0]

hinfo['address'] = i[1]

hinfo['price'] = i[2]

hinfo['LatLng']=i[3]

#append to all\_houses

all\_houses.append(hinfo)

a += 1

return all\_houses

def geocode\_registerhouses():

num = 0

registerprice = open\_propertyregister()

# initialise registerprice to the return of open\_propertyregister()

for i in registerprice:

# iter through register price

address = i['address'] + 'Cork'

#set adrees to the address of i

g = geocoder.google(address,key='AIzaSyC6c29GBWSO3xkQhMPgPuYhdMpn3XgCMhA')

# geocode the address and add it to i['LatLng']

i['LatLng'] = g.latlng

return registerprice

def compareGeocode(houses):

#initialise registerprice as the return of geocode registerhouses

registerprice = geocode\_registerhouses()

num = 0

for house in houses:

#loop through houses

house['dateofsale'] = ''

house['saleprice'] = ''

#initialise house dateofsale and saleprice

for h in registerprice:

#loop registerprice

if house['LatLng'] == h['LatLng']:

#if the LatLngs are the same then update dateofsale and saleprice

house['dateofsale'] = h['DateOfSale']

house['saleprice'] = h['price']

num += 1

#break the registerprice loop

break

return houses

def createCorkCity() :

with open('..\\..\\..\\Database\\CorkHouses.csv') as csv\_file:

#open corkhouses and convert to a dictionary

readCSV = csv.reader(csv\_file, delimiter=',')

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'area': '',

'Garage': '',

'Garden': '',

'Ensuite': ''

}

if i and a > 0:

hinfo['price'] = i[0]

hinfo['address'] = i[1]

hinfo['bedrooms'] = i[2]

hinfo['bathrooms'] = i[3]

hinfo['type'] = i[4]

hinfo['area'] = i[5]

hinfo['Garage'] = i[6]

hinfo['Garden'] = i[7]

hinfo['Ensuite'] = i[8]

all\_houses.append(hinfo)

a += 1

corkc = []

for i in all\_houses:

if "Cork City" in i['address']:

#loop through all-houses and if cork city is in the address then append to corkc

corkc.append(i)

#set keys and save corkc to csv file

keys = all\_houses[0].keys()

with open('..\\..\\Database\\CorkCityHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(corkc)

return(corkc)

def googlecode(my\_list):

#os.environ["GOOGLE\_API\_KEY"] = 'AIzaSyDqvHU4A74hFNgU0MLGPs80\_ZR8clIdoH4'

#gmaps = googlemaps.Client(api\_key)

for i in my\_list:

address = i['address'] + 'Cork'

#loop mylist and gecode all the addresses

g = geocoder.google(address,key='AIzaSyC6c29GBWSO3xkQhMPgPuYhdMpn3XgCMhA')

print(g)

#add the latlng to i['LatLng]

i['LatLng'] = g.latlng

#use the compare method to compare the latlng and match a house with a sale price

my\_list= compareGeocode(my\_list)

allhouses = []

for house in my\_list:

if house['LatLng']:

allhouses.append(house)

keys =allhouses[0].keys()

#wite to the file

with open('..\\..\\Database\\CorkCityHousesComplete.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(allhouses)

**Property Register Data Maker**

import csv

import itertools

import geocoder

import os

import time

os.environ["GOOGLE\_API\_KEY"] = 'AIzaSyDD5oIRo5GEgb\_26iG2eEhSvAVVxYoxYZ4'

def geocode\_registerhouses(registerprice):

for i in registerprice:

address = i['address'] + 'Dublin'

#add dublin to string for syntax purposes

#geocode the address

g = geocoder.google(address,key='AIzaSyC6c29GBWSO3xkQhMPgPuYhdMpn3XgCMhA')

print(i['address'])

#add it to the instance

i['LatLng'] = g.latlng

print(g)

#sleep to ensure no overloading of the server

time.sleep(.1)

return registerprice

with open('..\\PPR-2018.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'DateOfSale': '',

'address': '',

'price': '',

}

if i and a > 0:

#i[1] = re.findall(r"'(.\*?)'", i[1], re.DOTALL)

hinfo['DateOfSale'] = i[0]

hinfo['address'] = i[1]

hinfo['price'] = i[4]

all\_houses.append(hinfo)

a += 1

#open the csv into a dictionary

for h in all\_houses:

if type(h['address']) is not type('string'):

h['address'] = ''.join(str(h['address']))

h['price'] = h['price'].replace(",","")

# make the address a string to avoid errors

#replace all commas with nothing

with open('..\\PPR-2019.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

a = 0

ninteen\_houses = []

for i in readCSV:

hinfo = {

'DateOfSale': '',

'address': '',

'price': '',

}

if i and a > 0:

# i[1] = re.findall(r"'(.\*?)'", i[1], re.DOTALL)

hinfo['DateOfSale'] = i[0]

hinfo['address'] = i[1]

hinfo['price'] = i[4]

ninteen\_houses.append(hinfo)

a += 1

#repeat of previous loading conversion

for h in ninteen\_houses:

if type(h['address']) is not type('string'):

h['address'] = ''.join(str(h['address']))

h['price'] = h['price'].replace(",", "")

eighteenhouses = []

for h in all\_houses:

h['price'] = h['price'].replace(",", "")

h['price'] = h['price'].replace("€", "")

h['price'] = h['price'].replace(".00", "")

price = float(h['price'])/5000

price = int(price) \* 5000

h['price'] = price

eighteenhouses.append(h)

#remove all commas euro signs and decimals

#use the math standars of python to round the price to its closest 5000

for h in ninteen\_houses:

h['price'] = h['price'].replace(",", "")

h['price'] = h['price'].replace("€", "")

h['price'] = h['price'].replace(".00", "")

price = float(h['price']) / 5000

price = int(price) \* 5000

h['price'] = price

eighteenhouses.append(h)

#repeat the price processing for the ninteen houses dataset

counter = 0

print(len(eighteenhouses))

dubfifthouses = []

for h in eighteenhouses:

if 'dublin 15' in h['address'].lower():

counter+=1

dubfifthouses.append(h)

dubfifthouses = geocode\_registerhouses(dubfifthouses)

print(counter)

keys = dubfifthouses[0].keys()

print(keys)

with open('..\\..\\Database\\propregister.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(dubfifthouses)

**CsV SPlitter**

import re

import csv

def splitCSV():

with open('..\\..\\Database\\allviablehousescsv.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

#import the csv file and convert it to a dictionary

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'area': '',

'Garage' : '',

'Garden' : '',

'Ensuite': ''

}

if i and a > 0:

#regular expression to find anything inside single quotes

#. is anything except newline

#\*any amount of combinations of the preceding syntax

#? only a or ab of the preceding syntax

#DOTALL allows newline to be found

i[1] = re.findall(r"'(.\*?)'", i[1], re.DOTALL)

i[1] = "".join(i[1])

i[4] = re.findall(r"'(.\*?)'", i[4], re.DOTALL)

i[4] = "".join(i[4])

i[0] = float(i[0])

hinfo['price'] = i[0]

hinfo['address'] = i[1]

hinfo['bedrooms'] = i[2]

hinfo['bathrooms'] = i[3]

hinfo['type'] = i[4]

hinfo['area'] = i[5]

hinfo['Garage'] = i[6]

hinfo['Garden'] = i[7]

hinfo['Ensuite'] = i[8]

all\_houses.append(hinfo)

a += 1

#all\_houses = add\_date\_and\_sale\_price(all\_houses)

onehund = []

halfmill = []

onemill = []

twomill = []

overtwo = []

Antrim = []

Armagh = []

Carlow = []

Cavan = []

Clare = []

Cork = []

Donegal = []

Down = []

Dublin = []

Fermanagh = []

Galway = []

Kerry = []

Kildare = []

Kilkenny = []

Laois = []

Leitrim = []

Limrick = []

Derry = []

Longford = []

Louth = []

Mayo = []

Meath = []

Monaghan = []

Offaly = []

Roscommon = []

Sligo = []

Tipperary = []

Tyrone = []

Waterford = []

Westmeath = []

Wexford = []

Wicklow = []

OtherCounty = []

DetHouse = []

SemDetHouse = []

TerrHouse = []

AppSale = []

Bungalow = []

DuplexSale = []

EndTHouse = []

NewDHA = []

SHDHA = []

SiteSale = []

Townhouse = []

HouseSale = []

OtherType = []

for house in all\_houses:

if house['price'] < 100000:

onehund.append(house)

elif house['price'] < 500000:

halfmill.append(house)

elif house['price'] < 1000000:

onemill.append(house)

elif house['price'] < 2000000:

twomill.append(house)

else:

overtwo.append(house)

# if the county is in the address string then append the house to the county list

if 'antrim' in house['address'].lower():

Antrim.append(house)

house['county'] = 'antrim'

elif 'armagh' in house['address'].lower():

Armagh.append(house)

house['county'] ='armagh'

elif 'carlow' in house['address'].lower():

Carlow.append(house)

house['county'] ='carlow'

elif 'cavan' in house['address'].lower():

Cavan.append(house)

house['county'] ='cavan'

elif 'clare' in house['address'].lower():

Clare.append(house)

house['county'] ='clare'

elif 'cork' in house['address'].lower():

Cork.append(house)

house['county'] = 'cork'

elif 'donegal' in house['address'].lower():

Donegal.append(house)

house['county'] = 'donegal'

elif 'down' in house['address'].lower():

Down.append(house)

house['county'] ='down'

elif 'dublin' in house['address'].lower():

Dublin.append(house)

house['county'] ='dublin'

elif 'fermanagh' in house['address'].lower():

Fermanagh.append(house)

house['county'] ='fermanagh'

elif 'galway' in house['address'].lower():

Galway.append(house)

house['county'] ='galway'

elif 'kerry' in house['address'].lower():

Kerry.append(house)

house['county'] ='kerry'

elif 'kildare' in house['address'].lower():

Kildare.append(house)

house['county'] ='kildare'

elif 'kilkenny' in house['address'].lower():

Kilkenny.append(house)

house['county'] ='kilkenny'

elif 'laois' in house['address'].lower():

Laois.append(house)

house['county'] ='laois'

elif 'leitrim' in house['address'].lower():

Leitrim.append(house)

house['county'] ='leitrim'

elif 'limerick' in house['address'].lower():

Limrick.append(house)

house['county'] ='limerick'

elif 'derry' in house['address'].lower():

Derry.append(house)

house['county'] ='derry'

elif 'longford' in house['address'].lower():

Longford.append(house)

house['county'] ='longford'

elif 'louth' in house['address'].lower():

Louth.append(house)

house['county'] ='louth'

elif 'mayo' in house['address'].lower():

Mayo.append(house)

house['county'] ='mayo'

elif 'meath' in house['address'].lower():

Meath.append(house)

house['county'] ='meath'

elif 'monaghan' in house['address'].lower():

Monaghan.append(house)

house['county'] ='monaghan'

elif 'offaly' in house['address'].lower():

Offaly.append(house)

house['county'] ='offaly'

elif 'roscommon' in house['address'].lower():

Roscommon.append(house)

house['county'] ='roscommon'

elif 'sligo' in house['address'].lower():

Sligo.append(house)

house['county'] ='sligo'

elif 'tipperary' in house['address'].lower():

Tipperary.append(house)

house['county'] ='tipperary'

elif 'tyrone' in house['address'].lower():

Tyrone.append(house)

house['county'] ='tyrone'

elif 'waterford' in house['address'].lower():

Waterford.append(house)

house['county'] ='waterford'

elif 'westmeath' in house['address'].lower():

Westmeath.append(house)

house['county'] ='westmeath'

elif 'wexford' in house['address'].lower():

Wexford.append(house)

house['county'] ='wexford'

elif 'wicklow' in house['address'].lower():

Wicklow.append(house)

house['county'] ='wicklow'

else:

OtherCounty.append(house)

house['county'] ='other'

if 'Detatched House' in house['type']:

DetHouse.append(house)

elif 'Semi-detatched House' in house['type']:

SemDetHouse.append(house)

elif 'Terraced House' in house['type']:

TerrHouse.append(house)

elif 'Apartment For Sale' in house['type']:

AppSale.append(house)

elif 'Bungalow' in house['type']:

Bungalow.append(house)

elif 'Semi-Detached' in house['type']:

SemDetHouse.append(house)

elif 'Detached' in house['type']:

DetHouse.append(house)

elif 'Duplex' in house['type']:

DuplexSale.append(house)

elif 'End of' in house['type']:

EndTHouse.append(house)

elif 'House For Sale' in house['type']:

HouseSale.append(house)

elif 'New Dwelling' in house['type']:

NewDHA.append(house)

elif 'Second-Hand' in house['type']:

SHDHA.append(house)

elif 'Site' in house['type']:

SiteSale.append(house)

elif 'Terraced' in house['type']:

TerrHouse.append(house)

elif 'Townhouse' in house['type']:

Townhouse.append(house)

else:

OtherType.append(house)

