In [1]:

```
#https://github.com/imasha1/DMHR #Github URL

import pandas
import pandasql
from datetime import datetime
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib.dates as dates
from scipy.stats import norm
import numpy as np
sql = pandasql.PandaSQL()

#Need to import Pandas library, PandaSQL, Matplotlib, Numpys etc in Jupyter Notebook so it’s loade
d into the memory and is available to work with
#Add ‘as plt’ and ‘as np’ so can access Plotting library and Numpys just by writing ‘plt.command’
etc every time you need to use it.
#A date in Python is not a data type of its own, so need to import it as datetime
```

```
C:\Users\imash\Anaconda3\lib\site-packages\ipykernel\parentpoller.py:116: UserWarning: Parent poll
failed.  If the frontend dies,
                the kernel may be left running.  Please let us know
                about your system (bitness, Python, etc.) at
                ipython-dev@scipy.org
  ipython-dev@scipy.org""")
```

In [2]:

```
import sys
#sys is system-specific parameters and functions. This module provides access to some variables us
ed or maintained by the interpreter and to functions that interact strongly with the interpreter
```

In [3]:

```
def find_patients_registered(practice_codes, practice_patient_data):
  return sql('select practice_code, number_of_patients from practice_codes left join
practice_patient_data on practice_code = code')
#Def defines a funtion, in this case function is defined as to find patients registered, with the
parameters being practice codes and patient data
#This code is using and Pandas and PandaSQL
#The return statement causes your find patients registered function to exit and hand back a value
to its caller with parameters being select practice codes, number of patients from practice codes
left join practice patient data on practice code
```

In [4]:

```
def find_prescriptions_count(practice_codes, practice_prescription_data):
  return sql('select practice_code, sum(quantity) as quantity from practice_codes left join
practice_prescription_data on practice_code = practice group by practice_code')
#Def defines a funtion, in this case function is defined as to find prescriptions count, with the
parameters being practice codes and practice prescription data
#This code is using and Pandas and PandaSQL
#The return statement causes your find prescriptions count function to exit and hand back a value
to its caller with parameters being select practice codes, total quantity from practice codes left
join practice data on practice codes equalling practice group by practice code
```

In [5]:

```
def find_prescriptions_cost(practice_codes, practice_prescription_data):
  return sql('select practice_code, sum([ACT COST   ]) as cost from practice_codes left join
practice_prescription_data on practice_code = practice group by practice_code')
#Def defines a function, in this case function is defined as to find prescriptions cost, with the
parameters being practice codes and practice prescription data
#This code is using and Pandas and PandaSQL
#The return statement causes your find prescriptions cost function to exit and hand back a value t
o its caller with parameters being select practice codes, total quantity as cost from practice cod
es left join practice prescription data on practice code equalling practice group by practice code
```

In [6]:

```
#the top 10 most frequent drugs prescribed
#Def defines a function, in this case function is defined as to find top 10 most frequent drugs, w
ith the parameters being practice codes and practice prescription data
#This code is using and Pandas and PandaSQL

#The return statement causes your find top 10 most frequent drugs function to exit and hand back a
value to its caller with parameters being select practice codes, BNF name, total quantity as quant
ity from practice codes left join practice prescription data on practice code equalling practice g
roup by practice code, name order by practice code and quantity in ascending order
def find_top_10_frequent_drugs(practice_codes, practice_prescription_data):
  return sql('select practice_code, [BNF NAME                                    ] as name, sum([qu
ntity]) as quantity from practice_codes left join practice_prescription_data on practice_code = pr
actice group by practice_code, name order by practice_code, quantity DESC')
```

In [7]:

```
#the bottom 10 least frequent drugs prescribed
def find_least_10_frequent_drugs(practice_codes, practice_prescription_data):
  return sql('select practice_code, [BNF NAME                                    ] as name, sum([qu
ntity]) as quantity from practice_codes left join practice_prescription_data on practice_code = pr
actice group by practice_code, name order by practice_code, quantity ASC')
#Def defines a function, in this case function is defined as to find top 10 frequent drugs, with t
he parameters being practice codes and practice prescription data
#This code is using and Pandas and PandaSQL
#The return statement causes your find top 10 frequent drugs function to exit and hand back a valu
e to its caller with parameters being select practice codes, BNF name, total quantity as quantity
from practice codes left join practice prescription data on practice code equalling practice group
by practice code, name order by practice code and quantity in descending order
```

In [8]:

```
def find_prescriptions_count_and_cost_cardiovascular(practice_prescription_data):
  # needs checking
  cardiovascular_drugs = pandas.DataFrame([{"name": "%%Digoxin%%"}, {"name": "%%Indapamide%%"}, {"n
ame": "%%Bendroflumethiazide%%"}, {"name": "%%Furosemide%%"}, {"name": "%%Bumetanide%%"}, {"name":
"%%Amiloride%%"}, {"name": "%%Eplerenone%%"}, {"name": "%%Spironolactone%%"}, {"name": "%%Co-amilof
ruse%%"}, {"name": "%%Amiodarone%%"}, {"name": "%%Flecainide%%"}, {"name": "%%Dronedarone%%"}, {"na
me": "%%Bisoprolol%%"}, {"name": "%%Carvedilol%%"}, {"name": "%%Labetalol%%"}, {"name": "%%Proprano
lol%%"}, {"name": "%%Atenolol%%"}, {"name": "%%Hydralazine%%"}, {"name": "%%Methyldopa%%"}, {"name"
: "%%Moxonidine%%"}, {"name": "%%Doxazosin%%"}, {"name": "%%Sacubitril%%"}, {"name": "%%Valsartan%%
"}, {"name": "%%Lisinopril%%"}, {"name": "%%Perindopril erbumine%%"}, {"name": "%%Ramipril%%"}, {"n
ame": "%%Candesartan%%"}, {"name": "%%Irbesartan%%"}, {"name": "%%Losartan%%"}, {"name": "%%Valsart
an%%"}, {"name": "%%Glyceryl%%"}, {"name": "%%Isosorbide%%"}, {"name": "%%Amlodipine%%"}, {"name":
"%%Lercanidipine%%"}, {"name": "%%Diltiazem%%"}, {"name": "%%Verapamil%%"}, {"name": "%%Ivabradine%
%"}, {"name": "%%Nicorandil%%"}, {"name": "%%Ranolazine%%"}, {"name": "%%Naftidrofuryl%%"}, {"name"
: "%%Bivalirudin%%"}, {"name": "%%Apixaban%%"}, {"name": "%%Dabigatran%%"}, {"name": "%%Edoxaban%%"
}, {"name": "%%Rivaroxaban%%"}, {"name": "%%Warfarin%%"}, {"name": "%%Acenocoumarol%%"}, {"name": "
%%Idarucizumab%%"}, {"name": "%%Aspirin%%"}, {"name": "%%Clopidogrel%%"}, {"name": "%%Dipyridamole%
%"}, {"name": "%%Prasugrel%%"}, {"name": "%%Ticagrelor%%"}, {"name": "%%Aspirin%%"}, {"name": "%%di
pyridamole%%"}, {"name": "%%Tranexamic acid%%"}, {"name": "%%Atorvastatin%%"}, {"name": "%%Simvasta
tin%%"}, {"name": "%%Rosuvastatin%%"}, {"name": "%%Pravastatin%%"}, {"name": "%%Colestyramine%%"},
{"name": "%%Ezetimibe%%"}, {"name": "%%Fenofibrate%%"}, {"name": "%%Alirocumab%%"}, {"name": "%%Evo
locumab%%"}])
  return sql('select sum(quantity) as quantity, sum([ACT COST   ]) as cost from
practice_prescription_data where (select count(*) as count from cardiovascular_drugs where [BNF NA
ME                                    ] like name) > 0')
#Def defines a function, in this case function is defined as to find prescriptions number and cost
of cardiovascular drugs, with the parameters being practice prescription data
#This code is using and Pandas and PandaSQL
#defining cardiovascular_drugs as all data frames that has said drugs included - Digoxin, Indapami
de, Bendroflumethiazide, Furosemide, Bumetanide, Amiloride, Eplerone, Spironolactone, Co-amilofrus
e etc.
#The return statement causes your find prescriptions number and cost of cardiovascular drugs funct
ion to exit and hand back a value to its caller with parameters being ‘select total quantity’ as q
uantity, ‘total of cost’ as cost from practice prescription data select count as count from cardio
vascular drugs where BNF name like relevant drug names.
```

In [9]:

```
def find_prescriptions_count_and_cost_antidepressants(practice_prescription_data):
  # needs checking
  antidepressants_drugs = pandas.DataFrame([{"name": "%%Amitriptyline hydrochloride%%"}, {"name": "
%%Clomipramine hydrochloride%%"}, {"name": "%%Dosulepin hydrochloride%%"}, {"name":
"%%Doxepin%%"}, {"name": "%%Imipramine hydrochloride%%"}, {"name": "%%Lofepramine%%"}, {"name": "%%
Mianserin hydrochloride%%"}, {"name": "%%Nortriptyline%%"}, {"name": "%%Trazodone hydrochloride%%"}
, {"name": "%%Trimipramine%%"}, {"name": "%%Isocarboxazid%%"}, {"name": "%%Moclobemide%%"}, {"name"
: "%%Phenelzine%%"}, {"name": "%%Tranylcypromine%%"}, {"name": "%%Citalopram%%"}, {"name": "%%Escit
alopram%%"}, {"name": "%%Fluoxetine%%"}, {"name": "%%Fluvoxamine maleate%%"}, {"name": "%%Paroxetin
e%%"}, {"name": "%%Sertraline%%"}, {"name": "%%Agomelatine%%"}, {"name": "%%Duloxetine%%"}, {"name"
: "%%Flupentixol%%"}, {"name": "%%Mirtazapine%%"}, {"name": "%%Reboxetine%%"}, {"name": "%%Tryptoph
an%%"}, {"name": "%%Venlafaxine%%"}])
  return sql('select sum(quantity) as quantity, sum([ACT COST   ]) as cost from
practice_prescription_data where (select count(*) as count from antidepressants_drugs where [BNF N
AME                                        ] like name) > 0')
#Def defines a function, in this case function is defined as to find prescriptions number and cost
of antidepressants drugs, with the parameters being practice prescription data
#This code is using and Pandas and PandaSQL
#defining antidepressants_drugs as all data frames that has relevant antidepressant drugs included
.
#The return statement causes your find prescriptions number and cost of antidepressants drugs func
tion to exit and hand back a value to its caller with parameters being ‘select total quantity’ as
quantity, ‘total of cost’ as cost from practice prescription data select count as count from antid
epressants drugs where BNF name like relevant drug names.
```

In [10]:

```
practice_prescription_data = pandas.read_csv('C:\\Users\\imash\\Downloads\\T201804PDPI+BNFT.CSV')
#importing practice prescription data from NHS as panda and naming it as
practice_prescription_data
```

In [11]:

```
practice_patient_data = pandas.read_csv('C:\\Users\\imash\\Downloads\\gp-reg-pat-prac-all.CSV')
#importing practice patient data from NHS as panda and naming it as practice_patient_data
```

In [12]:

```
practice_location_data = pandas.read_csv('C:\\Users\\imash\\Downloads\\T201804ADDR+BNFT.CSV', name
s=['date', 'practice_code', '_1', '_2', '_3', 'city', 'county', 'post_code'])
#importing practice location data from NHS as panda and naming it as practice_location_data and al
so adding headings to each coloumns to make address clearer.
```

In [13]:

```
# Identify all GP practices located in London, by using PandaSQL to search and select all data whe
re the word 'London' was included in city or county
london_practice_codes = sql('select practice_code from practice_location_data where city like "%%l
ondon%%" or county like "%%london%%"')
```

In [14]:

```
# LONDON: the total number of patients registered with parameters as london practice codes and pra
ctice prescription data

find_patients_registered(london_practice_codes, practice_patient_data)
```

Out[14]:

	practice_code	NUMBER_OF_PATIENTS
0	E82113	4148.0
1	E83003	8911.0
2	E83005	6224.0
3	E83006	6885.0
4	E83007	5706.0
5	E83008	7900.0
6	E83009	10822.0
7	E83010	11407.0
8	E83011	[illegible]

	practice_code	NUMBER_OF_PATIENTS
8	E83011	8116.0
9	E83013	6505.0
10	E83016	18356.0
11	E83020	10855.0
12	E83021	12552.0
13	E83024	10704.0
14	E83025	8930.0
15	E83026	4405.0
16	E83027	7389.0
17	E83034	5506.0
18	E83035	9919.0
19	E83037	5481.0
20	E83039	7452.0
21	E83041	4687.0
22	E83045	8670.0
23	E83046	9894.0
24	E83049	7587.0
25	E83050	7668.0
26	E83600	5606.0
27	E83621	8468.0
28	E83622	7047.0
29	E83631	1273.0
...	...	...
919	Y05454	NaN
920	Y05455	NaN
921	Y05481	NaN
922	Y05482	NaN
923	Y05493	NaN
924	Y05502	NaN
925	Y05552	NaN
926	Y05565	NaN
927	Y05566	NaN
928	Y05642	NaN
929	Y05643	NaN
930	Y05644	NaN
931	Y05646	NaN
932	Y05658	NaN
933	Y05686	NaN
934	Y05689	NaN
935	Y05691	NaN
936	Y05696	NaN
937	Y05741	NaN
938	Y05813	NaN
939	Y05837	NaN
940	Y05838	NaN
941	Y05850	NaN
942	Y05855	NaN
943	Y05858	NaN
944	Y05881	NaN
945	Y05889	NaN
946	Y05904	NaN

	practice_code	NUMBER_OF_PATIENTS
947	Y05954	NaN
948	Y05984	NaN

949 rows × 2 columns

In [15]:

```
# LONDON: the total number of prescriptions with parameters as london practice codes and practice
prescription data
find_prescriptions_count(london_practice_codes, practice_prescription_data)
```

Out[15]:

	practice_code	quantity
0	E82113	338001
1	E83003	950148
2	E83005	578742
3	E83006	618685
4	E83007	531001
5	E83008	663524
6	E83009	913246
7	E83010	1230721
8	E83011	977745
9	E83013	638429
10	E83016	1666530
11	E83020	795791
12	E83021	1368144
13	E83024	1142283
14	E83025	1243020
15	E83026	407709
16	E83027	1589639
17	E83034	694852
18	E83035	1143305
19	E83037	976488
20	E83039	445057
21	E83041	277609
22	E83045	888179
23	E83046	950803
24	E83049	757729
25	E83050	524097
26	E83600	1105796
27	E83621	703154
28	E83622	557524
29	E83631	1182
...	...	...
919	Y05454	101096
920	Y05455	4838
921	Y05481	2
922	Y05482	2001
923	Y05493	52860
924	Y05502	2662
925	Y05552	63526
926	Y05565	10599

	practice_code	quantity
927	Y05566	221698
928	Y05642	181046
929	Y05643	208481
930	Y05644	144686
931	Y05646	112
932	Y05658	781
933	Y05686	24742
934	Y05689	233
935	Y05691	28
936	Y05696	51998
937	Y05741	13625
938	Y05813	100433
939	Y05837	43025
940	Y05838	5164
941	Y05850	4554
942	Y05855	106
943	Y05858	1885
944	Y05881	1365
945	Y05889	14110
946	Y05904	457
947	Y05954	2753
948	Y05984	1755