# write the county lists to a file

keys = all\_houses[0].keys()

with open('..\\..\\Database\\allhouseswithcounty.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(all\_houses)

with open('..\\..\\Database\\DublinHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Dublin)

with open('..\\..\\Database\\AntrimHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Antrim)

with open('..\\..\\Database\\ArmaghHuses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Armagh)

with open('..\\..\\Database\\CarlowHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Carlow)

with open('..\\..\\Database\\CavanHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Cavan)

with open('..\\..\\Database\\ClareHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Clare)

with open('..\\..\\Database\\CorkHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Cork)

with open('..\\..\\Database\\DonegalHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Donegal)

with open('..\\..\\Database\\DownHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Down)

with open('..\\..\\Database\\FermanaghHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Fermanagh)

with open('..\\..\\Database\\GalwayHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Galway)

with open('..\\..\\Database\\KerryHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Kerry)

with open('..\\..\\Database\\KildareHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Kildare)

with open('..\\..\\Database\\KilkennyHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Kilkenny)

with open('..\\..\\Database\\LaoisHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Laois)

with open('..\\..\\Database\\LeitrimHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Leitrim)

with open('..\\..\\Database\\LimerickHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Limrick)

with open('..\\..\\Database\\DerryHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Derry)

with open('..\\..\\Database\\LongfordHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Longford)

with open('..\\..\\Database\\LouthHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Louth)

with open('..\\..\\Database\\MayoHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Mayo)

with open('..\\..\\Database\\MeathHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Meath)

with open('..\\..\\Database\\MonaghanHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Monaghan)

with open('..\\..\\Database\\OffallyHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Offaly)

with open('..\\..\\Database\\RoscommonHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Roscommon)

with open('..\\..\\Database\\SligoHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Sligo)

with open('..\\..\\Database\\TipperaryHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Tipperary)

with open('..\\..\\Database\\TyroneHouses.csv', 'w+') as output\_file:

dict\_writer = csv.writer(output\_file, dialect='excel')

dict\_writer.writerows(Tyrone)

with open('..\\..\\Database\\WaterfordHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Waterford)

with open('..\\..\\Database\\WestmeathHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Westmeath)

with open('..\\..\\Database\\WexfordHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Wexford)

with open('..\\..\\Database\\WicklowHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Wicklow)

with open('..\\..\\Database\\DetatchedHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(DetHouse)

with open('..\\..\\Database\\SemiDetatchedHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(SemDetHouse)

with open('..\\..\\Database\\TerracedHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(TerrHouse)

with open('..\\..\\Database\\AppartmentsHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(AppSale)

with open('..\\..\\Database\\BungalowHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Bungalow)

with open('..\\..\\Database\\DuplexHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(DuplexSale)

with open('..\\..\\Database\\EndTerraceHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(EndTHouse)

with open('..\\..\\Database\\NewDwellingHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(NewDHA)

with open('..\\..\\Database\\SecondHandHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(SHDHA)

with open('..\\..\\Database\\SitesHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(SiteSale)

with open('..\\..\\Database\\TownHouseHouses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(Townhouse)

with open('..\\..\\Database\\HouseForSale.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(HouseSale)

with open('..\\..\\Database\\OneHundred.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(onehund)

with open('..\\..\\Database\\HalfMill.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(halfmill)

with open('..\\..\\Database\\OneMill.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(onemill)

with open('..\\..\\Database\\TwoMill.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(twomill)

with open('..\\..\\Database\\OverTwo.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(overtwo)

print(len(OtherCounty))

**Daft Rent Spider**

import sqlite3

def run\_daft\_rentspider():

st = datetime.now()

daftstr = 'https://www.daft.ie'

links = rentlinkspider.crawl\_links()

i = 1

db = sqlite3.connect('dafthouses.db')

cursor = db.cursor()

cursor.execute('DELETE FROM rentcurrenttable')

print(links)

print(datetime.now())

for link in links:

link = daftstr+''.join(link)

data = rentscraper.scrapedetails(link)

houseinfo = {

'price': data[0],

'address':data[1],

'type':data[2],

'details':data[3],

'furnished': data[4],

'facilities':data[5],

'ccdistance':data[6],

}

print(type(houseinfo['price']))

print(type(houseinfo['address']))

typestr ="".join( houseinfo['type'])

print(type(houseinfo['details']))

print(type(houseinfo['furnished']))

print(type(houseinfo['ccdistance']))

print(type(houseinfo['facilities']))

print(houseinfo['ccdistance'])

print(houseinfo['facilities'])

newstr = houseinfo['facilities'].replace("/", "")

keys = houseinfo.keys()

cursor.execute('''INSERT INTO rentcurrenttable(price,address,type,details,furnished,facilities,ccdistance)

VALUES(?,?,?,?,?,?,?)''', (houseinfo['price'], houseinfo['address'], typestr,

houseinfo['details'], houseinfo['furnished'],

newstr, houseinfo['ccdistance']))

print('House' + str(i) + 'Inserted')

i += 1

db.commit()

print('alan')

#tablescleaner.addtofulltables()

print('Total download and databsase addition time:')

print(datetime.now() - st)

**Daft Spider**

import tablescleaner

def run\_spider():

st = datetime.now()

daftstr = 'https://www.daft.ie'

# the links in daft.ie are dynamic. ie. they are internal links that dont contain the https

# this needs to be added if a link is to be opened

links = linkspider.crawl\_links()

# this assigns the links to the links variable.

# these links are retrieved by the crawllinks function

i = 1

db = sqlite3.connect('dafthouses.db')

cursor = db.cursor()

cursor.execute('DELETE FROM currenthouses')

# remove all values from the current houses table

print(datetime.now())

for link in links:

# appends the daftstring 'htts' etc to the dynamic link

link = daftstr+''.join(link)

data = salesscraper.scrapedetails(link)

houseinfo = {

'price': data[0],

'address':data[1],

'bedrooms':data[2],

'bathrooms':data[3],

'type' : data[4].split('|',1)[0],

'details':data[5],

'views': data[6],

'all\_prices':data[7],

'price\_dates':data[8],

'area' : data[9]

}

keys = houseinfo.keys()

# convert data to a list of dicts

#send data to the database

cursor.execute('''INSERT INTO currenthouses(price,address,bedrooms,bathrooms,type,details,views,allprices,pricedates,area)

VALUES(?,?,?,?,?,?,?,?,?,?)''', (houseinfo['price'],houseinfo['address'],houseinfo['bedrooms'],

houseinfo['bathrooms'], houseinfo['type'],

houseinfo['details'], houseinfo['views'],

houseinfo['all\_prices'],houseinfo['price\_dates'],

houseinfo['area']))

i +=1

db.commit()

#tablescleaner.addtofulltables()

print('Total download and databsase addition time:')

print(datetime.now() - st)

**Data Base to Csv**

mport csv

import sqlite3

db = sqlite3.connect('dafthouses.db')

cursor = db.cursor()

current = cursor.execute('Select \* FROM viablehousestable')

currenthouses = []

for i in current:

a = 0

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'views': '',

'area': ''

}

for t in i:

if a is 0: hinfo['id'] = t

if a is 1:

hinfo['price'] = t

elif a is 2:

hinfo['address'] = t.encode("utf-8")

elif a is 3:

hinfo['bedrooms'] = t

elif a is 4:

hinfo['bathrooms'] = t

elif a is 5:

hinfo['type'] = t.encode("utf-8")

elif a is 7:

hinfo['area'] = t

a += 1

currenthouses.append(hinfo)

keys = currenthouses[0].keys()

with open('viablecsv.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(currenthouses)

**Data Cleaner**

import csv

def create\_csvs() :

db = sqlite3.connect('dafthouses.db')

#import database and transfer it to a dictionary

cursor = db.cursor()

current = cursor.execute('Select \* FROM mainhousestable')

myhouses = []

for i in current:

a = 0

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'details': '',

'area': ''

}

for t in i:

if a is 1:hinfo['price']=t

elif a is 2: hinfo['address']=t.encode("utf-8")

elif a is 3:hinfo['bedrooms'] = t

elif a is 4:hinfo['bathrooms'] = t

elif a is 5:hinfo['type'] = t.encode("utf-8")

elif a is 6:hinfo['details'] = t

#elif a is 7:hinfo['views'] = t

elif a is 10:hinfo['area'] = t

a+=1

myhouses.append(hinfo)

#call the cleaning functions and reinitise myhouses to the return value

print('Starting Cleaning')

myhouses = additional\_columns(myhouses)

myhouses = currency\_and\_strings(myhouses)

print('Currency')

myhouses = removenovalues(myhouses)

print('nullvals')

myhouses = change\_to\_int(myhouses)

print('Changed Strings to Int')

myhouses = remove\_duplicates(myhouses)

print('Duplicates')

for i in myhouses:

del i['details']

#remove details as not needed anymore

#write to file

keys = myhouses[0].keys()

with open('..\\..\\Database\\allviablehousescsv.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(myhouses)

def currency\_and\_strings(houses):

for t in houses:

#set the price to the digits returned by the regular expression.

t['price'] = re.findall('\d+', t['price'])

if t['bedrooms']:

# set bedrooms to the digits returned by the re.

t['bedrooms'] = re.findall('\d+', str(t['bedrooms']))

if t['bathrooms']:

#ste bathrooms to the digits returned by the re.

t['bathrooms'] = re.findall('\d+', str(t['bathrooms']))

#change them back to strings for future ease

if t['price']:

t['price'] = ''.join(t['price'])

else:

t['price'] = ''

if t['bedrooms']:

t['bedrooms'] = ''.join(t['bedrooms'])

else:

t['bedrooms'] = ''

if t['bathrooms']:

t['bathrooms'] = ''.join(t['bathrooms'])

else:

t['bathrooms'] = ''

return houses

def removenovalues(houses):

nothouse = []

#if the value is null then append the full house to the new list

for house in houses:

if not house['price']:

nothouse.append(house)

elif not house['bedrooms']:

nothouse.append(house)

elif not house['bathrooms']:

nothouse.append(house)

elif house['area'] is 0:

nothouse.append(house)

elif not house['area']:

nothouse.append(house)

b = 0

# loop the list and remove the corrosponding value from the other list

for house in nothouse:

b+=1

houses.remove(house)

return houses

def change\_to\_int(houses):

for house in houses:

house['price'] = float(house['price'])

house['bathrooms'] = int(house['bathrooms'])

house['bedrooms'] = int(house['bedrooms'])

house['area'] = int(house['area'])

#change all to int

return houses

def remove\_duplicates(houses):

dup = False

newhouses = []

for house in houses:

#loop the houses list

for i in newhouses:

#loop the new houses list

if house == i:

#if the house is already in the newhouses list set dup to True

dup = True

if dup is False:

#if dup is false then there is no existing duplicate so,

# append the house to the newhouses

newhouses.append(house)

dup = False

return newhouses

def additional\_columns(houses):

for i in houses:

# check the details for the specified substring

#if the substring exists

#set the key to true

#otherwise set it as false

if 'garage' in i['details']:

i['Garage'] = 'Yes'

elif 'Garage' in i['details']:

i['Garage'] = 'Yes'

else: i['Garage'] = 'No'

if 'garden' in i['details']:

i['Garden'] = 'Yes'

elif 'Garden' in i['details']:

i['Garden'] = 'Yes'

else: i['Garden'] = 'No'

if 'ensuite' in i['details']:

i['Ensuite'] = 'Yes'

elif 'en-suite' in i['details']:

i['Ensuite'] = 'Yes'

elif 'EnSuite' in i['details']:

i['Ensuite'] = 'Yes'

elif 'En-Suite' in i['details']:

i['Ensuite'] = 'Yes'

else: i['Ensuite'] = 'No'

return houses

**Details Stats**

import sqlite3

import re

import csv

db = sqlite3.connect('dafthouses.db')

cursor = db.cursor()

current = cursor.execute('Select \* FROM currenthouses')

currenthouses = []

for i in current:

a = 0

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'details': '',

'views': '',

'all\_prices': '',

'price\_dates': '',

'area': ''

}

for t in i:

if a is 0:hinfo['id'] = t

if a is 1:hinfo['price']=t

elif a is 2: hinfo['address']=t

elif a is 3:hinfo['bedrooms'] = t

elif a is 4:hinfo['bathrooms'] = t

elif a is 5:hinfo['type'] = t

elif a is 6:hinfo['details'] = t

elif a is 7:hinfo['views'] = t

elif a is 8:hinfo['all\_prices'] = t

elif a is 9:hinfo['price\_dates'] = t

elif a is 10:hinfo['area'] = t

a+=1

currenthouses.append(hinfo)

garage = 0

conservatory = 0

underfloorheating = 0

ensuite = 0

garden = 0

land = 0

acres = 0

count = 0

hectare = 0

mylist = []

newlist = []

for i in currenthouses:

myacre = ''

myacre = re.search(r'([-+]?\d\*\.\d+|\d+) acres', i['details'])

if myacre:

mylist.append(myacre.group(1))

mylist.append(i['address'])

else:

myacre = re.search(r'(\d+) Acres', i['details'])

if myacre: mylist.append(myacre.group(1))

else:

myacre = re.search(r'(\d+)Acres', i['details'])

if myacre: mylist.append(myacre.group(1))

else:

myacre = re.search(r'(\d+)acres', i['details'])

if myacre: mylist.append(myacre.group(1))

else:

myacre = re.search(r'(\d+) acres', i['details'])

if myacre: mylist.append(myacre.group(1))

if 'garage' in i['details']:

garage +=1

elif 'Garage' in i['details']:

garage +=1

if 'garden' in i['details']:

garden += 1

elif 'Garden' in i['details']:

garden += 1

if 'conservatry' in i['details']:

conservatory += 1

elif 'Conservatory' in i['details']:

conservatory += 1

if 'underfloor' in i['details']:

underfloorheating += 1

elif 'under-floor' in i['details']:

underfloorheating += 1

elif 'UnderFloor' in i['details']:

underfloorheating += 1

elif 'Under-Floor' in i['details']:

underfloorheating += 1

if 'ensuite' in i['details']:

ensuite += 1

elif 'en-suite' in i['details']:

ensuite += 1

elif 'EnSuite' in i['details']:

ensuite += 1

elif 'En-Suite' in i['details']:

ensuite += 1

if 'acres' in i['details']:

acres += 1

elif 'Acres' in i['details']:

acres += 1

if 'Hectare' in i['details']:

hectare += 1

#mytown = re.search(r'(\d+) Acres', i['details'])

#if mytown: mylist.append(mytown.group(1))

#mytown = re.search(r'(\d+)Acres', i['details'])

#if mytown: mylist.append(mytown.group(1))

#mytown = re.search(r'(\d+)acres', i['details'])