949 rows × 2 columns

In [16]:

```
# LONDON: the total actual cost of these prescriptions with parameters as london practice codes an
d practice prescription data
find_prescriptions_cost(london_practice_codes, practice_prescription_data)
```

Out[16]:

	practice_code	cost
0	E82113	30991.45
1	E83003	89607.38
2	E83005	47240.62
3	E83006	54746.99
4	E83007	53459.56
5	E83008	71808.73
6	E83009	92674.43
7	E83010	100409.55
8	E83011	74748.34
9	E83013	49177.60
10	E83016	156116.22
11	E83020	95638.95
12	E83021	119132.05
13	E83024	94602.61
14	E83025	95683.40
15	E83026	39761.96
16	E83027	97078.61
17	E83034	55907.99
18	E83035	107449.47

	practice_code	cost
18	E83035	107449.47
19	E83037	74762.48
20	E83039	41430.87
21	E83041	24456.39
22	E83045	74572.81
23	E83046	77196.80
24	E83049	62810.14
25	E83050	55251.19
26	E83600	58906.09
27	E83621	63892.06
28	E83622	58970.71
29	E83631	275.46
...	...	...
919	Y05454	6891.33
920	Y05455	319.74
921	Y05481	7.57
922	Y05482	435.29
923	Y05493	4948.88
924	Y05502	293.15
925	Y05552	4912.82
926	Y05565	1329.30
927	Y05566	13107.37
928	Y05642	6734.60
929	Y05643	6694.46
930	Y05644	2432.98
931	Y05646	2.50
932	Y05658	61.01
933	Y05686	2363.14
934	Y05689	56.56
935	Y05691	2.13
936	Y05696	3797.76
937	Y05741	944.75
938	Y05813	25356.57
939	Y05837	2222.33
940	Y05838	90.32
941	Y05850	531.83
942	Y05855	3.56
943	Y05858	985.04
944	Y05881	1634.96
945	Y05889	2795.21
946	Y05904	110.54
947	Y05954	160.60
948	Y05984	1007.80

949 rows × 2 columns

In [17]:

```
# LONDON: the top 10 most frequent drugs prescribed with parameters as london practice codes and p
ractice prescription data
find_top_10_frequent_drugs(london_practice_codes, practice_prescription_data)
```

Out[17]:

	practice_code	name	quantity
0	E82113	Fortisip Compact_Liq (8 Flav)	14000
1	E82113	Ensure Plus_Milkshake Style Liq (9 Flav)	12200
2	E82113	Ensure TwoCal_Liq (4 Flav)	11200
3	E82113	Ensure Compact_Liq (4 Flav)	11000
4	E82113	Lactulose_Soln 3.1g-3.7g/5ml	10700
5	E82113	Metformin HCl_Tab 500mg	9751
6	E82113	Zerobase Crm (Appl)	8500
7	E82113	Zerocream (Appl)	6100
8	E82113	Neocate LCP_Pdr	5600
9	E82113	Omeprazole_Cap E/C 20mg	5334
10	E82113	Paracet_Tab 500mg	5328
11	E82113	Amlodipine_Tab 5mg	4914
12	E82113	Nutramigen 2 + LGG_Pdr	4800
13	E82113	Gaviscon Advance_Liq (Aniseed) (Reckitt)	4500
14	E82113	Co-Codamol_Tab 8mg/500mg	4250
15	E82113	Simvastatin_Tab 40mg	4189
16	E82113	Emulsif_Oint	4000
17	E82113	Nutramigen 1 + LGG_Pdr	4000
18	E82113	ZeroAQS Crm	4000
19	E82113	Acidex_Liq (Aniseed)	3500
20	E82113	Dermol 500_Lot	3500
21	E82113	Co-Codamol_Tab 30mg/500mg	3452
22	E82113	Gliclazide_Tab 80mg	3080
23	E82113	Gaviscon Advance_Liq (Peppermint) S/F	3000
24	E82113	Atorvastatin_Tab 20mg	2632
25	E82113	Ramipril_Cap 10mg	2604
26	E82113	Amlodipine_Tab 10mg	2520
27	E82113	Lansoprazole_Cap 30mg (E/C Gran)	2520
28	E82113	Prazosin HCl_Tab 1mg	2520
29	E82113	Ramipril_Cap 5mg	2520
...	...	...	...
825574	Y05889	Cetraben Crm 500g	3
825575	Y05889	Ingenol Mebutate_Gel 150mcg/g	2
825576	Y05904	Betameth Sod Phos_Ear/Eye/Nsl Dps 0.1%	290
825577	Y05904	Betameth Val_Scalp Applic 0.1%	100
825578	Y05904	Doxycycline Hyclate_Cap 100mg	39
825579	Y05904	Chlorhex HCl/Neomycin Sulf_Crm 0.1/0.5%	15
825580	Y05904	Ciprofloxacin_Eye Dps 0.3%	5
825581	Y05904	Otovent Inflation Dev Auto	5
825582	Y05904	Fluticasone Fur_Nsl Spy 27.5mcg (120D)	2
825583	Y05904	T & R Sod Bicarb Ear Dps 5% Ear Wax Soft	1
825584	Y05954	Peptac_Liq (Aniseed) S/F	1000
825585	Y05954	Trimethoprim_Oral Susp 50mg/5ml S/F	560
825586	Y05954	Prednisolone_Tab 5mg	280
825587	Y05954	Cetirizine HCl_Oral Soln 1mg/1ml S/F	200
825588	Y05954	Betameth Sod Phos_Ear/Eye/Nsl Dps 0.1%	140
825589	Y05954	Omeprazole_Cap E/C 20mg	140
825590	Y05954	Doxycycline Hyclate_Cap 100mg	92
825591	Y05954	Amoxicillin_Cap 500mg	75

	practice_code	name	quantity
825592	Y05954	Timodine_Crm	60
825593	Y05954	Oxytetracycline_Tab 250mg	56
825594	Y05954	Clarithromycin_Tab 500mg	42
825595	Y05954	Chlorhex HCl/Neomycin Sulf_Crm 0.1/0.5%	30
825596	Y05954	Prochlpzine Mal_Tab Buccal 3mg	30
825597	Y05954	Mometasone Fur_Aq N/Spy 50mcg (140 D)	18
825598	Y05954	Naseptin_Crm	15
825599	Y05954	Betameth/Neomycin_Ear/Eye/Nose Dps.1/.5%	10
825600	Y05954	Otomize_Ear Spy 5ml	3
825601	Y05954	T & R Sod Bicarb Ear Dps 5% Ear Wax Soft	2
825602	Y05984	Roaccutane_Cap 20mg	1290
825603	Y05984	Roaccutane_Cap 10mg	465

825604 rows × 3 columns

In [18]:

```
# LONDON: the bottom 10 less frequent drugs prescribed with parameters as london practice codes and practice precription data
find_least_10_frequent_drugs(london_practice_codes, practice_prescription_data)
```

Out[18]:

	practice_code	name	quantity
0	E82113	3m Health Care_Cavilon No Sting Barrier	1
1	E82113	Acti-Fast 2-Way Stch 10.75cmx3m(Yell)Stk	1
2	E82113	Activa Leg Ulcer Hose Kit Med O/T Compre	1
3	E82113	BioXtra Dry Mth Oral gel 40ml (App)	1
4	E82113	Ciclesonide_Inh 160mcg (60 D) CFF	1
5	E82113	Clenil Modulite_Inha 50mcg (200D)	1
6	E82113	CliniFast 10.75cm x 5m (Yellow) Stkntte	1
7	E82113	Colifoam_Foam Aero Enem 10% 20.8g (14 D)	1
8	E82113	Coloplast SpeediCath Compact Plus Fle 10	1
9	E82113	Comfifast 10.75cm x 5m (Yellow)Stkntte E	1
10	E82113	ConvaTec_Orabase Paste 30g	1
11	E82113	Dansac_Nodor S Deod 50ml	1
12	E82113	Desmopressin Acet_I/Nsl Spy 10mcg (60 D)	1
13	E82113	Fluticasone Prop_Nsl Spy 50mcg (150 D)	1
14	E82113	Fluticasone/Salmeterol_Inh 250/50mcg 60D	1
15	E82113	Fluticasone/Vilanterol_Inha 184/22mcg30D	1
16	E82113	Formoterol Fumar_Pdr For Inh 12mcg(120D)	1
17	E82113	Glyceryl Trinit_Sub A/Spy 400mcg (180D)	1
18	E82113	Ibandronic Acid_Tab 150mg	1
19	E82113	Ipratrop Brom_Inha 20mcg (200 D) CFF	1
20	E82113	Leuprorelin Acet_Inj 11.25mg Dil + Pfs	1
21	E82113	Medroxyprogest Acet_Inj 150mg/ml 1ml Pfs	1
22	E82113	Mirena_Intra-Uterine System	1
23	E82113	Nafarelin Acet_Nsl Spy 200mcg (60 D)	1
24	E82113	Nebido_Inj 1000mg/4ml Vl	1
25	E82113	Peak Flow Meter Unspec Low Range Hand He	1
26	E82113	Salbutamol_Pdr For Inh 200mcg (60 D)	1
27	E82113	Tiotropium_Inha 2.5mcg (60D) CFF + Dev	1

	practice_code	name	quantity
27	E82113	Tiotropium_Inha 2.5mcg (60D) CFF + Dev	1
28	E82113	Triamcinol Aceton_Aq Nsl Spy 55mcg(120D)	1
29	E82113	3m Health Care_Cavilon Durable Barrier C	2
...	...	...	...
825574	Y05889	Isotretinoin_Cap 20mg	1630
825575	Y05889	Dermol 500_Lot	4500
825576	Y05904	T & R Sod Bicarb Ear Dps 5% Ear Wax Soft	1
825577	Y05904	Fluticasone Fur_Nsl Spy 27.5mcg (120D)	2
825578	Y05904	Ciprofloxacin_Eye Dps 0.3%	5
825579	Y05904	Otovent Inflation Dev Auto	5
825580	Y05904	Chlorhex HCl/Neomycin Sulf_Crm 0.1/0.5%	15
825581	Y05904	Doxycycline Hyclate_Cap 100mg	39
825582	Y05904	Betameth Val_Scalp Applic 0.1%	100
825583	Y05904	Betameth Sod Phos_Ear/Eye/Nsl Dps 0.1%	290
825584	Y05954	T & R Sod Bicarb Ear Dps 5% Ear Wax Soft	2
825585	Y05954	Otomize_Ear Spy 5ml	3
825586	Y05954	Betameth/Neomycin_Ear/Eye/Nose Dps.1/.5%	10
825587	Y05954	Naseptin_Crm	15
825588	Y05954	Mometasone Fur_Aq N/Spy 50mcg (140 D)	18
825589	Y05954	Chlorhex HCl/Neomycin Sulf_Crm 0.1/0.5%	30
825590	Y05954	Prochlpzine Mal_Tab Buccal 3mg	30
825591	Y05954	Clarithromycin_Tab 500mg	42
825592	Y05954	Oxytetracycline_Tab 250mg	56
825593	Y05954	Timodine_Crm	60
825594	Y05954	Amoxicillin_Cap 500mg	75
825595	Y05954	Doxycycline Hyclate_Cap 100mg	92
825596	Y05954	Betameth Sod Phos_Ear/Eye/Nsl Dps 0.1%	140
825597	Y05954	Omeprazole_Cap E/C 20mg	140
825598	Y05954	Cetirizine HCl_Oral Soln 1mg/1ml S/F	200
825599	Y05954	Prednisolone_Tab 5mg	280
825600	Y05954	Trimethoprim_Oral Susp 50mg/5ml S/F	560
825601	Y05954	Peptac_Liq (Aniseed) S/F	1000
825602	Y05984	Roaccutane_Cap 10mg	465
825603	Y05984	Roaccutane_Cap 20mg	1290

825604 rows × 3 columns

In [19]:

```
# Identify all GP practices located in Cambridge by using PandaSQL to search and select all data w
here the word 'cambridge' was included in city or county
cambridge_practice_codes = sql('select practice_code from practice_location_data where city like "
%%cambridge%%" or county like "%%cambridge%%"')
```

In [20]:

```
# CAMBRIDGE: the total number of patients registered with parameters as cambridge practice codes a
nd practice prescription data
find_patients_registered(cambridge_practice_codes, practice_patient_data)
```

Out[20]:

	practice_code	NUMBER_OF_PATIENTS
0	C83026	23895.0
1	D81001	12057.0

	practice_code	NUMBER_OF_PATIENTS
1	D81001	[illegible]
2	D81002	16939.0
3	D81003	9927.0
4	D81004	9772.0
5	D81005	14941.0
6	D81007	9236.0
7	D81008	19605.0
8	D81009	9071.0
9	D81010	11943.0
10	D81011	11766.0
11	D81012	11332.0
12	D81013	17443.0
13	D81014	20166.0
14	D81015	6626.0
15	D81016	12557.0
16	D81017	6190.0
17	D81019	4151.0
18	D81020	13069.0
19	D81021	11112.0
20	D81022	8132.0
21	D81023	14070.0
22	D81025	10694.0
23	D81026	25225.0
24	D81027	7453.0
25	D81028	12288.0
26	D81029	12394.0
27	D81030	11104.0
28	D81032	11741.0
29	D81033	4700.0
...	...	...
97	Y00673	NaN
98	Y01778	NaN
99	Y02051	NaN
100	Y02769	6243.0
101	Y02888	NaN
102	Y02978	NaN
103	Y02990	NaN
104	Y02991	NaN
105	Y02992	NaN
106	Y02993	NaN
107	Y02994	NaN
108	Y03041	NaN
109	Y03272	NaN
110	Y03454	NaN
111	Y03472	NaN
112	Y03550	NaN
113	Y04145	NaN
114	Y04184	NaN
115	Y04185	NaN
116	Y04186	NaN
117	Y04208	NaN

	practice_code	NUMBER_OF_PATIENTS
118	Y04210	NaN
119	Y04212	NaN
120	Y04214	NaN
121	Y04602	NaN
122	Y05140	NaN
123	Y05173	NaN
124	Y05176	NaN
125	Y05474	NaN
126	Y05839	NaN

127 rows × 2 columns

In [21]:

```
# CAMBRIDGE: the total number of prescriptions with parameters as cambridge practice codes and pra
ctice prescription data
find_prescriptions_count(cambridge_practice_codes, practice_prescription_data)
```

Out[21]:

	practice_code	quantity
0	C83026	2887247
1	D81001	575098
2	D81002	890112
3	D81003	736032
4	D81004	1108492
5	D81005	600923
6	D81007	1283177
7	D81008	2670335
8	D81009	833888
9	D81010	1628287
10	D81011	1761848
11	D81012	1155284
12	D81013	706811
13	D81014	1840784
14	D81015	952755
15	D81016	1055050
16	D81017	542217
17	D81019	545550
18	D81020	2039251
19	D81021	1286622
20	D81022	998380
21	D81023	1846188
22	D81025	899892
23	D81026	3600174
24	D81027	736747
25	D81028	1652780
26	D81029	1396681
27	D81030	1112260
28	D81032	1544914
29	D81033	289382
...	...	...
97	Y00673	23258

	practice_code	quantity
98	Y01778	617
99	Y02051	71
100	Y02769	337700
101	Y02888	806
102	Y02978	200
103	Y02990	43
104	Y02991	1639
105	Y02992	5779
106	Y02993	4069
107	Y02994	18322
108	Y03041	1082
109	Y03272	6381
110	Y03454	442548
111	Y03472	3755
112	Y03550	29450
113	Y04145	263
114	Y04184	1005
115	Y04185	10857
116	Y04186	1891
117	Y04208	5433
118	Y04210	2417
119	Y04212	750
120	Y04214	50
121	Y04602	8239
122	Y05140	22587
123	Y05173	412
124	Y05176	1141
125	Y05474	1361
126	Y05839	1130