#if mytown: mylist.append(mytown.group(1))

count += 1

print('Ensuite : ', ensuite)

print('UnDerFloor : ', underfloorheating)

print('Conservatory : ', conservatory)

print('Garden : ', garden)

print('Garage : ', garage)

print('Acres : ', acres)

print('Hectare : ', hectare)

print('Total :', count)

#for i in mylist: print(i)

**Dublin Data Maker**

import csv

import googlemaps

import itertools

import os

os.environ["GOOGLE\_API\_KEY"] = 'AIzaSyC6c29GBWSO3xkQhMPgPuYhdMpn3XgCMhA'

import geocoder

import re

api\_key = 'AIzaSyB\_YV5ioa-B8wQf83dC5hpOslDeXlAumLE'

def open\_propertyregister():

with open('..\\..\\Database\\propregister.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

#convert csv to dictionary

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'DateOfSale': '',

'address': '',

'price': '',

'LatLng': ''

}

if i and a > 0:

hinfo['DateOfSale'] = i[0]

hinfo['address'] = i[1]

hinfo['price'] = i[2]

hinfo['LatLng']=i[3]

all\_houses.append(hinfo)

a += 1

return all\_houses

def geocode\_registerhouses():

registerprice = open\_propertyregister()

for i in registerprice:

address = i['address'] + 'Dublin'

#geocode the addresses and add a latlng column

g = geocoder.google(address,key='AIzaSyC6c29GBWSO3xkQhMPgPuYhdMpn3XgCMhA')

i['LatLng'] = g.latlng

return registerprice

def compareGeocode(houses):

#

registerprice = geocode\_registerhouses()

num = 0

for house in houses:

house['dateofsale'] = ''

house['saleprice'] = ''

#initialise the houses columns to null

for h in registerprice:

if house['LatLng'] == h['LatLng']:

#compare the latlngs and if they are the same then add on the details

house['dateofsale'] = h['DateOfSale']

house['saleprice'] = h['price']

num += 1

print('SAME')

break

return houses

def createDublin15() :

with open('..\\..\\Database\\DublinHouses.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

#convert the csv to a dict

a = 0

all\_houses = []

for i in readCSV:

print(i)

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'area': '',

'Garage': '',

'Garden': '',

'Ensuite': ''

}

if i and a > 0:

hinfo['price'] = i[0]

hinfo['address'] = i[1]

hinfo['bedrooms'] = i[2]

hinfo['bathrooms'] = i[3]

hinfo['type'] = i[4]

hinfo['area'] = i[5]

hinfo['Garage'] = i[6]

hinfo['Garden'] = i[7]

hinfo['Ensuite'] = i[8]

all\_houses.append(hinfo)

a += 1

dub\_15 = []

for i in all\_houses:

if "Dublin 15" in i['address']:

dub\_15.append(i)

# append all dublin15 addresses to the new

keys = all\_houses[0].keys()

with open('..\\..\\Database\\Dublin15Houses.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(dub\_15)

return(dub\_15)

def googlecode(my\_list):

#loop geocode and add the latlng as usual

for i in my\_list:

print(1)

address = i['address'] + 'Dublin'

g = geocoder.google(address,key='AIzaSyC6c29GBWSO3xkQhMPgPuYhdMpn3XgCMhA')

i['LatLng'] = g.latlng

print(g)

my\_list= compareGeocode(my\_list)

allhouses = []

for house in my\_list:

if house['LatLng']:

allhouses.append(house)

#write to file

keys =allhouses[0].keys()

with open('..\\..\\Database\\Dublin15HousesComplete.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(allhouses)

**Link Gatherer**

rom bs4 import BeautifulSoup

import requests

from datetime import datetime

def twentylinks(link):

# create a session and a user agent.

# daft requires this to access their website

session = requests.Session()

session.headers.update({'User-Agent': 'Custom user agent'})

html =session.get(link)

# assign the contents of the link to the html variable

html = html.text

# convert it to beautiful soup

soup = BeautifulSoup(html, 'html5lib')

# this retrievs the class specified which was researched to be the location of the link

links = [a.get('href') for a in soup.find\_all('a',{'class': 'PropertyInformationCommonStyles\_\_propertyPrice--link'}, href=True)]

# the sublink is exactly the same

nextlink = soup('li',{'class':'next\_page'})

nextpage = []

for li in nextlink:

# add pages to the next page list

nextpage.append(li.a['href'])

return links,nextpage

**Link Spider**

from datetime import datetime

import LinkGatherer

def crawl\_links():

st = datetime.now()

daftstr = 'https://www.daft.ie'

alllinks = []

link = LinkGatherer.twentylinks('https://www.daft.ie/ireland/property-for-sale/?ad\_type=sale&advanced=1&s%5Badvanced%5D=1&searchSource=sale')

#get the first 20 links using the property for sale link as a base start point

for i in link[0]: alllinks.append(i)

# while there is a link in link[1] continue to travese the website to gather links

# this link[1] is assigned in twentylinks

while link[1]:

nextlink = daftstr+''.join(link[1])

link = LinkGatherer.twentylinks(nextlink)

print(link[1])

#append to all links

for i in link[0]: alllinks.append(i)

print (datetime.now()-st)

return alllinks

**rentlink gatherer**

from bs4 import BeautifulSoup

import requests

from datetime import datetime

def twentylinks(link):

session = requests.Session()

session.headers.update({'User-Agent': 'Custom user agent'})

html =session.get(link)

html = html.text

soup = BeautifulSoup(html, 'html5lib')

headers = soup('h2')

links = []

for i in headers:

for a in i.find\_all('a',href=True):

links.append(a['href'])

nextlink = soup('li',{'class':'next\_page'})

nextpage = []

for li in nextlink:

nextpage.append(li.a['href'])

return links,nextpage

**rentlinkspider**

from bs4 import BeautifulSoup

import requests

from datetime import datetime

import rentlinkgatherer

def crawl\_links():

st = datetime.now()

daftstr = 'https://www.daft.ie'

alllinks = []

link = rentlinkgatherer.twentylinks('https://www.daft.ie/ireland/residential-property-for-rent/?ad\_type=rental&advanced=1&s%5Badvanced%5D=1&searchSource=rental')

for i in link[0]: alllinks.append(i)

while link[1]:

nextlink = daftstr+''.join(link[1])

link = rentlinkgatherer.twentylinks(nextlink)

print(link[1])

for i in link[0]: alllinks.append(i)

print(len(alllinks))

print(datetime.now()-st)

return alllinks

**rent scraper**

from bs4 import BeautifulSoup

import requests

def scrapedetails(link):

session = requests.Session()

session.headers.update({'User-Agent': 'Custom user agent'})

html =session.get(link)

html = html.text

soup = BeautifulSoup(html, 'html5lib')

price = ''

address = ''

furnished = ''

type = []

movein = ''

extradetails = ''

facilities = ''

ccdistance = ''

for tag in soup.find\_all('div', {'id': 'smi-price-string'}):

price = tag.text

for tag in soup.find\_all('div', {'class': 'smi-object-header'}):

address = tag.find('h1').text

for tag in soup.find\_all('span', {'class': 'header\_text'}):

type.append(tag.text)

for tag in soup.find\_all('div', {'id': 'overview'}):

furnished = tag.text

for tag in soup.find\_all('div', {'id': 'description'}):

extradetails = tag.text

for tag in soup.find\_all('table', {'id': 'facilities'}):

facilities = tag.text

for tag in soup.find\_all('div', {'class': 'map\_info\_box'}):

ccdistance = tag.text

return price, address, type, extradetails, furnished, facilities, ccdistance

**Run Spider and Pre Processing**

import daftspider

import daftrentspider

import DataCleaner

import CSVSplitter

import DublinDataMaker

import tablescleaner

import CorkDataMaker

import tablecreaters

daftspider.run\_spider()

daftrentspider.run\_daft\_rentspider()

tablescleaner.addtofulltables()

DataCleaner.create\_csvs()

CSVSplitter.splitCSV()

dubhouses = DublinDataMaker.createDublin15()

DublinDataMaker.googlecode(dubhouses)

corkhouses = CorkDataMaker.createCorkCity()

CorkDataMaker.googlecode(corkhouses)

**Sales scraper**

mport re

def scrapedetails(link):

session = requests.Session()

session.headers.update({'User-Agent': 'Custom user agent'})

# user agen as per daft.ie requirments

html =session.get(link)

# get the html from the link and convert it to text

html = html.text

soup = BeautifulSoup(html, 'html5lib')

# then convert it to beautiful soup

price = ''

address = ''

bedrooms = ''

bathrooms =''

house\_type = ''

extradetails = ''

dateentered = ''

price\_history = ''

dates\_listed = ''

meters = ''

# use the researched tags to retrieve the information that id needed from the beautiful soup instance

for strong\_tag in soup.find\_all('strong', {'class': 'PropertyInformationCommonStyles\_\_costAmountCopy'}):

price = strong\_tag.text

for tag in soup.find\_all('h1', {'class': 'PropertyMainInformation\_\_address'}):

address = tag.text

for tag in soup.find\_all('div', {'class': 'QuickPropertyDetails\_\_iconCopy'}):

bedrooms = tag.text

for tag in soup.find\_all('div', {'class': 'QuickPropertyDetails\_\_iconCopy--WithBorder'}):

bathrooms = tag.text

for tag in soup.find\_all('div', {'class': 'QuickPropertyDetails\_\_propertyType'}):

house\_type = tag.text

for tag in soup.find\_all('p', {'class': 'PropertyDescription\_\_propertyDescription '

'is-expandable PropertyDescription\_\_propertyDescription--'

'is-collasped'}):

extradetails = tag.text

for tag in soup.find\_all('div', {'class': 'PropertyStatistics\_\_iconData'}):

dateentered = tag.text

for tag in soup.find\_all('div', {'class': 'PropertyPriceHistory\_\_propertyPrice'}):

price\_history = tag.text

for tag in soup.find\_all('div', {'class': 'PropertyPriceHistory\_\_propertyPriceDate'}):

dates\_listed = tag.text

for tag in soup.find\_all('div', {'class': 'PropertyOverview\_\_propertyOverviewDetails'}):

overview\_details = tag.text

meters = re.findall('\d+',overview\_details)

if meters:

area = meters[0]

else:

area = 0

return price,address,bedrooms,bathrooms,house\_type,extradetails,dateentered,price\_history,dates\_listed,area

**tables creator**

import sqlite3

def create\_rent\_current():

db = sqlite3.connect('dafthouses.db')

cursor = db.cursor()

cursor.execute('''

CREATE TABLE if not exists rentcurrenttable(id INTEGER PRIMARY KEY, price TEXT,

address TEXT, type TEXT ,

details TEXT, furnished TEXT,

facilities TEXT, ccdistance TEXT)

''')

db.commit()

def create\_rent\_main():

db = sqlite3.connect('dafthouses.db')

cursor = db.cursor()

cursor.execute('''

CREATE TABLE if not exists rentmaintable(id INTEGER PRIMARY KEY, price TEXT,

address TEXT, type TEXT ,

details TEXT, furnished TEXT,

facilities TEXT, ccdistance TEXT)

''')

db.commit()

def create\_no\_duplicates():

db = sqlite3.connect('dafthouses.db')

cursor = db.cursor()

cursor.execute('''

CREATE TABLE if not exists mainhousestable(id INTEGER PRIMARY KEY, price TEXT,

address TEXT, bedrooms INTEGER ,

bathrooms INTEGER, type TEXT,

details TEXT, views INTEGER,

allprices TEXT, pricedates TEXT,

area INTEGER)

''')

db.commit()

def create\_viable():

db = sqlite3.connect('dafthouses.db')

cursor = db.cursor()

cursor.execute('''

CREATE TABLE if not exists viablehousestable(id INTEGER PRIMARY KEY, price INTEGER,

address TEXT, bedrooms INTEGER ,

bathrooms INTEGER, type TEXT,

details TEXT, views INTEGER,

allprices TEXT, pricedates TEXT,

area INTEGER)

''')

db.commit()

def create\_current():

db = sqlite3.connect('dafthouses.db')

cursor = db.cursor()

cursor.execute('''

CREATE TABLE if not exists currenthousestable(id INTEGER PRIMARY KEY, price INTEGER,

address TEXT, bedrooms INTEGER ,

bathrooms INTEGER, type TEXT,

details TEXT, views INTEGER,

allprices TEXT, pricedates TEXT,

area INTEGER)

''')

db.commit()

create\_rent\_main()

create\_rent\_current()

create\_current()

create\_no\_duplicates()

create\_viable()

**tables cleaner**

import sqlite3

import re

def addtofulltables():

print('started')

db = sqlite3.connect('dafthouses.db')

cursor = db.cursor()

current = cursor.execute('Select \* FROM currenthouses')

print('b')

currenthouses = []

for i in current:

a = 0

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'details': '',

'views': '',

'all\_prices': '',

'price\_dates': '',

'area': ''

}

for t in i:

if a is 1:hinfo['price']=t

elif a is 2: hinfo['address']=t

elif a is 3:hinfo['bedrooms'] = t

elif a is 4:hinfo['bathrooms'] = t

elif a is 5:hinfo['type'] = t

elif a is 6:hinfo['details'] = t

elif a is 7:hinfo['views'] = t

elif a is 8:hinfo['all\_prices'] = t

elif a is 9:hinfo['price\_dates'] = t

elif a is 10:hinfo['area'] = t

a+=1

currenthouses.append(hinfo)

viablehouses = []

allviable = cursor.execute('Select \* FROM mainhousestable')

print('c')

for i in allviable:

a = 0

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'details': '',

'views': '',

'all\_prices': '',

'price\_dates': '',

'area': ''