127 rows × 2 columns

In [22]:

```
# CAMBRIDGE: the total actual cost of these prescriptions with parameters as cambridge practice codes and practice prescription data
find_prescriptions_cost(cambridge_practice_codes, practice_prescription_data)
```

Out[22]:

	practice_code	cost
0	C83026	297645.49
1	D81001	65873.09
2	D81002	90156.76
3	D81003	71266.77
4	D81004	111906.74
5	D81005	63413.35
6	D81007	111265.22
7	D81008	251141.55
8	D81009	84364.61
9	D81010	150453.19
10	D81011	151332.17
11	D81012	101104.92

	practice_code	cost
12	D81013	80673.77
13	D81014	187783.98
14	D81015	98328.83
15	D81016	101374.52
16	D81017	49332.05
17	D81019	51758.48
18	D81020	165236.00
19	D81021	133227.46
20	D81022	101508.92
21	D81023	151727.40
22	D81025	90312.69
23	D81026	304277.06
24	D81027	83875.42
25	D81028	136120.46
26	D81029	129324.76
27	D81030	122048.86
28	D81032	140715.93
29	D81033	32286.65
...	...	...
97	Y00673	5202.31
98	Y01778	918.37
99	Y02051	11.85
100	Y02769	40023.85
101	Y02888	496.60
102	Y02978	13.92
103	Y02990	2.34
104	Y02991	123.35
105	Y02992	428.14
106	Y02993	334.93
107	Y02994	1594.83
108	Y03041	66.42
109	Y03272	878.02
110	Y03454	7306.90
111	Y03472	1250.18
112	Y03550	2856.74
113	Y04145	22.98
114	Y04184	39.24
115	Y04185	311.40
116	Y04186	844.32
117	Y04208	65.99
118	Y04210	55.91
119	Y04212	10.97
120	Y04214	2.38
121	Y04602	1533.30
122	Y05140	1856.65
123	Y05173	93.76
124	Y05176	35.97
125	Y05474	363.41
126	Y05839	45.38

127 rows × 2 columns

127 rows × 2 columns

In [23]:

```
# CAMBRIDGE: the top 10 most frequent drugs prescribed with parameters as cambridge practice codes
and practice prescription data
find_top_10_frequent_drugs(cambridge_practice_codes, practice_prescription_data)
```

Out[23]:

	practice_code	name	quantity
0	C83026	Fresubin 2kcal_Drink (6 Flav)	103200
1	C83026	Fortisip Compact_Liq (8 Flav)	99000
2	C83026	Fortisip Bottle_Liq (8 Flav)	66400
3	C83026	Lactulose_Soln 3.1g-3.7g/5ml	58700
4	C83026	Nutrison Pack_Energy	58500
5	C83026	Paracet_Tab 500mg	58346
6	C83026	Dermol 500_Lot	55500
7	C83026	Gaviscon Advance_Liq (Aniseed) (Reckitt)	51550
8	C83026	Omeprazole_Cap E/C 20mg	46034
9	C83026	Fresubin Energy_Liq (8 Flav)	44000
10	C83026	Fortijuce_Liq (7 Flav)	38400
11	C83026	Metformin HCl_Tab 500mg	38283
12	C83026	Fresubin Jucy_Drink (5 Flav)	36000
13	C83026	KetoCal 4:1_LQ Liq (Unflav)	33400
14	C83026	Fortisip Compact Protein_Liq (8 Flav)	31000
15	C83026	Pur_Water	30000
16	C83026	Atorvastatin_Tab 20mg	28125
17	C83026	Nutrison Pack_Energy M/Fibre	28000
18	C83026	Diprobase_Crm	26100
19	C83026	Tentrini Pack_G/F M/Fibre Feed	24000
20	C83026	Aspirin Disper_Tab 75mg	21235
21	C83026	Fortisip Compact Protein_S/Pack Liq	21000
22	C83026	Fresubin Protein Energy_Drink (5 Flav)	20800
23	C83026	Tramadol HCl_Cap 50mg	20436
24	C83026	Levothyrox Sod_Tab 100mcg	20302
25	C83026	Co-Codamol_Tab 30mg/500mg	20086
26	C83026	Metformin HCl_Tab 500mg M/R	19222
27	C83026	Levothyrox Sod_Tab 25mcg	18400
28	C83026	Simvastatin_Tab 40mg	18251
29	C83026	Amlodipine_Tab 5mg	18250
...	...	...	...
135762	Y05474	Solifenacin_Tab 5mg	28
135763	Y05474	Gentamicin/Hydrocort Acet_Ear Dps 0.3/1%	20
135764	Y05474	Bactroban_Oint 2%	15
135765	Y05474	Aspirin_Tab 75mg	14
135766	Y05474	Mirabegron_Tab 50mg M/R	14
135767	Y05474	Chloramphen_Eye Dps 0.5%	10
135768	Y05474	Acular_Ophth Soln 0.5%	5
135769	Y05474	Apraclonidine_Eye Dps 5mg/ml	5
135770	Y05474	Azopt_Eye Dps 10mg/ml	5
135771	Y05474	Ciloxan_Eye Dps 0.3%	5

	practice_code	name	quantity
135772	Y05474	Dexameth_Eye Dps 0.1%	5
135773	Y05474	Iopidine_Eye Dps 5mg/ml	5
135774	Y05474	Maxidex_Eye Dps 0.1%	5
135775	Y05474	Pred Fte_Eye Dps 1%	5
135776	Y05474	Otomize_Ear Spy 5ml	3
135777	Y05474	Nasonex_Aq N/Spy 50mcg (140 D)	1
135778	Y05474	Phos Enem_(For B) 128ml Long Tube	1
135779	Y05474	Sterimar Isotonic Nsl Spy 50ml	1
135780	Y05839	Dermol 500_Lot	500
135781	Y05839	Morph Sulf_Oral Soln 10mg/5ml	300
135782	Y05839	Pregabalin_Cap 25mg	56
135783	Y05839	Biotene Oralbalance Gel	50
135784	Y05839	Salivix Pastil (App)	50
135785	Y05839	Hydrocort Acet/Fusidic Acid_Crm 1%/2%	30
135786	Y05839	Hydrocort_Crm 1%	30
135787	Y05839	Loperamide HCl_Cap 2mg	30
135788	Y05839	Fluclox Sod_Cap 500mg	28
135789	Y05839	Omeprazole_Cap E/C 40mg	28
135790	Y05839	Dexameth_Tab 500mcg	14
135791	Y05839	Levomeprom Mal_Tab 25mg	14

135792 rows × 3 columns

In [24]:

```
# CAMBRIDGE: the bottom 10 less frequent drugs prescribed with parameters as cambridge practice co
des and practice prescription data
find_least_10_frequent_drugs(cambridge_practice_codes, practice_prescription_data)
```

Out[24]:

	practice_code	name	quantity
0	C83026	Picato_Gel 500mcg/g	0
1	C83026	3m Health Care_Cavilon No Sting 3ml Barr	1
2	C83026	Acapella Oscillating Positive Expiratory	1
3	C83026	Activa Leg Ulcer Hose Kit Lge Clsd Toe C	1
4	C83026	Activon Tulle 5cm x 5cm Manuka Honey Ste	1
5	C83026	Aderma Dermal Pad 10 x 10 x 1.2cm Sheet	1
6	C83026	AirFluSal Forspiro_Inh 500/50mcg (60D)	1
7	C83026	Airomir_Inha 100mcg (200 D)	1
8	C83026	Ambirix_Vac 720u/20mcg/ml 1ml Pfs	1
9	C83026	Aspen_Sorbaderm No-Sting Barrier Film Sp	1
10	C83026	Atimos Modulite_Inh 12mcg (100D)	1
11	C83026	Autopen Classic 3ml 2u (2-42u) Hypod Reu	1
12	C83026	Avaxim_Vac 320u/ml 0.5ml Pfs	1
13	C83026	Beclomet Diprop_Inha B/A100mcg(200 D)CFF	1
14	C83026	Bettamousse_Foam Aero 0.1% 100g	1
15	C83026	BioXtra Dry Mth Gel Mth Spy 50ml (App)	1
16	C83026	Budesonide_Pdr For Inh 100mcg (200 D)	1
17	C83026	Budesonide_Pdr For Inh 200mcg (100 D)	1
18	C83026	Cetraben Crm 1050g	1
19	C83026	Cetraben Crm 150g	1
20	C83026	Clement Clarke AirZone Stnd Range Peak F	1

	practice_code	name	quantity
21	C83026	Clement Clarke Stnd Range Peak Flow Mete	1
22	C83026	Coloplast SpeediCath Compact Fle Size 8-	1
23	C83026	Coloplast_Brava Lubricating Deod 240ml	1
24	C83026	Coloplast_Brava Skin Barrier Spy 50ml	1
25	C83026	Depo-Medrone_Inj 40mg/ml 2ml Vl	1
26	C83026	Depo-Provera_Inj 150mg/ml 1ml Pfs	1
27	C83026	Difflam_P/Spy 0.15% 30ml	1
28	C83026	Easyhaler_Budesonide 100mcg (200 D)	1
29	C83026	Easyhaler_Budesonide 200mcg (200 D)	1
...	...	...	...
135762	Y05474	Bactroban_Oint 2%	15
135763	Y05474	Gentamicin/Hydrocort Acet_Ear Dps 0.3/1%	20
135764	Y05474	Aciclovir_Tab 400mg	28
135765	Y05474	Ciprofloxacin_Tab 500mg	28
135766	Y05474	Fexofenadine HCl_Tab 180mg	28
135767	Y05474	Fybogel_Gran Sach 3.5g Orange G/F S/F	28
135768	Y05474	Movicol_Pdr Sach 13.8g (Lem & Lim)	28
135769	Y05474	Omeprazole_Cap E/C 20mg	28
135770	Y05474	Solifenacin_Tab 5mg	28
135771	Y05474	Rectogesic_Oint 0.4%	30
135772	Y05474	Gabapentin_Cap 100mg	31
135773	Y05474	Doxycycline Hyclate_Cap 100mg	35
135774	Y05474	Methocarbamol_Tab 750mg	56
135775	Y05474	Diltiazem HCl_Crm 2%	60
135776	Y05474	Amitriptyline HCl_Tab 10mg	70
135777	Y05474	Fluclox Sod_Cap 500mg	96
135778	Y05474	Gaviscon Advance_Liq (Aniseed) (Reckitt)	150
135779	Y05474	Prednisolone_Tab 5mg	518
135780	Y05839	Dexameth_Tab 500mcg	14
135781	Y05839	Levomeprom Mal_Tab 25mg	14
135782	Y05839	Fluclox Sod_Cap 500mg	28
135783	Y05839	Omeprazole_Cap E/C 40mg	28
135784	Y05839	Hydrocort Acet/Fusidic Acid_Crm 1%/2%	30
135785	Y05839	Hydrocort_Crm 1%	30
135786	Y05839	Loperamide HCl_Cap 2mg	30
135787	Y05839	Biotene Oralbalance Gel	50
135788	Y05839	Salivix Pastil (App)	50
135789	Y05839	Pregabalin_Cap 25mg	56
135790	Y05839	Morph Sulf_Oral Soln 10mg/5ml	300
135791	Y05839	Dermol 500_Lot	500

135792 rows × 3 columns

In [25]:

```
# cardiovascular: total number of prescriptions and their total actual cost across all practices f
or drugs
find_prescriptions_count_and_cost_cardiovascular(practice_prescription_data)
```

Out[25]:

	quantity	cost
0	865947844	7.509311e+07

	quantity	cost
0	685947644	7.505511e+07

In [26]:

```
# antidepressants: total number of prescriptions and their total actual cost across all practices
for drugs
find_prescriptions_count_and_cost_antidepressants(practice_prescription_data)
```

Out[26]:

	quantity	cost
0	146956907	1.241988e+07

In [28]:

```
# visualize the monthly total spending per registered patients using a scatterplot and provide a t
rend line
all_practice_codes = sql('select practice_code from practice_location_data')
```

In [29]:

```
number_of_patients = find_patients_registered(all_practice_codes, practice_patient_data)
#identifying patients registered using all practice codes, and practice patient data as parameters
, and defined as number of patients for plot
```

In [30]:

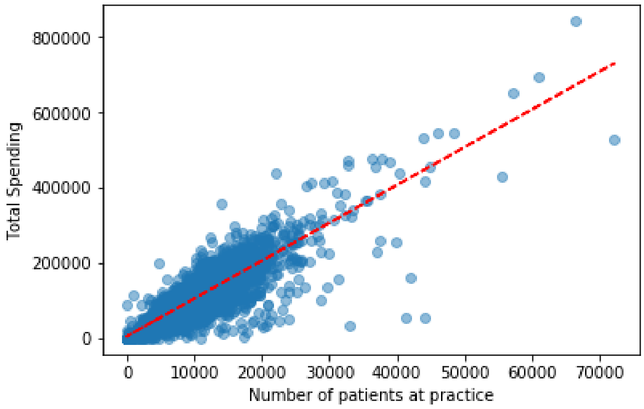
```
#identifying total cost for plot
total_cost = find_prescriptions_cost(all_practice_codes, practice_prescription_data)
```

In [31]:

```
plot_data = sql("select p.number_of_patients as number_of_patients, c.cost as cost, (c.cost / p.nu
mber_of_patients) as relative_cost from all_practice_codes a left join number_of_patients p on a.p
ractice_code = p.practice_code left join total_cost c on c.practice_code = a.practice_code where c
.cost is not null and p.number_of_patients is not null")
```

In [32]:

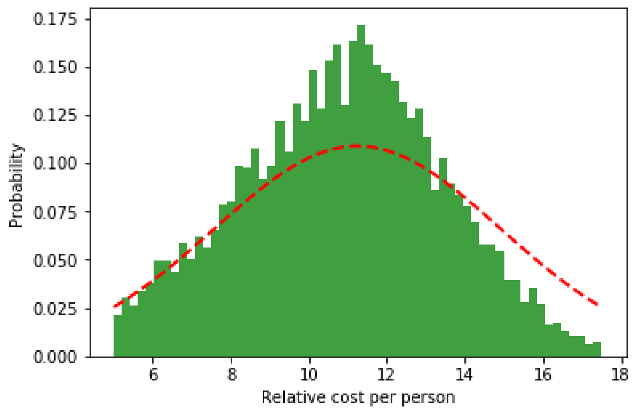
```
xs = plot_data.number_of_patients #plot number of patients on x axis
ys = plot_data.cost #plot total spending/data cost on y axis
plt.scatter(xs, ys, alpha=0.5)
z = np.polyfit(xs, ys, 1)
p = np.poly1d(z)
plt.xlabel('Number of patients at practice')
plt.ylabel('Total Spending')
plt.plot(xs,p(xs),"r--")
plt.show()
```