}

for t in i:

if a is 1:hinfo['price'] = t

elif a is 2:hinfo['address'] = t

elif a is 3:hinfo['bedrooms'] = t

elif a is 4:hinfo['bathrooms'] = t

elif a is 5:hinfo['type'] = t

elif a is 6:hinfo['details'] = t

elif a is 7:hinfo['views'] = t

elif a is 8:hinfo['all\_prices'] = t

elif a is 9:hinfo['price\_dates'] = t

elif a is 10:hinfo['area'] = t

a += 1

viablehouses.append(hinfo)

print('d')

current = currenthouses

for i in currenthouses:

# loop through the list and any values that are already in the viable list will not be added again

for viable in viablehouses:

if i['address'] == viable['address'] and i['price'] == viable['price']:

current.remove(i)

break

for house in current:

viablehouses.append(house)

cursor.execute('DELETE FROM mainhousestable')

for house in viablehouses:

cursor.execute('''INSERT INTO mainhousestable(price,address,bedrooms,bathrooms,type,details,views,allprices,pricedates,area)

VALUES(?,?,?,?,?,?,?,?,?,?)''', (house['price'], house['address'], house['bedrooms'],

house['bathrooms'], house['type'],

house['details'], house['views'],

house['all\_prices'], house['price\_dates'],

house['area']))

print('house entered')

db.commit()

**Wikipedea Scraper**

from bs4 import BeautifulSoup

import requests

import csv

from datetime import datetime

import re

def scrapedetails():

session = requests.Session()

session.headers.update({'User-Agent': 'Custom user agent'})

html =session.get('https://en.wikipedia.org/wiki/List\_of\_towns\_in\_the\_Republic\_of\_Ireland/2002\_Census\_Records')

html = html.text

soup = BeautifulSoup(html, 'html5lib')

my\_data = []

for tr in soup.find\_all('tr'):

my\_data.append(tr.text)

print(my\_data)

return my\_data

def clean\_data(mydata):

mylist = []

for town in mydata:

mytown = re.search('\\n(.\*),', town)

if mytown: mylist.append(mytown.group(1))

print(mylist)

return mylist

def scrapestreetdetails():

session = requests.Session()

session.headers.update({'User-Agent': 'Custom user agent'})

html =session.get('https://en.wikipedia.org/wiki/List\_of\_streets\_and\_squares\_in\_Dublin')

html = html.text

soup = BeautifulSoup(html, 'html5lib')

my\_data = []

for tr in soup.find\_all('tr'):

my\_data.append(tr.text)

print(my\_data)

return my\_data

def clean\_streetdata(mydata):

mylist = []

for town in mydata:

mytown = re.search('\\n(.\*)\\n', town)

if mytown: mylist.append(mytown.group(1))

print(mylist)

return mylist

clean\_streetdata(scrapestreetdetails())

**Classifier Main method**

import csv

import compassSplitter

import sectionStepper

import section\_splitting\_runprogram

import re

import point\_setter

from datetime import datetime

import coverageCompleter

from pathlib import Path

def open\_CorkCity():

#opens cork city and converts to a dictionary

with open('..\\..\\Database\\CorkCityHousesComplete.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'area': '',

'Garage' : '',

'Garden' : '',

'Ensuite': '',

'LatLng' : '',

'DateOfSale':'',

'SalePrice' :''

}

if i and a > 0:

hinfo['price'] = i[0]

hinfo['address'] = i[1]

hinfo['bedrooms'] = i[2]

hinfo['bathrooms'] = i[3]

hinfo['type'] = i[4]

hinfo['area'] = i[5]

hinfo['Garage'] = i[6]

hinfo['Garden'] = i[7]

hinfo['Ensuite'] = i[8]

hinfo['LatLng'] = i[9]

hinfo['DateOfSale'] = i[10]

hinfo['SalePrice'] = i[11]

all\_houses.append(hinfo)

a += 1

return all\_houses

def open\_Dublin15():

with open('..\\..\\Database\\Dublin15HousesComplete.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'area': '',

'Garage' : '',

'Garden' : '',

'Ensuite': '',

'LatLng' : '',

'DateOfSale':'',

'SalePrice' :''

}

if i and a > 0:

hinfo['price'] = i[0]

hinfo['address'] = i[1]

hinfo['bedrooms'] = i[2]

hinfo['bathrooms'] = i[3]

hinfo['type'] = i[4]

hinfo['area'] = i[5]

hinfo['Garage'] = i[6]

hinfo['Garden'] = i[7]

hinfo['Ensuite'] = i[8]

hinfo['LatLng'] = i[9]

hinfo['DateOfSale'] = i[10]

hinfo['SalePrice'] = i[11]

all\_houses.append(hinfo)

a += 1

return all\_houses

def split\_houseType(all\_houses,htype):

myusablehouses = []

# if the house type matches the passed htype it is appended

for house in all\_houses:

if str(htype) in str(house['type']):

myusablehouses.append(house)

return myusablehouses

def split\_Type\_detached(all\_houses,htype1,htype2):

myusablehouses = []

# splits detached as there are two spellings of detached/detatched

# if either is in the address they are appended

for house in all\_houses:

if str(htype1) in str(house['type']) or str(htype2) in str(house['type']):

myusablehouses.append(house)

return myusablehouses

def dublin\_outliers(newhouses):

cleanhouses = []

for house in newhouses:

# make sure the data is in dublin 15 by removing points outside a general area

# not very specific as borders can be inconsistent

# a property with dublin 15 in the address but a lat lng in dub4 for eg will cause major issues

# a property directly next to dub 15 will cause no issues

if house['LatLng'][0] < 53.45 and house['LatLng'][0] > 53.35:

if house['LatLng'][1] > -6.5 and house['LatLng'][1] < -6.3:

if float(house['price']) < float(5000000) and float(house['price']) > float(90000):

cleanhouses.append(house)

return cleanhouses

def cork\_outliers(newhouses):

# same with cork addresses

cleanhouses = []

for house in newhouses:

if house['LatLng'][0] < 52 and house['LatLng'][0] > 51.7:

if house['LatLng'][1] > -8.6 and house['LatLng'][1] < -8.4:

if float(house['price']) < float(5000000) and float(house['price']) > float(90000):

cleanhouses.append(house)

return cleanhouses

def run\_classifier(myusablehouses,htype,mainArea):

price = 100000

newhouses = []

for i in myusablehouses:

i['label'] = ''

if i['LatLng']:

# this converts the string of latlng to a list of latlng

i['LatLng'] = str(i['LatLng']).split(",")

i['LatLng'][0] = float(i['LatLng'][0].strip("["))

i['LatLng'][1] = float(i['LatLng'][1].strip("]"))

newhouses.append(i)

if 'Dublin'in mainArea:

cleanhouses = dublin\_outliers(newhouses)

else:

cleanhouses= cork\_outliers(newhouses)

#removes outliers

length = len(cleanhouses)

#sets the length for future use

mydict = {}

if length > 9:

# if length is less than 10 the algorithm will not be able to classiffy the houses so it is rejected

# point setter is called and variable initialised to the return values

borders , mynewhouses, gate,area,points = point\_setter.number\_of\_areas(cleanhouses,price,length)

# if the algorithm was able to classify the houses then borders will not be 0

if len(borders) > 0:

counts = {}

labels = ['1','2','3','4','5','6','7']

for l in labels:

counts[l] = 0

for house in mynewhouses:

# adds lat and long full values for graphing

house['Lattitude'] = house['LatLng'][0]

house['Longitude'] = house['LatLng'][1]

for l in labels:

if house['label'] ==l:

counts[l] +=1

# count the lables for future use

# set type and area for future use

# write to the files

mydict['type'] = htype

mydict['area'] = area

keys = myusablehouses[0].keys()

with open('..\\..\\Database\\'+str(mainArea)+str(htype)+'Complete.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(myusablehouses)

keys = borders[0].keys()

with open('..\\..\\Database\\'+str(mainArea)+str(htype)+'BorderPoints.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(borders)

print(points)

keys = points[0].keys()

with open('..\\..\\Database\\'+str(mainArea) + str(htype) + 'CentrePoints.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(points)

return mydict

st = datetime.now()

#################################

#RUN OFF EACH HOUSE TYPE HERE

type\_list =[ 'Terraced House', 'Apartment For Sale','Bungalow'

'Duplex' , 'End of T' , 'House For Sale' ,'New Dwelling' , 'Second-Hand' ,

'Site' , 'Terraced' 'Townhouse' ]

# for cork run through the type list and segregate each house type and attempt to classify the houses within

# semi-detached and detached are run seperatley to other due to spelling duplicates

area\_list=[]

all\_houses = open\_CorkCity()

for t in type\_list:

myusablehouses = split\_houseType(all\_houses,t)

if len(myusablehouses) > 10:

print(t)

area\_list.append(run\_classifier(myusablehouses,t.replace(' ','\_'),'CorkCity'))

myusablehouses = split\_Type\_detached(all\_houses,'Semi-detatched House','Semi-Detached')

if len(myusablehouses) > 10:

area\_list.append(run\_classifier(myusablehouses,'Semi-Detached','CorkCity'))

myusablehouses = split\_Type\_detached(all\_houses,'Detached','Detatched House')

if len(myusablehouses) > 10:

area\_list.append(run\_classifier(myusablehouses,'Detatched\_House','CorkCity'))

keys = area\_list[0].keys()

with open('..\\..\\Q\_Learning\_CSVs\\CorkCityAreaNumbers.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(area\_list)

type\_list2 =[ 'Terraced\_House', 'Apartment\_For\_Sale','Bungalow'

'Duplex' , 'End\_of\_T' , 'House\_For\_Sale' ,'New\_Dwelling' , 'Second-Hand' ,

'Site' , 'Terraced' 'Townhouse','Semi-Detached' ,'Detatched\_House']

# finish the coverage of any file that exists

for t in type\_list2:

my\_file = Path('..\\..\\Database\\CorkCity'+str(t)+'Complete.csv')

if my\_file.is\_file():

coverageCompleter.finish\_coverage(t, 'CorkCity')

# REPEAT FOR DUBLIN

##################################################

area\_list=[]

all\_houses = open\_Dublin15()

for t in type\_list:

myusablehouses = split\_houseType(all\_houses,t)

if len(myusablehouses) > 10:

area\_list.append(run\_classifier(myusablehouses,t.replace(' ','\_'),'Dublin15'))

myusablehouses = split\_Type\_detached(all\_houses,'Semi-detatched House','Semi-Detached')

if len(myusablehouses) > 10:

area\_list.append(run\_classifier(myusablehouses,'Semi-Detached','Dublin15'))

myusablehouses = split\_Type\_detached(all\_houses,'Detached','Detatched House')

if len(myusablehouses) > 10:

area\_list.append(run\_classifier(myusablehouses,'Detatched\_House','Dublin15'))

keys = area\_list[0].keys()

with open('..\\..\\Q\_Learning\_CSVs\\Dublin15AreaNumbers.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(area\_list)

type\_list2 =[ 'Terraced\_House', 'Apartment\_For\_Sale','Bungalow'

'Duplex' , 'End\_of\_T' , 'House\_For\_Sale' ,'New\_Dwelling' , 'Second-Hand' ,

'Site' , 'Terraced' 'Townhouse','Semi-Detached' ,'Detatched\_House']

for t in type\_list2:

my\_file = Path('..\\..\\Database\\Dublin15'+str(t)+'Complete.csv')

if my\_file.is\_file():

coverageCompleter.finish\_coverage(t,'Dublin15')

**claaifier main method**

import csv

import compassSplitter

import sectionStepper

import section\_splitting\_runprogram

import re

import point\_setter

from datetime import datetime

import coverageCompleter

from pathlib import Path

def open\_CorkCity():

#opens cork city and converts to a dictionary

with open('..\\..\\Database\\CorkCityHousesComplete.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'area': '',

'Garage' : '',

'Garden' : '',

'Ensuite': '',

'LatLng' : '',

'DateOfSale':'',

'SalePrice' :''

}

if i and a > 0:

hinfo['price'] = i[0]

hinfo['address'] = i[1]

hinfo['bedrooms'] = i[2]

hinfo['bathrooms'] = i[3]

hinfo['type'] = i[4]

hinfo['area'] = i[5]

hinfo['Garage'] = i[6]

hinfo['Garden'] = i[7]

hinfo['Ensuite'] = i[8]

hinfo['LatLng'] = i[9]

hinfo['DateOfSale'] = i[10]

hinfo['SalePrice'] = i[11]

all\_houses.append(hinfo)

a += 1

return all\_houses

def open\_Dublin15():

with open('..\\..\\Database\\Dublin15HousesComplete.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'area': '',

'Garage' : '',

'Garden' : '',

'Ensuite': '',

'LatLng' : '',

'DateOfSale':'',

'SalePrice' :''

}

if i and a > 0:

hinfo['price'] = i[0]

hinfo['address'] = i[1]

hinfo['bedrooms'] = i[2]

hinfo['bathrooms'] = i[3]

hinfo['type'] = i[4]

hinfo['area'] = i[5]

hinfo['Garage'] = i[6]

hinfo['Garden'] = i[7]

hinfo['Ensuite'] = i[8]

hinfo['LatLng'] = i[9]

hinfo['DateOfSale'] = i[10]

hinfo['SalePrice'] = i[11]

all\_houses.append(hinfo)

a += 1

return all\_houses

def split\_houseType(all\_houses,htype):

myusablehouses = []

# if the house type matches the passed htype it is appended

for house in all\_houses:

if str(htype) in str(house['type']):

myusablehouses.append(house)

return myusablehouses

def split\_Type\_detached(all\_houses,htype1,htype2):

myusablehouses = []

# splits detached as there are two spellings of detached/detatched

# if either is in the address they are appended

for house in all\_houses:

if str(htype1) in str(house['type']) or str(htype2) in str(house['type']):

myusablehouses.append(house)

return myusablehouses

def dublin\_outliers(newhouses):

cleanhouses = []

for house in newhouses:

# make sure the data is in dublin 15 by removing points outside a general area

# not very specific as borders can be inconsistent

# a property with dublin 15 in the address but a lat lng in dub4 for eg will cause major issues

# a property directly next to dub 15 will cause no issues

if house['LatLng'][0] < 53.45 and house['LatLng'][0] > 53.35:

if house['LatLng'][1] > -6.5 and house['LatLng'][1] < -6.3:

if float(house['price']) < float(5000000) and float(house['price']) > float(90000):

cleanhouses.append(house)

return cleanhouses

def cork\_outliers(newhouses):

# same with cork addresses

cleanhouses = []

for house in newhouses:

if house['LatLng'][0] < 52 and house['LatLng'][0] > 51.7:

if house['LatLng'][1] > -8.6 and house['LatLng'][1] < -8.4:

if float(house['price']) < float(5000000) and float(house['price']) > float(90000):

cleanhouses.append(house)

return cleanhouses

def run\_classifier(myusablehouses,htype,mainArea):

price = 100000

newhouses = []

for i in myusablehouses:

i['label'] = ''

if i['LatLng']:

# this converts the string of latlng to a list of latlng

i['LatLng'] = str(i['LatLng']).split(",")

i['LatLng'][0] = float(i['LatLng'][0].strip("["))

i['LatLng'][1] = float(i['LatLng'][1].strip("]"))

newhouses.append(i)

if 'Dublin'in mainArea:

cleanhouses = dublin\_outliers(newhouses)

else:

cleanhouses= cork\_outliers(newhouses)

#removes outliers

length = len(cleanhouses)

#sets the length for future use

mydict = {}

if length > 9:

# if length is less than 10 the algorithm will not be able to classiffy the houses so it is rejected

# point setter is called and variable initialised to the return values

borders , mynewhouses, gate,area,points = point\_setter.number\_of\_areas(cleanhouses,price,length)

# if the algorithm was able to classify the houses then borders will not be 0

if len(borders) > 0:

counts = {}

labels = ['1','2','3','4','5','6','7']

for l in labels:

counts[l] = 0

for house in mynewhouses:

# adds lat and long full values for graphing

house['Lattitude'] = house['LatLng'][0]

house['Longitude'] = house['LatLng'][1]

for l in labels:

if house['label'] ==l:

counts[l] +=1

# count the lables for future use

# set type and area for future use

# write to the files

mydict['type'] = htype

mydict['area'] = area

keys = myusablehouses[0].keys()

with open('..\\..\\Database\\'+str(mainArea)+str(htype)+'Complete.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(myusablehouses)

keys = borders[0].keys()

with open('..\\..\\Database\\'+str(mainArea)+str(htype)+'BorderPoints.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(borders)

print(points)

keys = points[0].keys()

with open('..\\..\\Database\\'+str(mainArea) + str(htype) + 'CentrePoints.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(points)

return mydict

st = datetime.now()

#################################

#RUN OFF EACH HOUSE TYPE HERE

type\_list =[ 'Terraced House', 'Apartment For Sale','Bungalow'

'Duplex' , 'End of T' , 'House For Sale' ,'New Dwelling' , 'Second-Hand' ,

'Site' , 'Terraced' 'Townhouse' ]

# for cork run through the type list and segregate each house type and attempt to classify the houses within

# semi-detached and detached are run seperatley to other due to spelling duplicates

area\_list=[]

all\_houses = open\_CorkCity()

for t in type\_list:

myusablehouses = split\_houseType(all\_houses,t)

if len(myusablehouses) > 10:

print(t)

area\_list.append(run\_classifier(myusablehouses,t.replace(' ','\_'),'CorkCity'))

myusablehouses = split\_Type\_detached(all\_houses,'Semi-detatched House','Semi-Detached')

if len(myusablehouses) > 10:

area\_list.append(run\_classifier(myusablehouses,'Semi-Detached','CorkCity'))

myusablehouses = split\_Type\_detached(all\_houses,'Detached','Detatched House')

if len(myusablehouses) > 10:

area\_list.append(run\_classifier(myusablehouses,'Detatched\_House','CorkCity'))

keys = area\_list[0].keys()

with open('..\\..\\Q\_Learning\_CSVs\\CorkCityAreaNumbers.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(area\_list)

type\_list2 =[ 'Terraced\_House', 'Apartment\_For\_Sale','Bungalow'

'Duplex' , 'End\_of\_T' , 'House\_For\_Sale' ,'New\_Dwelling' , 'Second-Hand' ,

'Site' , 'Terraced' 'Townhouse','Semi-Detached' ,'Detatched\_House']

# finish the coverage of any file that exists

for t in type\_list2:

my\_file = Path('..\\..\\Database\\CorkCity'+str(t)+'Complete.csv')

if my\_file.is\_file():

coverageCompleter.finish\_coverage(t, 'CorkCity')

# REPEAT FOR DUBLIN

##################################################

area\_list=[]

all\_houses = open\_Dublin15()

for t in type\_list:

myusablehouses = split\_houseType(all\_houses,t)

if len(myusablehouses) > 10:

area\_list.append(run\_classifier(myusablehouses,t.replace(' ','\_'),'Dublin15'))

myusablehouses = split\_Type\_detached(all\_houses,'Semi-detatched House','Semi-Detached')

if len(myusablehouses) > 10:

area\_list.append(run\_classifier(myusablehouses,'Semi-Detached','Dublin15'))

myusablehouses = split\_Type\_detached(all\_houses,'Detached','Detatched House')

if len(myusablehouses) > 10:

area\_list.append(run\_classifier(myusablehouses,'Detatched\_House','Dublin15'))

keys = area\_list[0].keys()

with open('..\\..\\Q\_Learning\_CSVs\\Dublin15AreaNumbers.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(area\_list)

type\_list2 =[ 'Terraced\_House', 'Apartment\_For\_Sale','Bungalow'

'Duplex' , 'End\_of\_T' , 'House\_For\_Sale' ,'New\_Dwelling' , 'Second-Hand' ,

'Site' , 'Terraced' 'Townhouse','Semi-Detached' ,'Detatched\_House']

for t in type\_list2:

my\_file = Path('..\\..\\Database\\Dublin15'+str(t)+'Complete.csv')

if my\_file.is\_file():

coverageCompleter.finish\_coverage(t,'Dublin15')

**compass splitter**

def splitter(property\_list,point) :

north =[]

south =[]

east = []

west = []

neast = []

nwest = []

swest = []

seast = []

thecentre = []

for prop in property\_list:

prop['Compass'] = ''

for p in property\_list:

try:

if p['LatLng'][0] > point[0] and p['LatLng'][1] == point[1]:

p['Compass'] = 'North'

north.append(p)

#print(len(north))

elif p['LatLng'][0] < point[0] and p['LatLng'][1] == point[1]:

p['Compass'] = 'South'

south.append(p)

elif p['LatLng'][0] == point[0] and p['LatLng'][1] > point[1]:

p['Compass'] = 'East'

east.append(p)

elif p['LatLng'][0] == point[0] and p['LatLng'][1] < point[1]:

p['Compass'] = 'West'

west.append(p)

elif p['LatLng'][0] > point[0] and p['LatLng'][1] > point[1]:

p['Compass'] = 'NorthEast'

neast.append(p)

elif p['LatLng'][0] < point[0] and p['LatLng'][1] > point[1]:

p['Compass'] = 'SouthEast'

seast.append(p)

elif p['LatLng'][0] > point[0] and p['LatLng'][1] < point[1]:

p['Compass'] = 'NorthWest'

nwest.append(p)

elif p['LatLng'][0] < point[0] and p['LatLng'][1] < point[1]:

p['Compass'] = 'SouthWest'

swest.append(p)

else:

p['Compass'] ='Unknown'

thecentre.append(p)

except:

print(property\_list)

return neast,nwest,seast,swest,north,south,east,west,thecentre

**coverage completer**

import csv

import section\_splitting\_runprogram

import math

from pathlib import Path

def import\_points(htype,area):

with open('..\\..\\Database\\'+str(area) + str(htype) + 'CentrePoints.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

## import points using the htype as a reference

a = 0

all\_points = []

for i in readCSV:

hinfo = {

'point': '',

'label': ''

}

if i and a > 0:

hinfo['point'] = i[0]

hinfo['label'] = i[1]

all\_points.append(hinfo)

a += 1

return all\_points

def import\_houses(htype,area):

with open('..\\..\\Database\\'+str(area)+str(htype)+'Complete.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'area': '',

'Garage': '',

'Garden': '',

'Ensuite': '',

'LatLng': '',

'DateOfSale': '',

'SalePrice': '',

'label':'',

'Latitude':'',

'Longitude':''

}

if i and a > 0:

hinfo['price'] = i[0]

hinfo['address'] = i[1]

hinfo['bedrooms'] = i[2]

hinfo['bathrooms'] = i[3]

hinfo['type'] = i[4]

hinfo['area'] = i[5]

hinfo['Garage'] = i[6]

hinfo['Garden'] = i[7]

hinfo['Ensuite'] = i[8]

hinfo['LatLng'] = i[9]

hinfo['DateOfSale'] = i[10]

hinfo['SalePrice'] = i[11]

hinfo['label'] = i[12]

hinfo['Latitude'] = i[13]

hinfo['Longitude'] = i[14]

all\_houses.append(hinfo)

a += 1

return all\_houses

def distance\_calculator(centre,point):

centre['point'] = str(centre['point']).split(",")

centre['point'][0] = float(centre['point'][0].strip("["))

centre['point'][1] = float(centre['point'][1].strip("]"))

# set the centre points lat and long

x = float(centre['point'][0])

y = float(centre['point'][1])

print(point)

point['LatLng'] = str(point['LatLng']).split(",")

point['LatLng'][0] = float(point['LatLng'][0].strip("["))

point['LatLng'][1] = float(point['LatLng'][1].strip("]"))

# set the points lat and long

x2 = float(point['LatLng'][0])

y2 = float(point['LatLng'][1])

# use the formula srt((x1-x2)^2+(y1-y2)^2) to calculate the distance between the points

mx = ((x2) - (x))

my = ((y2) - (y))

add = ((mx) + (my))

sq = ((add) \* (add))

distance = math.sqrt(sq)

return distance

def finish\_coverage(htype,area):

points = import\_points(htype,area)

houses = import\_houses(htype,area)

distance\_dict={}

# import the relevant data

for house in houses:

for p in points:

distance\_dict[str(p['label'])] = 100

# assign the distance as 100 as a default

if house['label'] == '':

for p in points:

# calculate the distance of the point from the centre

distance\_dict[str(p['label'])]=distance\_calculator(p,house)

lowest = distance\_dict['1']

newlabel = '1'

for p in points:

new = distance\_dict[str(p['label'])]

#iterate the list and if a distance is lower than the assigned it becomes the new lowest.

# set the new label

if new < lowest:

lowest = new

newlabel = str(p['label'])

#whatever the lowest is should be the newlabel

house['label'] = newlabel

**Point Setter**

import section\_splitting\_runprogram

def check\_coverage(coverage\_houses):

# checks how many houses have a label and returns percentage

count = 0

for house in coverage\_houses:

if house['label']:

#print(house)

count +=1

amount = len(coverage\_houses)

percent = count/amount

print('MY NEW %%%%%%%%%%%%%%%% COVERAGE')

print(percent)

return percent

def set\_point(set\_point\_houses,lowprice):

price = 99999999999

point = []

##initialise point to [] and price to 999999999999999999999

myhouse = ''

###loop through the houses

for house in set\_point\_houses:

#if a house has a label it is excluded

if house['label']: n = 'no'

else:

#otherwise it is tested to see if it fits between the prices

if float(house['price']) < float(price) and float(house['price']) > float(lowprice):

#price is set to the first house that is lower than 999999999 and greater than the lowprice

price = float(house['price'])

#point is also set to this house's point

point = house['LatLng']

myhouse = house

###if a house has been choosen as a viable house then its price is set to lowprice.

###this stops the program lloking for previous low priced houses!!!!

if myhouse:

lowprice = myhouse['price']

###returns both variabnles.

return point,lowprice

def test\_area(test\_allhouses,area,length):

####testing area to ensure it has a length over 8% of the total houses

count = 0

amount = length \*.08

for h in test\_allhouses:

if h['label'] == str(area):

count +=1

if count < amount:

check = False

else:

check = True

### if check is false reset all values the same as the area sent in as that area is invalid

# other previous areas are valid so dont reset all values!!!

if check == False:

for h in test\_allhouses:

if h['label'] == str(area):

h['label'] = ''

##return all houses and check

return check,test\_allhouses

def nintey\_percent\_runner(nhouses,gate,price,lowprice,borderpoints,length):

### run the splitter untill 90% coverage

# check coverage

points = []

perc = check\_coverage(nhouses)

area = 1

#set area and border and test

areatest = False

percentage = []

# while the percentage coverage is less than 85%

while perc < .85:

number = 0

#while area test is False \*\*\*\*\* this should stop areas with 0 from escaping!!!!!

while areatest == False:

print(number)

# set point and the lowprice

point,lowprice = set\_point(nhouses,lowprice)

# makesure there is a point to use as reference

if point:

#lowprice = float(point['price'])

##if there is a point of reference then run the section splitter running program.

##return borders and nhouses

newgate = gate

gatetest = False

########################################################################################################

while newgate < 2500000 and gatetest == False:

newborderpoints, nhouses = section\_splitting\_runprogram.run\_section\_splitter(nhouses, point,

price, str(area),

newgate,borderpoints)

gatetest,ngate = test\_area(nhouses,area,length)

newgate+= 5000

##test the area

for pt in newborderpoints:

borderpoints.append(pt)

areatest, nhouses = test\_area(nhouses, area,length)

#otherwise escape the entire loop as no points are left

else:

print('AREA IS 12')

# not ideal but effective way of escaping the while areatest loop

area = 12

areatest = True

number+=1

#number is the label and is incremented after each new label is set

areatest = False

lowprice = 0

percentage.append(check\_coverage(nhouses))

perc = check\_coverage((nhouses))

my\_dict = {}

#set my dict params for future use

my\_dict['point'] = point

my\_dict['label'] = area

points.append(my\_dict)

area += 1

if area > 10:

perc = 1.1

#escape the loop as 10 areas is too big

if not percentage:

percentage.append(check\_coverage(nhouses))

print(percentage)

return area,borderpoints,nhouses,percentage[0],points

def checknumber(area):

if area < 11 and area >3:

check = True

else: check = False

# check if area numbers is acceptable

return check

def opening\_gate(gate,houses,lowprice,price,borderpoints,length):

# set the opening gate

newgate = gate

area = 1

gatetest = False

point, lowprice = set\_point(houses, lowprice)

# run the loop through the classifier untill the classifier returns a viable gate

while newgate < 700000 and gatetest == False:

newborderpoints, nhouses = section\_splitting\_runprogram.run\_section\_splitter(houses, point,

price, str(area), newgate,

borderpoints)

gatetest, ngate = test\_area(nhouses, area,length)

newgate += 5000

return newgate

def new\_gate(gate,houses,lowprice,price,borderpoints,length,oldcoverage):

newgate = gate

area = 1

gatetest = False

coverage = 0

# ad .001 to ensure an increase in the coverage

oldcoverage = oldcoverage + .001

print(oldcoverage)

print('Outside')

point, lowprice = set\_point(houses, lowprice)

while gatetest == False or coverage <= oldcoverage :

# run through classier in loops untill a new larger coverage is found in comparison to the last one

for h in houses:

h['label'] = ''

newborderpoints, nhouses = section\_splitting\_runprogram.run\_section\_splitter(houses, point,

price, str(area), newgate,

borderpoints)

gatetest, ngate = test\_area(nhouses, area,length)

newgate += 5000

coverage = check\_coverage(nhouses)

print('HERE')

print(newgate)

print(coverage)

# problem with recurring rerunning of the loop had to be stopped by using this coverage > .9

if coverage > .9:

break

return newgate,coverage

def number\_of\_areas(newhouses,price,length):

print(length)

gate = opening\_gate(0,newhouses,0,price,[],length)

#assign the gate

percentage = 0

check = False

while check == False:

# loop while the check for the area numbers is false

lowprice = 0

for h in newhouses:

h['label'] = ''

#assign/reset label to null

borders =[]

gate,p = new\_gate(0, newhouses, 0, price, [], length, percentage)

if p > .9:

break

# second break for over 90% as a safety. Will return no borders so no values are saved

area,borders,mynewhouses,percentage,points = nintey\_percent\_runner(newhouses, gate, price,lowprice,borders,length)

check = checknumber(area)

print('PERCENTAGE',percentage)

print('a')

print(gate)

print('FFFFFFFFFFFFFFFFFIIIIIIIIIIIIIIIIIIIIIIIIIINNNNNNNNNNNNNNNNNNNAAAAAAAAAAAAAAAAAAAAAAAALLLLLLLLLLLLLLLLLLLLLL')

return borders,mynewhouses,gate,area,points

**section splitting run program**

import csv

import compassSplitter

import sectionStepper

def run\_section\_splitter(newhouses,point,price,label,gate,borderpoints):

##send in all border points

#borderpoints is done...

##send to a border points function

##return each compass of border points.

##send with 'SouthEast' etc.

north\_eastb,north\_westb,south\_eastb,south\_westb,northb,southb,eastb,westb,the\_centreb = compassSplitter.splitter(borderpoints,point)

north\_east,north\_west,south\_east,south\_west,north,south,east,west,the\_centre = compassSplitter.splitter(newhouses,point)

## split the houses into nsew etc

# send each set of properties into the classifier with the relevant string

south\_east\_endpoints,south\_east\_points = sectionStepper.step\_Section('SouthEast',point,south\_east,price,label,gate,south\_eastb)

north\_east\_endpoints, north\_east\_points= sectionStepper.step\_Section('NorthEast',point,north\_east,price,label,gate,north\_eastb)

north\_west\_endpoints,north\_west\_points = sectionStepper.step\_Section('NorthWest', point, north\_west, price, label,gate,north\_westb)

south\_west\_endpoints,south\_west\_points = sectionStepper.step\_Section('SouthWest', point, south\_west, price, label,gate,south\_westb)

north\_endpoints,north\_points = sectionStepper.step\_nsew(point,north,price,label,gate,'north',northb)

south\_endpoints, south\_points = sectionStepper.step\_nsew(point, south, price, label, gate, 'south',southb)

east\_endpoints, east\_points = sectionStepper.step\_nsew(point, east, price, label, gate, 'east',eastb)

west\_endpoints, west\_points = sectionStepper.step\_nsew(point, west, price, label, gate, 'west',westb)

# if the point has a distance add it to the section points

section\_points = []

for pt in south\_east\_endpoints:

if pt['Distance']:

section\_points.append(pt)

for pt in north\_east\_endpoints:

if pt['Distance']:

section\_points.append(pt)

for pt in north\_west\_endpoints:

if pt['Distance']:

section\_points.append(pt)

for pt in south\_west\_endpoints:

if pt['Distance']:

section\_points.append(pt)

for pt in south\_endpoints:

if pt['Distance']:

section\_points.append(pt)

for pt in north\_endpoints:

if pt['Distance']:

section\_points.append(pt)

for pt in east\_endpoints:

if pt['Distance']:

section\_points.append(pt)

for pt in west\_endpoints:

if pt['Distance']:

section\_points.append(pt)

# print(section\_points)

#append all the houses from the subsections to the full list.

all\_houses = []

for house in south\_east\_points:

all\_houses.append(house)

for house in south\_west\_points:

all\_houses.append(house)

for house in north\_east\_points:

all\_houses.append(house)

for house in north\_west\_points:

all\_houses.append(house)

for house in north\_points:

all\_houses.append(house)

for house in west\_points:

all\_houses.append(house)

for house in east\_points:

all\_houses.append(house)

for house in south\_points:

all\_houses.append(house)

for house in the\_centre:

all\_houses.append(house)

count = 0

# remove compas section and distance to prevent crossover issues

for house in all\_houses:

try:

del house['Compass']

except:

a=1

try:

del house['Section']

except:

a = 2

try:

del house['Distance']

except:

a=3

if house['label']:

count+= 1

###returns wrong amount of houses

return section\_points,all\_houses

**section splitter**

import math

def plot\_sub\_sections(point, section\_type):

## Tan(9degrees)

# clock of degrees

if section\_type is 'NorthEast':

degree = 9

elif section\_type is 'SouthEast':

degree = 99

elif section\_type is 'SouthWest':

degree = 189

else: degree = 279

sect = ['one','two','three','four','five','six','seven','eight','nine','ten']

sub\_sections = {}

## ploty = mx - b

#degree = 9

for s in sect:

slope = (math.tan(math.radians(degree)))

mx = (slope \* point[0])

b = point[1] - mx

sub\_sections[s] = [slope, b]

degree +=9

return sub\_sections

def split\_section(section\_type, point, section\_points):

sub\_sections = plot\_sub\_sections(point,section\_type)

return sub\_sections

**section stepper**

import sectionSplitter

import math

import operator

def order\_points(point,points):

for p in points:

p['Distance'] = 0

# calculate the distance using sqrt(x-x^2 + y-y^2)

x = point[0]

y = point[1]

for p in points:

x2 = p['LatLng'][0]

y2 = p['LatLng'][1]

mx = ((x2) - (x))

my = ((y2) - (y))

add = ((mx) + (my))

sq = ((add) \* (add))

distance = math.sqrt(sq)

p['Distance'] = distance

return points

def claasify\_points\_section(points, sub\_sections,section\_type) :

for p in points:

p['Section'] = ''

# plot point into y = mx + b

for key, section in sub\_sections.items():

for point in points:

x = point['LatLng'][0]

y = point['LatLng'][1]

m = sub\_sections[str(key)][0]

b = sub\_sections[str(key)][1]

# use y = mx+b formula

mx = m\*x

test = (mx + b) - (y)

if point['Section'] == '':

if section\_type is 'SouthEast' or 'NorthEast':

# if the section is southeast or northeast then the point is in this section

# if it is on the line-test is 0

# or

# if test is greater than zero

# the point is on the right of the line

# meaning it is in this section.

if test is 0:

point['Section'] = key

elif test > 0:

point['Section'] = key

else:

# otherwise it is the opposite

if test is 0:

point['Section'] = key

elif test < 0:

point['Section'] = key

for p in points:

if p['Section'] == '':

p['Section'] = 'Unknown'

return points

def step\_through\_nsew\_points(points, price, label, gate, point, compass, borders):

# step through points and assign lable where necessary:

point\_dict = {}

point\_dict[compass] = []

# use the gate to assign a small and big point for reference

small = price - gate

big = price + gate

# sort the points and borders by their distance

points.sort(key=operator.itemgetter('Distance'))

borders.sort(key=operator.itemgetter('Distance'))

endpointa = ''

if borders:

endpointa = borders[0]

# the closest border point for this section is used as an end point reference

end\_points =[]

end\_points.append({'Label': label, 'Compass': compass, 'Section': compass, 'Distance': 0})

counter = 0

endpoint = ''

for p in points:

# for all points

if endpointa:

# if the endpoint a was set continue

# IF A

if p['Distance'] < endpointa['Distance']:

# IF B

# if the distance of the point is less than the distance of endpoint a continue.

# distance is saved as avalue and represents the distance between p and endpointa

if float(p['price']) > small and float(p['price']) < big:

# if the price is within the gate

if p['label'] == '':

# if the point has no label set

p['label'] = label

# assign a label

counter = 0

# put counter as 0

else:

# if the counter is already one then the border has been found so set the p as the endpoint

if counter == 1:

endpoint = p

else:

# otherwise set the tempoint and increment counter

temppoint = p

# temp point is unused and only declared as a demonstration

# this point gets missed here but is assigned later in another function

counter += 1

# if endpoint the set enpoints lat and lang

if endpoint:

end\_points[0]['LatLng'] = endpoint['LatLng']

end\_points[0]['Distance'] = endpoint['Distance']

break

else:

#If B

if endpoint:

# if endpoint the set enpoints lat and lang

end\_points[0]['LatLng'] = endpoint['LatLng']

end\_points[0]['Distance'] = endpoint['Distance']

break

else:

endpoint = endpointa

# otherwise set the enpoints 0 to endpointa

end\_points[0]['LatLng'] = endpointa['LatLng']

end\_points[0]['LatLng'] = endpointa['LatLng']

break

# then break as all further points are outside the borderpoint

else:

# If A

# otherwise if there is no border do the following

# repeat of the above loops

if float(p['price']) > small and float(p['price']) < big:

if p['label'] == '':

p['label'] = label

counter = 0

else:

if counter == 1:

endpoint = p

else:

temppoint = p

counter += 1

if endpoint:

end\_points[0]['LatLng'] = endpoint['LatLng']

end\_points[0]['Distance'] = endpoint['Distance']

break

return end\_points

def step\_through\_points(points, price, label, gate, point, compass, borders):

# step through points and assign lable where necessary:

point\_dict = {}

#print('@STEPPING POINTS')

###new dict of lists

#################needs reviewing

####Possibly setting the wrong distances

for p in points:

point\_dict[str(p['Section'])]=[]

for p in points:

point\_dict[p['Section']].append(p)

for keys,vale in point\_dict.items():

point\_dict[keys].sort(key=operator.itemgetter('Distance'))

the\_border\_points = {}

# sort by distance

big = price + gate

small = price - gate

# set the gate

sect = ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten','Unknown']

for s in sect:

the\_border\_points[s] = []

for b in borders:

the\_border\_points[b['Section']].append(b)

###put the other borderpoints into a list for each section to make it easier to eliminate ones too far away.

for s in sect:

for key,value in the\_border\_points.items():

from operator import itemgetter

the\_border\_points[key] = sorted(value, key=itemgetter('Distance'))

# sort each dictionary of points based off of distance

if len(the\_border\_points[key]) > 0:

the\_border\_points[key] = the\_border\_points[key][0:1]

# set the border points list is longer than 0 then slice off the positions above 0.

end\_points =[]

###set a distance for each endpoint

for i in sect:

# create reference variables for later!

end\_points.append({'Label':label,'Compass':compass,'Section':i,'Distance':0,'Point':point})

# iter through the dictionary

for key, value in point\_dict.items():

# BIGGEST DIFFERENCE HERE

# iterate through the the point-dict

# This repeats the previous functions transactions on each section instead of just nsew

counter = 0

endpoint = ''

# iter through the list at dict point

if the\_border\_points[key]:

for p in point\_dict[key]:

# check to see if this point fits the criteria

if p['Distance'] < the\_border\_points[key][0]['Distance']:

if float(p['price']) >small and float(p['price']) < big:

if p['label'] == '':

p['label'] = label

##assign a label

counter = 0

else:

if counter == 1:

endpoint = p

else:

temppoint = p

counter+=1

if endpoint:

for pt in end\_points:

if pt['Section'] == key:

pt['LatLng'] = endpoint['LatLng']

pt['Distance'] = endpoint['Distance']

break

else:

if endpoint:

break

else:

endpoint = the\_border\_points[key][0]

for pt in end\_points:

if pt['Section'] == key:

pt['LatLng'] = endpoint['LatLng']

pt['Distance'] = endpoint['Distance']

break

else:

for p in point\_dict[key]:

# check to see if this point fits the criteria

if float(p['price']) > small and float(p['price']) < big:

if p['label'] == '':

p['label'] = label

##assign a label

counter = 0

else:

if counter == 1:

endpoint = p

else:

temppoint = p

counter += 1

if endpoint:

for pt in end\_points:

if pt['Section'] == key:

pt['LatLng'] = endpoint['LatLng']

pt['Distance'] = endpoint['Distance']

break

#ordered\_end\_points = order\_points(point,end\_points)

return end\_points

def checkpoints(orderedpoints,endpoints,label):

# thi sets the missed point tempoint

for point in orderedpoints:

####here is wherer i had new label setting

try:

if point['Distance'] < endpoints[point['Section']]['Distance']:

if point['label'] == '':

point['label'] = label

except:

#print(point['Distance'])

#print(endpoints[point['Section']])

a = 1

return orderedpoints

def get\_all\_borders(section\_points):

all\_borders=[]

for house in section\_points:

if house['label']:

all\_borders.append(house)

return all\_borders

def step\_Section(section\_type,point,section\_points,price,label,gate,borders ):

## recieve borderpoints

all\_borders = get\_all\_borders(section\_points)

##split into sections

##get the closest for each section.

sub\_sections = sectionSplitter.split\_section(section\_type,point,section\_points)

if all\_borders:

all\_borders = claasify\_points\_section(all\_borders,sub\_sections,section\_type)

all\_borders = order\_points(point,all\_borders)

section\_points = claasify\_points\_section(section\_points,sub\_sections,section\_type)

ordered\_points = order\_points(point,section\_points)

#Actually Step through the points!!!!!

###needs new input

####pass the borderpoints to the function

##use the border points to ensure that the function does not pass over the points of another area

endpoints = step\_through\_points(ordered\_points,price,label,gate,point,section\_type,all\_borders)

##endpoints is a list only

section\_points = checkpoints(ordered\_points, endpoints,label)

#border\_equations = plot\_border(point,endpoints,section\_type)

return endpoints,section\_points

def step\_nsew(point,section\_points,price,label,gate,compass,borders):

if borders:

borders = order\_points(point, borders)

ordered\_points = order\_points(point,section\_points)

endpoints = step\_through\_nsew\_points(ordered\_points, price, label, gate, point,compass,borders)

section\_points = checkpoints(ordered\_points, endpoints, label)

return endpoints,section\_points

###order the points

# root[(x2-x1)sq + (y2-y1)sq]

# step each section

**Actor**

import initialise\_Q\_enviroment as envinit

import step\_chooser

import q\_value\_handler

import time

def get\_predictions(environment,oldposition):

old\_pos = 0

position = oldposition

loopexit = False

x = 0

while not loopexit:

# while the loopexit is not true the loop keeps chhoosing a new step

new\_pos = step\_chooser.choose\_next\_step(position,environment)

# if ever the loop rebounds it is then stuck and must be exited imediatley

if new\_pos[2] == old\_pos:

loopexit = True

position = 0

else:

old\_pos = position

loopexit = new\_pos[3]

position = new\_pos[2]

x+=1

# if the loop goes above 2000 it is likley to be caught in a constant and must be exited

if x > 2000:

loopexit=True

position = 0

return position

def update\_all\_qvalues(environment,oldposition):

#env = envinit.set\_exit\_points()

totaltest = True

needstraining = True

while needstraining:

position = oldposition

loopexit = False

trainingcheck = True

x = 0

while not loopexit:

x+=1

new\_pos = step\_chooser.choose\_next\_step(position,environment)

newmax = step\_chooser.get\_max(new\_pos[2],environment)

Q = new\_pos[0]

if Q > 10:

needstraining = False

loopexit = new\_pos[3]

for e in environment:

if position in e['5KStep'].keys():

if loopexit == False:

# fixing a bug for long floats

newq = float("{0:.5f}".format(q\_value\_handler.calculate\_new\_Q(Q,newmax)))

if Q == newq:

a=1

if str(Q) != str(newq):

trainingcheck = False

totaltest = False

e['5KStep'][position]['Actions'][new\_pos[1]]['Q'] = newq

if x > 5000:

loopexit = True

position = new\_pos[2]

if trainingcheck:

needstraining = False

return environment,totaltest

def enter\_environment(exit\_points ):

environment = envinit.set\_exit\_points(exit\_points)

# set the exit points

totaltest = False

while not totaltest:

# total test is designed to keep the loop continuing

# only a complete run with no updates will make it return true

position = 100000

position2 = 195000

for lp in range(99):

environment,totaltest = update\_all\_qvalues(environment, position)

if totaltest:

# if true it will be retested

environment, totaltest = update\_all\_qvalues(environment, position2)

else:

# if false a seperate value will be used as a false on the first

# means the loop should not break

environment, totaltest2 = update\_all\_qvalues(environment, position2)

position += 100000

position2 += 100000

return environment

**check data maker file**

import csv

with open('C:\\Users\\Cahil\\Desktop\\Database\\Dublin15HousesComplete.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'views': '',

'area': '',

'Garage': '',

'Garden': '',

'Ensuite': '',

'LatLng': '',

'dateofsale': ''

}

if i and a > 0:

hinfo['price'] = i[0]

hinfo['address'] = i[1]

hinfo['bedrooms'] = i[2]

hinfo['bathrooms'] = i[3]

hinfo['type'] = i[4]

hinfo['area'] = i[6]

hinfo['Garage'] = i[7]

hinfo['Garden'] = i[8]

hinfo['Ensuite'] = i[9]

hinfo['LatLng'] = i[10]

hinfo['dateofsale'] = i[11]

all\_houses.append(hinfo)

a += 1

myusablehouses = []

for house in all\_houses:

if 'Semi-Detached House' in str(house['type']):

if house['dateofsale']:

myusablehouses.append(house)

print(len(myusablehouses))

**Data Importer**

import csv

def splitHouses(all\_houses):

DetHouse = []

SemDetHouse = []

TerrHouse = []

AppSale = []

Bungalow = []

DuplexSale = []

EndTHouse = []

NewDHA = []

SHDHA = []

SiteSale = []

Townhouse = []

HouseSale = []

OtherType = []

for house in all\_houses:

if 'Detatched House' in house['type']:

DetHouse.append(house)

elif 'Semi-detatched House' in house['type']:

SemDetHouse.append(house)

elif 'Terraced House' in house['type']:

TerrHouse.append(house)

elif 'Apartment For Sale' in house['type']:

AppSale.append(house)

elif 'Bungalow' in house['type']:

Bungalow.append(house)

elif 'Semi-Detached' in house['type']:

SemDetHouse.append(house)

elif 'Detached' in house['type']:

DetHouse.append(house)

elif 'Duplex' in house['type']:

DuplexSale.append(house)

elif 'End of' in house['type']:

EndTHouse.append(house)

elif 'House For Sale' in house['type']:

HouseSale.append(house)

elif 'New Dwelling' in house['type']:

NewDHA.append(house)

elif 'Second-Hand' in house['type']:

SHDHA.append(house)

elif 'Site' in house['type']:

SiteSale.append(house)

elif 'Terraced' in house['type']:

TerrHouse.append(house)

elif 'Townhouse' in house['type']:

Townhouse.append(house)

else:

OtherType.append(house)

Houses = {}

Houses['DetHouse'] = []

Houses['DetHouse'].append(DetHouse)

Houses['DetHouse'].append('DetachedHouse')

Houses['SemDetHouse']= []

Houses['SemDetHouse'].append(SemDetHouse)

Houses['SemDetHouse'].append('SemiDetachedHouse')

Houses['TerrHouse']=[]

Houses['TerrHouse'].append(TerrHouse)

Houses['TerrHouse'].append('TerracedHouse')

Houses['AppSale'] = []

Houses['AppSale'].append(AppSale)

Houses['AppSale'].append('Apartment')

Houses['Bungalow'] = []

Houses['Bungalow'].append(Bungalow)

Houses['Bungalow'].append('Bungalow')

Houses['Duplex'] = []

Houses['Duplex'].append(DuplexSale)

Houses['Duplex'].append('Duplex')

Houses['EndTHouse'] = []

Houses['EndTHouse'].append(EndTHouse)

Houses['EndTHouse'].append('EndOfTerraceHouse')

Houses['NewDHA'] = []

Houses['NewDHA'].append(NewDHA)

Houses['NewDHA'].append('NewDHA')

Houses['SHDHA'] = []

Houses['SHDHA'].append(SHDHA)

Houses['SHDHA'].append('SHDHA')

Houses['Site'] = []

Houses['Site'].append(SiteSale)

Houses['Site'].append('Site')

Houses['TownHouse'] = []

Houses['TownHouse'].append(Townhouse)

Houses['TownHouse'].append('TownHouse')

Houses['House'] = []

Houses['House'].append(HouseSale)

Houses['House'].append('House')

return Houses

def import\_Dublin(htype):

with open('..\\..\\Database\\Dublin15'+str(htype)+'Complete.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'area': '',

'Garage': '',

'Garden': '',

'Ensuite': '',

'LatLng': '',

'DateOfSale': '',

'SalePrice': '',

'label' : ''

}

if i and a > 0:

hinfo['price'] = i[0]

hinfo['address'] = i[1]

hinfo['bedrooms'] = i[2]

hinfo['bathrooms'] = i[3]

hinfo['type'] = i[4]

hinfo['area'] = i[5]

hinfo['Garage'] = i[6]

hinfo['Garden'] = i[7]

hinfo['Ensuite'] = i[8]

hinfo['LatLng'] = i[9]

hinfo['DateOfSale'] = i[10]

hinfo['SalePrice'] = i[11]

hinfo['label'] = i[12]

all\_houses.append(hinfo)

a += 1

new\_houses=[]

for house in all\_houses:

if house['label']:

new\_houses.append(house)

return new\_houses

def import\_Cork(htype):

with open('..\\..\\Database\\CorkCity' + str(htype) + 'Complete.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'area': '',

'Garage': '',

'Garden': '',

'Ensuite': '',

'LatLng': '',

'DateOfSale': '',

'SalePrice': '',

'label': ''

}

if i and a > 0:

hinfo['price'] = i[0]

hinfo['address'] = i[1]

hinfo['bedrooms'] = i[2]

hinfo['bathrooms'] = i[3]

hinfo['type'] = i[4]

hinfo['area'] = i[5]

hinfo['Garage'] = i[6]

hinfo['Garden'] = i[7]

hinfo['Ensuite'] = i[8]

hinfo['LatLng'] = i[9]

hinfo['DateOfSale'] = i[10]

hinfo['SalePrice'] = i[11]

hinfo['label'] = i[12]

all\_houses.append(hinfo)

a += 1

new\_houses = []

for house in all\_houses:

if house['label']:

new\_houses.append(house)

return new\_houses

**Data Segmentation**

import DataImporter

import pandas as pd

from pathlib import Path

import random

import csv

def split\_data(houses,htype,areas):

lenght = len(houses)

twenty = []

eighty = lenght\*.8

# pop twenty percent into a new file

while lenght > eighty:

twenty.append(houses.pop(random.randint(0, lenght-1)))

lenght -= 1

print(len(houses))

print(len(twenty))

#save to a new file

keys = twenty[0].keys()

with open('..\\..\\Q\_Learning\_CSVs\\'+str(areas)+str(htype)+'Twenty.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(twenty)

return houses

def create\_segment(htype):

housesegment = {}

housesegment['positions']={}

# create a total segment of bedrooms bathrooms and area inside of the Location

# It is already a specific type

for i in range(1, 11):

for x in range(1, 11):

for y in range(1,11):

###################################################################

pos = 'bd' + str(i) + 'br' + str(x) + 'area' + str(y)

housesegment['positions'][pos] = {}

housesegment['positions'][pos]['houses'] = []

housesegment['positions'][pos]['bedrooms'] = i

housesegment['positions'][pos]['bathrooms'] = x

housesegment['positions'][pos]['area'] = y

housesegment['type'] = htype

return housesegment

def segment\_thedata(all\_houses,htype,area):

housesegment = create\_segment(htype)

for i in range(1,11):

for x in range(1,11):

for y in range(1,11):

###################################################################

pos = 'bd'+str(i)+'br'+str(x)+'area'+str(y)

for house in all\_houses:

if int(house['bedrooms']) == i:

if int(house['bathrooms']) ==x:

if int(house['label'])==y:

housesegment['positions'][pos]['houses'].append(house['price'])

housesegment['positions'][pos]['area'] = area

housesegment['positions'][pos]['type']=htype

# aassign correct data for future use and append relevant properties

count = 0

for i in range(1, 11):

for x in range(1, 11):

for y in range(1, 11):

###################################################################

pos = 'bd' + str(i) + 'br' + str(x) + 'area' + str(y)

if len(housesegment['positions'][pos]['houses']) < 1:

del(housesegment['positions'][pos])

# over 20,000 combos

# deletes any without instances.

# could be a vast amount

return housesegment

def import\_then\_segment():

type\_list1 = ['Terraced\_House', 'Apartment\_For\_Sale', 'Bungalow'

'Duplex', 'End\_of\_T', 'House\_For\_Sale', 'New\_Dwelling',

'Second-Hand',

'Site', 'Terraced' 'Townhouse', 'Detatched\_House', 'Semi-Detached']

type\_list2 = [ 'Semi-Detached']

all\_segments =[]

for t in type\_list1:

my\_file = Path('..\\..\\Database\\CorkCity' + str(t) + 'Complete.csv')

if my\_file.is\_file():

houses = DataImporter.import\_Cork(t)

houses = split\_data(houses, t,'CorkCity')

all\_segments.append(segment\_thedata(houses, t, 'CorkCity'))

for t in type\_list1:

my\_file = Path('..\\..\\Database\\Dublin15' + str(t) + 'Complete.csv')

if my\_file.is\_file():

houses = DataImporter.import\_Dublin(t)

houses = split\_data(houses,t,'Dublin15')

all\_segments.append(segment\_thedata(houses, t,'Dublin15'))

return all\_segments

#dublin\_houses = DataImporter.splitHouses(dublin\_houses)

**Initialise Q environment**

import csv

def createDict():

financial\_list = []

value = 100000

for lp in range(99):

financial\_list.append({'Value':value})

value+= 100000

# initialise a 2d array of increments of 5000 on the x and 100,000 on the y

for dic in financial\_list:

dic['5KStep']={}

val = dic['Value']

actions = []

for lp in range(20):

plus5 =val + 5000

minus5 = val - 5000

plus100 = val +100000

minus100 = val - 100000

if plus5 > 9995000 or plus5 < 100000:

plus5 = val

if minus5 > 9995000 or minus5 < 100000:

minus5 = val

if plus100 > 9995000 or plus100 < 100000:

plus100 = val

if minus100 > 9995000 or minus100 < 100000:

minus100 = val

# each has an up a down a left and a right

#up = -100k

# down = +100k

# left = minus 5k

# right = plus 5k

# if at an edge the choice rebounds

# hence the if statement

dic['5KStep'][val] = {}

dic['5KStep'][val]['Actions']={'Minus5k':{'Q':0,'ActionVal':minus5,'Exit':False},'Plus5k':{'Q':0,'ActionVal':plus5,'Exit':False},

'Minus100k':{'Q':0,'ActionVal':minus100,'Exit':False},

'Plus100k':{'Q':0,'ActionVal':plus100,'Exit':False}}

val+= 5000

return financial\_list

def set\_exit\_points(exits):

environment = createDict()

position = 100000

# get the details from a house entery method.

#####all next steps containing the action val ex

for env in environment:

# loop down te list and check if the list contains a dict with the exit number

for ex in exits:

ex = int(ex)

if ex in env['5KStep'].keys():

print('INENV')

env['5KStep'][ex]['Exit'] = True

steps = env['5KStep'].keys()

for step in steps:

# if it dows then set all steps towards that exit number as exits

# and the Q value to 1

if ex == env['5KStep'][step]['Actions']['Minus5k']['ActionVal']:

env['5KStep'][step]['Actions']['Minus5k']['Q'] = 1

env['5KStep'][step]['Actions']['Minus5k']['Exit'] = True

if ex == env['5KStep'][step]['Actions']['Plus5k']['ActionVal']:

env['5KStep'][step]['Actions']['Plus5k']['Q'] = 1

env['5KStep'][step]['Actions']['Plus5k']['Exit'] = True

if ex == env['5KStep'][step]['Actions']['Minus100k']['ActionVal']:

env['5KStep'][step]['Actions']['Minus100k']['Q'] = 1

env['5KStep'][step]['Actions']['Minus100k']['Exit'] = True

if ex == env['5KStep'][step]['Actions']['Plus100k']['ActionVal']:

env['5KStep'][step]['Actions']['Plus100k']['Q'] = 1

env['5KStep'][step]['Actions']['Plus100k']['Exit'] = True

return environment

**q value handler**

import time

def calculate\_new\_Q(Q,max):

#Q+= alpha[r + (beta.max)-Q]

alpha = .75

gamma = .85

#betamax = beta.max

betamax = gamma\*max

r = (-.04)

# betamaxQ = (beta.max)-Q

betamaxQ = betamax - Q

# maxandr = r + (beta.max)-Q

maxandr = (r) + betamaxQ

# alphamax = alpha[r + (beta.max)-Q]

alphamax = maxandr \* alpha

Q += alphamax

return Q

#def find\_max(position,env):

**Run q learning**

import DataSegmentation

import actor

from datetime import datetime

import collections

import csv

start = datetime.now()

x = 0

all\_segments = DataSegmentation.import\_then\_segment()

env\_list =[]

# create segments and append to a full environment list

# save extra params for future testing

for s in all\_segments:

for p in s['positions']:

exit\_points = []

my\_dict ={}

for price in s['positions'][p]['houses']:

exit\_points.append(float(price))

print('before')

my\_dict['pos'] = p

my\_dict['area'] = s['positions'][p]['area']

my\_dict['type'] = s['positions'][p]['type']

# enter the environment for each env

my\_dict['env'] = actor.enter\_environment(exit\_points)

env\_list.append(my\_dict)

x+=1

mycounter = []

print('Predicting')

for e in env\_list:

print(1)

my\_dict={}

counter ={}

predictions =[]

position = 100000

position2 = 195000

for lp in range(99):

# Make predictios and add the necessary values to a list opf dicts

# for writing to a file.

predictions.append(actor.get\_predictions(e['env'], position))

predictions.append(actor.get\_predictions(e['env'], position2))

position += 100000

position2 += 100000

my\_dict['pos'] = e['pos']

my\_dict['area'] = e['area']

my\_dict['type'] = e['type']

for p in predictions:

counter[str(p)]=0

counter[str(0)]=0

for p in predictions:

counter[str(p)]+=1

del counter[str(0)]

my\_dict['counter'] = str(counter)

mycounter.append(my\_dict)

keys = mycounter[0].keys()

with open('..\\..\\Q\_Learning\_CSVs\\Predictions.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(mycounter)

print(mycounter)

end = datetime.now()-start

print(end)

**step chooser**

import operator

import random

def choose\_next\_step(position,env):

my\_list = []

position = int(position)

for e in env:

if position in e['5KStep'].keys():

plus5k = [e['5KStep'][position]['Actions']['Plus5k']['Q'],'Plus5k',e['5KStep'][position]['Actions']['Plus5k']['ActionVal'],e['5KStep'][position]['Actions']['Plus5k']['Exit']]

minus5k = [e['5KStep'][position]['Actions']['Minus5k']['Q'],'Minus5k',e['5KStep'][position]['Actions']['Minus5k']['ActionVal'],e['5KStep'][position]['Actions']['Minus5k']['Exit']]

minus100k = [e['5KStep'][position]['Actions']['Minus100k']['Q'], 'Minus100k',e['5KStep'][position]['Actions']['Minus100k']['ActionVal'],e['5KStep'][position]['Actions']['Minus100k']['Exit']]

plus100k = [e['5KStep'][position]['Actions']['Plus100k']['Q'], 'Plus100k',e['5KStep'][position]['Actions']['Plus100k']['ActionVal'],e['5KStep'][position]['Actions']['Plus100k']['Exit']]

my\_list.append(plus5k)

my\_list.append(minus5k)

my\_list.append(minus100k)

my\_list.append(plus100k)

# enters the env and supplies the possible choices to the list

# the list is sorted in descendiing order

my\_list = sorted(my\_list, key = operator.itemgetter(0), reverse=True)

maximumq = my\_list[0][0]

# the first is set as the highest and then a random choice is enforced

# if there is a match of two choices

final\_list = []

for i in my\_list:

if i[0] == maximumq:

final\_list.append(i)

nextstep = random.choice(final\_list)

return nextstep

def get\_max(position,env):

my\_list = []

for e in env:

# enters the environment and finds the mas then returns the relevant data

if position in e['5KStep'].keys():

plus5k = [e['5KStep'][position]['Actions']['Plus5k']['Q'], 'Plus5k',

e['5KStep'][position]['Actions']['Plus5k']['ActionVal']]

minus5k = [e['5KStep'][position]['Actions']['Minus5k']['Q'], 'Minus5k',

e['5KStep'][position]['Actions']['Minus5k']['ActionVal']]

minus100k = [e['5KStep'][position]['Actions']['Minus100k']['Q'], 'Minus100k',

e['5KStep'][position]['Actions']['Minus100k']['ActionVal']]

plus100k = [e['5KStep'][position]['Actions']['Plus100k']['Q'], 'Plus100k',

e['5KStep'][position]['Actions']['Plus100k']['ActionVal']]

my\_list.append(plus5k)

my\_list.append(minus5k)

my\_list.append(minus100k)

my\_list.append(plus100k)

my\_list = sorted(my\_list, key=operator.itemgetter(0), reverse=True)

maximumq = my\_list[0][0]

return maximumq

**import predictios**

import csv

import re

from operator import itemgetter

def import\_predictions():

with open('..\\..\\..\\Q\_Learning\_CSVs\\Predictions.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

x=0

my\_list=[]

for i in readCSV:

if i and x > 0:

my\_dict = {

'Position':'',

'Area':'',

'Type':'',

'Counter':''

}

my\_dict['Position']=i[0]

my\_dict['Area']=i[1]

my\_dict['Type']=i[2]

my\_dict['Counter']=i[3]

my\_list.append(my\_dict)

x+=1

for j in my\_list:

j['Counter'] =j['Counter'].replace(' ','').split(',')

count\_list=[]

for i in range(len(j['Counter'])):

test = True

count\_dict = {}

count\_dict['value'] = re.findall(r"'(.\*?)'", j['Counter'][i])

count\_dict['probability'] = re.findall(r':(\d+)', j['Counter'][i])

if count\_dict['probability']:

count\_dict['probability'] = int(count\_dict['probability'][0])/2

count\_list.append(count\_dict)

j['Counter']=sorted(count\_list, key=itemgetter('probability'), reverse=True)

return my\_list

**import twenties**

import csv

def import\_Dublin(htype):

with open('..\\..\\..\\Q\_Learning\_CSVs\\Dublin15'+str(htype)+'Tewnty.csv') as csv\_file:

readCSV = csv.reader(csv\_file, delimiter=',')

a = 0

all\_houses = []

for i in readCSV:

hinfo = {

'price': '',

'address': '',

'bedrooms': '',

'bathrooms': '',

'type': '',

'area': '',

'Garage': '',

'Garden': '',

'Ensuite': '',

'LatLng': '',

'DateOfSale': '',

'SalePrice': '',

'label' : ''

}

if i and a > 0:

hinfo['price'] = i[0]

hinfo['address'] = i[1]

hinfo['bedrooms'] = i[2]

hinfo['bathrooms'] = i[3]

hinfo['type'] = i[4]

hinfo['area'] = i[5]

hinfo['Garage'] = i[6]

hinfo['Garden'] = i[7]

hinfo['Ensuite'] = i[8]

hinfo['LatLng'] = i[9]

hinfo['DateOfSale'] = i[10]

hinfo['SalePrice'] = i[11]

hinfo['label'] = i[12]

all\_houses.append(hinfo)

a += 1

new\_houses=[]

for house in all\_houses:

if house['label']:

new\_houses.append(house)

new\_houses=add\_pos(new\_houses)

return new\_houses

def add\_pos(houses):

for i in range(1,11):

for x in range(1,11):

for y in range(1,11):

for house in houses:

if int(house['bedrooms']) == i:

if int(house['bathrooms']) == x:

if int(house['label']) == y:

house['pos']= 'bd' + str(i) + 'br' + str(x) + 'area' + str(y)

return houses

**make predictions**

import Import\_twenties

import Import\_predictions

from pathlib import Path

import csv

def import\_andrun():

type\_list1 = ['Terraced\_House', 'Apartment\_For\_Sale', 'Bungalow'

'Duplex', 'End\_of\_T', 'House\_For\_Sale', 'New\_Dwelling',

'Second-Hand',

'Site', 'Terraced' 'Townhouse', 'Detatched\_House', 'Semi-Detached']

type\_list2 = ['Semi-Detached']

all\_Houses = {}

for t in type\_list1:

my\_file = Path('..\\..\\..\\Q\_Learning\_CSVs\\Dublin15'+str(t)+'Tewnty.csv')

if my\_file.is\_file():

all\_Houses[str(t)]=Import\_twenties.import\_Dublin(t)

return all\_Houses

def predict\_values():

allHouses = import\_andrun()

predictions = Import\_predictions.import\_predictions()

for h in allHouses:

for house in allHouses[str(h)]:

house['prediction']= ''

for p in predictions:

if str(p['Type'][0:2]) == str(house['type'][0:2]):

if str(p['Position'])== str(house['pos']):

if p['Counter']:

house['prediction'] = p['Counter'][0]['value']

newhouses = []

for h in allHouses:

print(h)

for house in allHouses[str(h)]:

print(house['price'])

print(house['prediction'])

newhouses.append(house)

return allHouses

**test q learning**

import make\_predictions

import math

import csv

allHouses = make\_predictions.predict\_values()

figure\_list = []

for h in allHouses:

print(h)

for house in allHouses[str(h)]:

mydict={}

mydict['price'] = float(house['price'])/10000

mydict['prediction' ]=house['prediction']

if mydict['prediction']:

figure\_list.append(mydict)

sigmax = 0

sigmay = 0

sigmaxy=0

sigmax2=0

sigmay2=0

keys = figure\_list[0].keys()

print(keys)

with open('..\\..\\..\\Q\_Learning\_CSVs\\testinPredictions.csv', 'w+') as output\_file:

dict\_writer = csv.DictWriter(output\_file, keys)

dict\_writer.writeheader()

dict\_writer.writerows(figure\_list)

for fig in figure\_list:

fig['prediction'] = float(fig['prediction'][0])/10000

print(fig)

fig['pricesq']= float(fig['price']) \* float(fig['price'])

fig['predictsq']=float(fig['prediction']) \* float(fig['prediction'])

fig['xy']=float(fig['price']) \* float(fig['prediction'])

sigmax += float(fig['price'])

sigmay += float(fig['prediction'])

sigmaxy += float(fig['xy'])

sigmax2 += float(fig['pricesq'])

sigmay2 += float(fig['predictsq'])

print(sigmax)

print(sigmay)

print(sigmaxy)

print(sigmax2)

print(sigmay2)

n = len(figure\_list)

print(n)

topl = sigmaxy \* n

topr = sigmax \* sigmay

top = topl - topr

bottomleftl = n \* sigmax2

bottomleftr = sigmax \* sigmax

bottomleft = bottomleftl - bottomleftr

bottomrightl = n \* sigmay2

bottomrightr = sigmay \* sigmay

bottomright = bottomrightl - bottomrightr

bottomcentre = bottomleft \* bottomright

bottom = math.sqrt(bottomcentre)

total = top/bottomcentre

print('Accuracy is')

print(total)