In [33]:

```
# generate a histogram for relative spending for all practices and fit a Gaussian (normal) curve
n, bins, patches = plt.hist(plot_data.relative_cost, 60, range=(5, 17.5), density=True, facecolor='green', alpha=0.75)
(mu, sigma) = norm.fit(bins)
y = mlab.normpdf(bins, mu, sigma)
l = plt.plot(bins, y, 'r--', linewidth=2)
plt.xlabel('Relative cost per person')
plt.ylabel('Probability')
plt.show()
```

```
C:\Users\imash\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: MatplotlibDeprecationWarning: scipy.stats.norm.pdf
  after removing the cwd from sys.path.
```



In [9]:

```
#Assignment B


def find_population_and_deaths_neoplasms(population_data, mortality_data, country_code):
  return sql('select p.pop1 as population, sum(m.deaths1) as deaths from population_data p left join mortality_data m on p.country = m.country where m.year=2010 and p.year=2010 and m.cause >= "C00" and m.cause <= "D48" and p.country = ' + str(country_code) + ' group by population')

#Def defines a funtion, in this case function is defined as to find population and deaths neoplasms, with the parameters being population data, mortality data and country code
#This code is using and Pandas and PandaSQL
#The return statement causes your find population and deaths neoplasms function to exit and hand back a value to its caller

def find_population_and_deaths(population_data, mortality_data, country_code):
  return sql('select p.pop1 as population, sum(m.deaths1) as deaths from population_data p left join mortality_data m on p.country = m.country where m.year=2010 and p.year=2010 and p.country = ' + str(country_code) + ' group by population')

#Def defines a funtion, in this case function is defined as to find population and deaths, with the parameters being population data, mortality data and country code
#This code is using and Pandas and PandaSQL
#The return statement causes your find population and deaths function to exit and hand back a value to its caller


def find_country_code(country_code_data, country_name):
  return sql('select country from country_code_data where name = "' + country_name + '"').country[0]

#Def defines a funtion, in this case function is defined as to find country code, with the parameters being country code data and country name
#This code is using and Pandas and PandaSQL
#The return statement causes your find country code function to exit and hand back a value to its caller


low_memory=False
```

In [10]:

```
mortality_data = pandas.read_csv('Morticd10') # combine part 1 and part 2 mortality datan files
manually beforehand on disk then import mortality data from WHO as panda and naming it as mortalit
y_data
population_data = pandas.read_csv('pop') #importing population data from WHO as panda and naming i
t as population_data
country_code_data = pandas.read_csv('country_codes') #importing country codes data from WHO as
panda and naming it as country_code_data
iceland_country_code = find_country_code(country_code_data, 'Iceland') #Identifying country code u
sing Panda and PandaSQL
italy_country_code = find_country_code(country_code_data, 'Italy')
new_zealand_country_code = find_country_code(country_code_data, 'New Zealand')
australia_country_code = find_country_code(country_code_data, 'Australia')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3020: DtypeWarning:
Columns
(0,1,2,3,4,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36
38) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [11]:

```
# identifyin iceland population and the total number of deaths using pandasql
find_population_and_deaths(population_data, mortality_data, iceland_country_code)
```

Out[11]:

	population	deaths
0	158070.0	4038
1	159971.0	4038

In [12]:

```
# identifying italy population and the total number of deaths using pandasql
find_population_and_deaths(population_data, mortality_data, italy_country_code)
```

Out[12]:

	population	deaths
0	29350339.0	1169230
1	31133047.0	1169230

In [14]:

```
# identifying new_zealand population and the total number of deaths using pandasql
find_population_and_deaths(population_data, mortality_data, new_zealand_country_code)
```

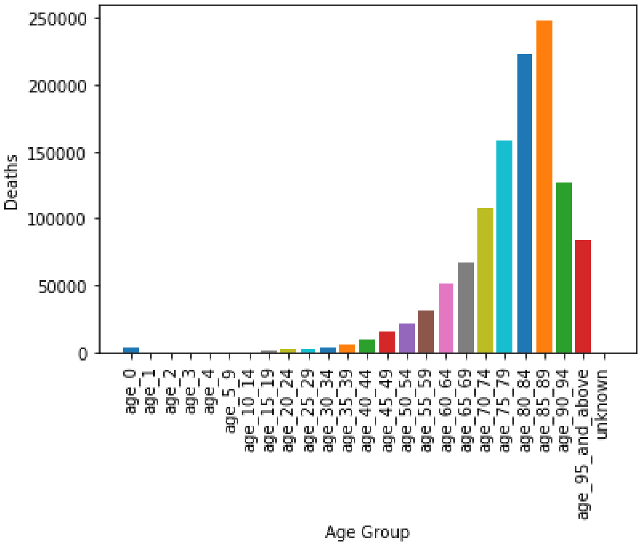
Out[14]:

	population	deaths
0	2144390.0	57298
1	2222970.0	57298

In [15]:

```
# What was the distribution of deaths (all causes, all years) by age group in Italy
plot_data = sql('select sum(deaths2) as age_0, sum(deaths3) as age_1, sum(deaths4) as age_2, sum(d
eaths5) as age_3, sum(deaths6) as age_4, sum(deaths7) as age_5_9, sum(deaths8) as age_10_14, sum(d
eaths9) as age_15_19, sum(deaths10) as age_20_24, sum(deaths11) as age_25_29, sum(deaths12) as
age_30_34, sum(deaths13) as age_35_39, sum(deaths14) as age_40_44, sum(deaths15) as age_45_49,
sum(deaths16) as age_50_54, sum(deaths17) as age_55_59, sum(deaths18) as age_60_64, sum(deaths19)
as age_65_69, sum(deaths20) as age_70_74, sum(deaths21) as age_75_79, sum(deaths22) as age_80_84,
sum(deaths23) as age_85_89, sum(deaths24) as age_90_94, sum(deaths25) as age_95_and_above,
sum(deaths26) as unknown from mortality_data where year=2010 and country = ' +
str(italy_country_code))
for age_group in plot_data:
    plt.bar(age_group, plot_data[age_group])
plt.ylabel('Deaths')
plt.xlabel('Age Group')
plt.xticks(rotation=90)
plt.show()

#below is visualisation of distribution of death by age group in italy.
```



In [16]:

```
# Generate a table with the cause of death, the number of deaths, and the proportion of overall de
aths
table_data = sql('select cause as cause, sum(deaths1) as deaths from mortality_data where cause >=
"C00" and cause <= "D48" and year = 2010 and country = ' + str(italy_country_code) + ' group by
cause order by deaths DESC')
total_deaths = float(sql('select sum(deaths) as total_deaths from table_data').total_deaths)
table_data = sql('select *, (deaths / ' + str(total_deaths) + ') as proportion from table_data')
```

In [18]:

```
print(table_data) # print table_data for table
```

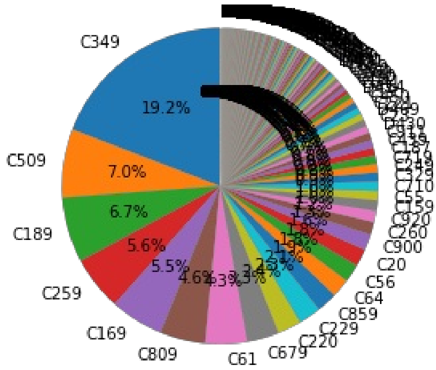
```
     cause  deaths  proportion
0     C349   33416    0.191600
1     C509   12231    0.070130
2     C189   11638    0.066730
3     C259    9683    0.055520
4     C169    9523    0.054603
5     C809    8036    0.046077
6      C61    7509    0.043055
7     C679    5675    0.032539
8     C220    4257    0.024409
9     C229    4018    0.023038
10    C859    3660    0.020986
11     C64    3361    0.019271
12     C56    3193    0.018308
13     C20    3101    0.017780
14    C900    2831    0.016232
15    C260    2240    0.012844
16    C920    2067    0.011852
17    C159    1823    0.010453
18     C55    1691    0.009696
19    C710    1687    0.009673
20    C329    1591    0.009122
21    C249    1532    0.008784
22    C719    1482    0.008497
23    C187    1406    0.008062
24    C439    1377    0.007895
25    C911    1317    0.007551
26    D430    1307    0.007494
27     C23    1199    0.006875
28    D469    1192    0.006835
```

```
29   C221    1139    0.006531
..    ...     ...         ...
410  C811       1    0.000006
411  C812       1    0.000006
412  C820       1    0.000006
413  C827       1    0.000006
414  C917       1    0.000006
415  C922       1    0.000006
416  C960       1    0.000006
417  D049       1    0.000006
418  D075       1    0.000006
419  D125       1    0.000006
420  D130       1    0.000006
421  D152       1    0.000006
422  D160       1    0.000006
423  D164       1    0.000006
424  D165       1    0.000006
425  D171       1    0.000006
426  D172       1    0.000006
427  D179       1    0.000006
428  D213       1    0.000006
429   D24       1    0.000006
430  D331       1    0.000006
431   D34       1    0.000006
432  D351       1    0.000006
433  D369       1    0.000006
434  D409       1    0.000006
435  D417       1    0.000006
436  D421       1    0.000006
437  D433       1    0.000006
438  D446       1    0.000006
439  D448       1    0.000006

[440 rows x 3 columns]
```

In [19]:

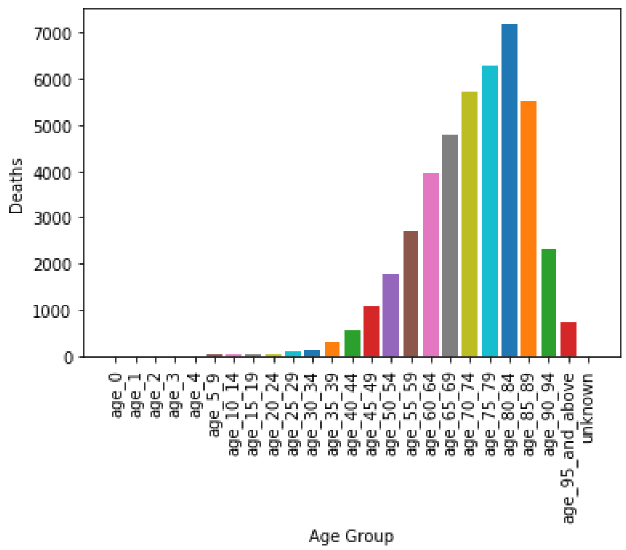
```
# Generate a pie chart to visualize the proportion of deaths
_fig1, ax1 = plt.subplots()
ax1.pie(table_data.deaths, labels=table_data.cause, autopct='%1.1f%%', startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



In [20]:

```
# Are there differences by age group for deaths from Neoplasms (C00-D48) in Australia for 2010
plot_data = sql('select sum(deaths2) as age_0, sum(deaths3) as age_1, sum(deaths4) as age_2, sum(d
eaths5) as age_3, sum(deaths6) as age_4, sum(deaths7) as age_5_9, sum(deaths8) as age_10_14, sum(d
eaths9) as age_15_19, sum(deaths10) as age_20_24, sum(deaths11) as age_25_29, sum(deaths12) as
age_30_34, sum(deaths13) as age_35_39, sum(deaths14) as age_40_44, sum(deaths15) as age_45_49,
sum(deaths16) as age_50_54, sum(deaths17) as age_55_59, sum(deaths18) as age_60_64, sum(deaths19)
as age_65_69, sum(deaths20) as age_70_74, sum(deaths21) as age_75_79, sum(deaths22) as age_80_84,
sum(deaths23) as age_85_89, sum(deaths24) as age_90_94, sum(deaths25) as age_95_and_above,
sum(deaths26) as unknown from mortality_data where cause >= "C00" and cause <= "D48" and year=2010
and country = ' + str(australia_country_code))
for age_group in plot_data:
  plt.bar(age_group, plot_data[age_group])
plt.ylabel('Deaths')
plt.xlabel('Age Group')
plt.xticks(rotation=90)
plt.xticks(rotation=90)
plt.show()


#Identify the top five age groups in Australia dying with a Neoplasms cause of death.
#looking at graph, top age groups are from most to least, ages 80-84, ages 75-79, ages 70-74, and
ages 85-89.
```



In [21]:

```
find_population_and_deaths_neoplasms(population_data, mortality_data, australia_country_code)
#using pandasql, finding population and deaths neoplasms using population data, mortality data and
australia country code as parameters
```

Out[21]:

	population	deaths
0	11100244.0	43276
1	11197271.0	43276

In [22]:

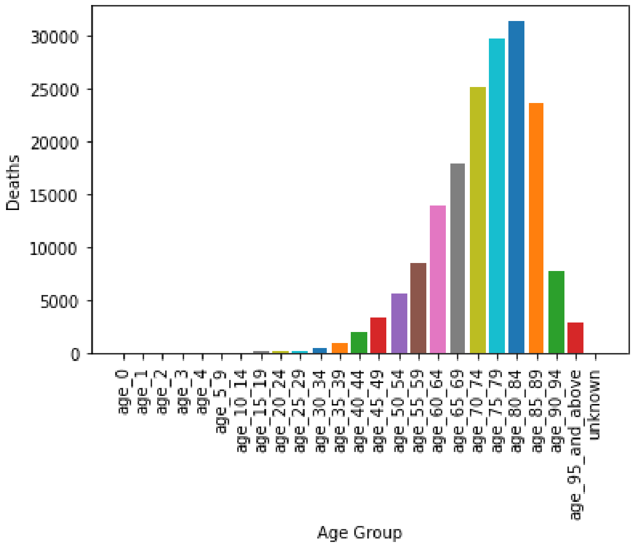
```
find_population_and_deaths_neoplasms(population_data, mortality_data, italy_country_code)
#using pandasql, finding population and deaths neoplasms using population data, mortality data and
italy country code as parameters
```

Out[22]:

	population	deaths
0	29350339.0	174405
1	31133047.0	174405

In [23]:

```
# Are there differences by age group for deaths from Neoplasms (C00-D48) in Italy for 2010
plot_data = sql('select sum(deaths2) as age_0, sum(deaths3) as age_1, sum(deaths4) as age_2, sum(d
eaths5) as age_3, sum(deaths6) as age_4, sum(deaths7) as age_5_9, sum(deaths8) as age_10_14, sum(d
eaths9) as age_15_19, sum(deaths10) as age_20_24, sum(deaths11) as age_25_29, sum(deaths12) as
age_30_34, sum(deaths13) as age_35_39, sum(deaths14) as age_40_44, sum(deaths15) as age_45_49,
sum(deaths16) as age_50_54, sum(deaths17) as age_55_59, sum(deaths18) as age_60_64, sum(deaths19)
as age_65_69, sum(deaths20) as age_70_74, sum(deaths21) as age_75_79, sum(deaths22) as age_80_84,
sum(deaths23) as age_85_89, sum(deaths24) as age_90_94, sum(deaths25) as age_95_and_above,
sum(deaths26) as unknown from mortality_data where cause >= "C00" and cause <= "D48" and year=2010
and country = ' + str(italy_country_code))
for age_group in plot_data:
  plt.bar(age_group, plot_data[age_group])
plt.ylabel('Deaths')
plt.xlabel('Age Group')
plt.xlabel('Age Group')
plt.xticks(rotation=90)
plt.show()
#Identify the top five age groups in Italy dying with a Neoplasms cause of death.
#looking at graph, top age groups are from most to least, ages 80-84, ages 75-79, ages 70-74, and
ages 85-89, so same as Australia.
```



In [ ]:

In [ ]: