Project overview:
We are a Private Equity firm looking to create an insurance company. Our goal is to find the largest market opportunities by state and area type based on diversity of services and quantity of providers. We are also looking to find a reasonable network of providers to cover in a few specialties of interest for each state.

1. We investigated the diversity of medicare-accepted services available to populations living in different area types (i.e. metropolitan, rural…)
2. We also found the three most diverse area types to explore where the greatest opportunities for expansion would be
3. We investigated the quantity of medicare-accepted services available to populations living in different area types
4. We investigated the proportion of male versus female doctors who take medicare, and compared these proportions to the total number of male and female doctors in the US
    a. Total number of female doctors versus male doctors who take medicare
    b. Proportion of female doctors who take medicare out of the total number of female practicing physicians versus proportion of male doctors who take medicare out of the total number of male practicing physicians
        i. According to Becker Hospital Review, there are 366,759 (36%) practicing female physicians and 652,017 (64%) practicing male physicians, for a total of 1,018,776 practicing physicians in the US total
5. We investigated the proportion of individuals versus corporations who take medicare, so as to further understand our target market
6. We investigated the diversity of medicare-accepted services available in each state, and found the top 10 most diverse states in regards to number of unique service types, so as to better understand the opportunities for the highest impact states when making a referral network
7. We investigated the quantity of medicare-accepted services available in each state to further understand our target market with the greatest opportunities
8. Another question that we wanted to answer: We were looking to create a referral network in each state for internal medicine with cardiology doctors to map out coverage.
    - We made a google storage bucket and got the link to this file, in order to access it in mapper_init() to pair up the dataset
    - This code is in the mapper_pairs.py document

To provide a report to the PE firm, we visualized our findings with barplots and pie charts. Additionally, we stored our data in GoogleCloud to work with large amounts of data efficiently. Analogously to Lab3, we ran all over our map reduce code through Google Cloud. Additionally, we used a Google Storage Bucket to store our dataset.

Extraneous files:

- We also attempted to create a referral network based on zip codes, but unfortunately, every library includes too many “NaN”s (does not have information about many zip codes) to obtain the latitude and longitude for each zip code
    - However, the code (which otherwise works) is included in the git repository

1. **Unique kinds of providers by area type (area_providers.py)**
    a. **We are investigating the diversity of services available to populations living in different area types**

```
'''
Analyzing number of unique provider types available in an area
'''
from mrjob.job import MRJob
import re
class MRAreaProviders(MRJob):
   def mapper(self, _, line):

       COMMA_MATCHER = re.compile(r",(?=(?:[^\"']*[\"'][^\"']*[\"'])*[^\"']*$)")
       line_cols = COMMA_MATCHER.split(line)

       if len(line_cols) > 13 and "flow" in line_cols[14]:
           area = line_cols[14]
           provider_type = line_cols[16]
       else:
           area = ""
           provider_type = ""

       if area:
           yield (area, provider_type)

   def combiner(self, area, provider_type):
       types = set(provider_type)
       for unq_type in types:
           yield (area, unq_type)


   def reducer(self, area, provider_type):
       types = set(provider_type)
       yield area, len(types)


if __name__ == '__main__':
   MRAreaProviders.run()
```

Output:
\"Secondary flow 30% to <50% to a larger urbanized area of 50,000 and greater\"" 59
"\"Metropolitan area low commuting: primary flow 10% to <30% to a urbanized area of 50,000 and greater\"" 23
"\"Micropolitan area core: primary flow within an urban cluster of 10,000 to 49,999\"" 69
"\"Metropolitan area core: primary flow within an urbanized area of 50,000 and greater\"" 80
"\"Metropolitan area high commuting: primary flow 30% or more to a urbanized area of 50,000 and greater\"" 58
"\"Small town high commuting: primary flow 30% or more to a urban cluster of 2,500 to 9,999\"" 20
"\"Small town low commuting: primary flow 10% to <30% to a urban cluster of 2,500 to 9,999\"" 14

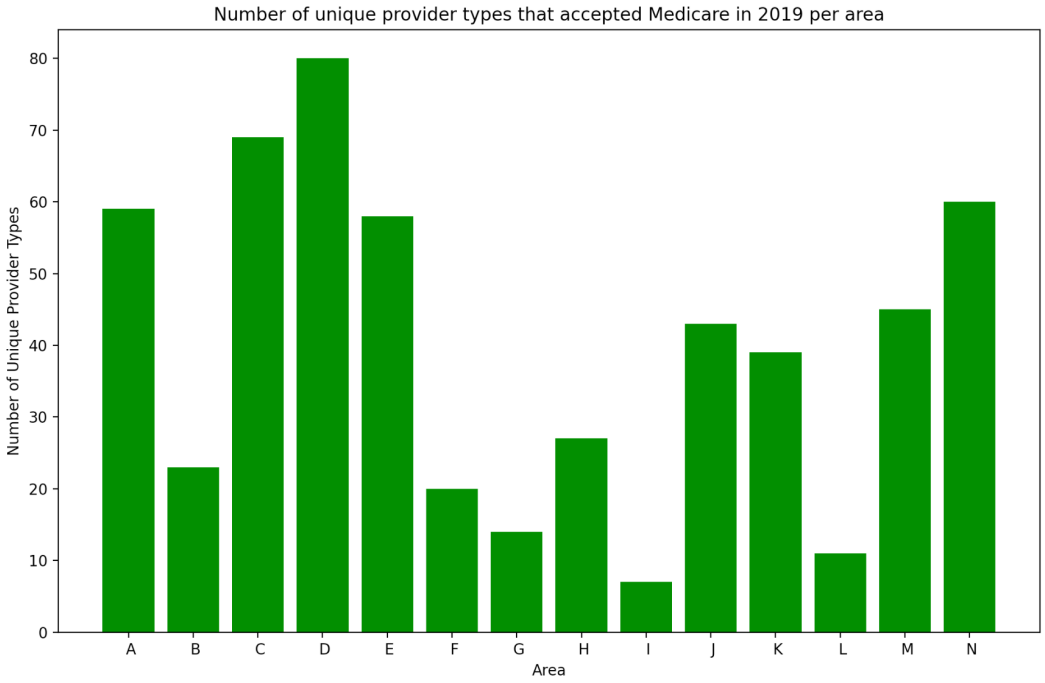
"\"Secondary flow 30% to <50% to a urban cluster of 10,000 to 49,999\"" 27
"\"Secondary flow 30% to <50% to a urban cluster of 2,500 to 9,999\"" 7
"\"Secondary flow 30% to <50% to a urbanized area of 50,000 and greater\"" 43
"\"Micropolitan high commuting: primary flow 30% or more to a urban cluster of 10,000 to 49,999\"" 39
"\"Micropolitan low commuting: primary flow 10% to <30% to a urban cluster of 10,000 to 49,999\"" 11
"\"Rural areas: primary flow to a tract outside a urbanized area of 50,000 and greater or UC\"" 45
"\"Small town core: primary flow within an urban cluster of 2,500 to 9,999\"" 60

Processed Output (area_providers.csv):

,,
"Secondary flow 30% to <50% to a larger urbanized area of 50000 and greater", A, 59
"Metropolitan area low commuting: primary flow 10% to <30% to a urbanized area of 50000 and greater", B, 23
"Micropolitan area core: primary flow within an urban cluster of 10000 to 49999", C, 69
"Metropolitan area core: primary flow within an urbanized area of 50000 and greater", D, 80
"Metropolitan area high commuting: primary flow 30% or more to a urbanized area of 50000 and greater", E, 58
"Small town high commuting: primary flow 30% or more to a urban cluster of 2500 to 9999", F, 20
"Small town low commuting: primary flow 10% to <30% to a urban cluster of 2500 to 9999", G, 14
"Secondary flow 30% to <50% to a urban cluster of 10000 to 49999", H, 27
"Secondary flow 30% to <50% to a urban cluster of 2500 to 9999", I, 7
"Secondary flow 30% to <50% to a urbanized area of 50000 and greater", J, 43
"Micropolitan high commuting: primary flow 30% or more to a urban cluster of 10000 to 49999", K, 39
"Micropolitan low commuting: primary flow 10% to <30% to a urban cluster of 10000 to 49999", L, 11
"Rural areas: primary flow to a tract outside a urbanized area of 50000 and greater or UC", M, 45
"Small town core: primary flow within an urban cluster of 2500 to 9999", N, 60

Barchart code for area_providers.py: (barchart_area_providers.py)

```
import matplotlib.pyplot as plt
import pandas as pd


data = pd.read_csv('area_providers.csv')
 df = pd.DataFrame(data)
 X = list(df.iloc[:, 1])
Y = list(df.iloc[:, 2])
 plt.bar(X, Y, color='g')
plt.title("Number of unique provider types that accepted Medicare in 2019 per area")
plt.xlabel("Area")
plt.ylabel("Number of Unique Provider Types")
plt.show()
```



Key

A: Secondary flow 30% to <50% to a larger urbanized area of 50000 and greater (59)
B: Metropolitan area low commuting: primary flow 10% to <30% to a urbanized area of 50000 and greater (23)
C: Micropolitan area core: primary flow within an urban cluster of 10000 to 49999 (69)
D: Metropolitan area core: primary flow within an urbanized area of 50000 and greater (80)
E: Metropolitan area high commuting: primary flow 30% or more to a urbanized area of 50000 and greater (58)
F: Small town high commuting: primary flow 30% or more to a urban cluster of 2500 to 9999 (20)
G: Small town low commuting: primary flow 10% to <30% to a urban cluster of 2500 to 9999 (14)
H: Secondary flow 30% to <50% to a urban cluster of 10000 to 49999 (27)
I: Secondary flow 30% to <50% to a urban cluster of 2500 to 9999 (7)
J: Secondary flow 30% to <50% to a urbanized area of 50000 and greater (43)
K: Micropolitan high commuting: primary flow 30% or more to a urban cluster of 10000 to 49999 (39)
L: Micropolitan low commuting: primary flow 10% to <30% to a urban cluster of 10000 to 49999 (11)
M: Rural areas: primary flow to a tract outside a urbanized area of 50000 and greater or UC (45)
N: Small town core: primary flow within an urban cluster of 2500 to 9999 (60)

TOP 10 MOST DIVERSE AREAS BY SERVICE TYPE:

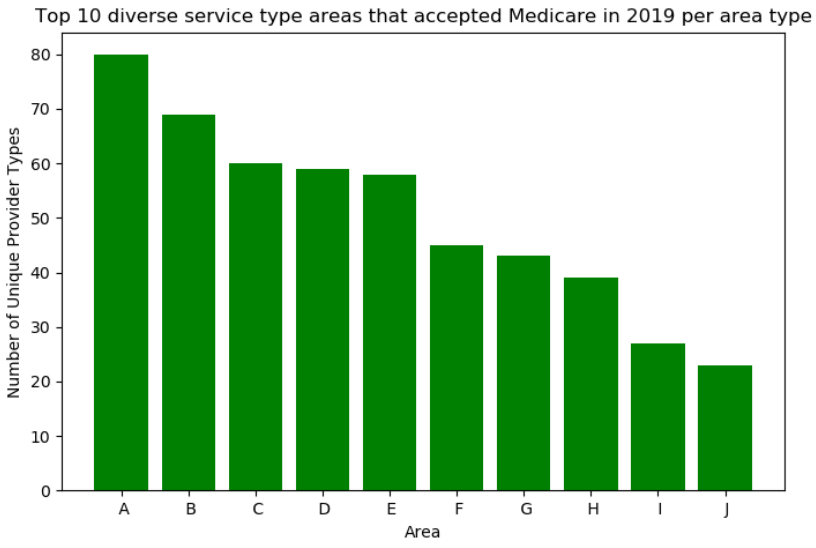
```
from mrjob.job import MRJob
import re
import heapq

# Question 1
class MRAreaProviders(MRJob):
   '''
   Finds top 10 areas for number of unique provider types available
   '''
   def mapper(self, _, line):
      COMMA_MATCHER = re.compile(r",(?=(?:[^\"']*[\"'][^\"']*[\"'])*[^\"']*$)")
      line_cols = COMMA_MATCHER.split(line)

      if len(line_cols) > 13 and "flow" in line_cols[14]:
          area = line_cols[14]
          provider_type = line_cols[16]
      else:
          area = ""
          provider_type = ""
      if area:
          yield (area, provider_type)
   def combiner(self, area, provider_type):
      types = set(provider_type)
      for unq_type in types:
          yield (area, unq_type)

   def reducer_init(self):
       self.h = []

   def reducer(self, area, provider_type):
      types = set(provider_type)
      num_types = len(types)
      if len(self.h) < 10:
          self.h.append((num_types, area))
          if len(self.h) == 10:
              heapq.heapify(self.h)
       else:
           min_count, _ = self.h[0]
           if num_types > min_count:
               heapq.heapreplace(self.h, (num_types, area))
```

```
    def reducer_final(self):
        self.h.sort(reverse=True)
        for num_types, area in self.h:
            yield area, num_types

if __name__ == '__main__':
    MRAreaProviders.run()
```

Output:

,,
Metropolitan area core: primary flow within an urbanized area of 50000 and greater, A, 80
Micropolitan area core: primary flow within an urban cluster of 10000 to 49999, B, 69
Small town core: primary flow within an urban cluster of 2500 to 9999, C, 60
Secondary flow 30% to <50% to a larger urbanized area of 50000 and greater, D, 59
Metropolitan area high commuting: primary flow 30% or more to a urbanized area of 50,000 and greater, E, 58
Rural areas: primary flow to a tract outside a urbanized area of 50000 and greater or UC, F, 45
Secondary flow 30% to <50% to a urbanized area of 50000 and greater, G, 43
Micropolitan high commuting: primary flow 30% or more to a urban cluster of 10000 to 49999, H, 39
Secondary flow 30% to <50% to a urban cluster of 10000 to 49999, I, 27
Metropolitan area low commuting: primary flow 10% to <30% to a urbanized area of 50000 and greater, J, 23

Barchart:



2. **Number of providers by area type (area_any_providers.py)**

a. **We are investigating the quantity of services available to populations living in different area types**

```
'''
Analyzing number of providers available in an area
'''

from mrjob.job import MRJob
import re

class MRAreaAnyProviders(MRJob):

   def mapper(self, _, line):
      COMMA_MATCHER = re.compile(r",(?=(?:[^\"']*[\"'][^\"']*[\"'])*[^\"']*$)")
      line_cols = COMMA_MATCHER.split(line)

      if len(line_cols) > 13 and "flow" in line_cols[14]:
          area = line_cols[14]
      else:
          area = ""
      if area:
          yield (area, 1)

   def combiner(self, area, counts):
       yield area, sum(counts)

   def reducer(self, area, counts):
       yield area, sum(counts)

if __name__ == '__main__':
   MRAreaAnyProviders.run()
```

Output:

"\"Metropolitan area core: primary flow within an urbanized area of 50,000 and greater\"" 221750
"\"Metropolitan area high commuting: primary flow 30% or more to a urbanized area of 50,000 and greater\"" 4241
"\"Metropolitan area low commuting: primary flow 10% to <30% to a urbanized area of 50,000 and greater\"" 215
"\"Micropolitan area core: primary flow within an urban cluster of 10,000 to 49,999\"" 13786
"\"Micropolitan high commuting: primary flow 30% or more to a urban cluster of 10,000 to 49,999\"" 658
"\"Micropolitan low commuting: primary flow 10% to <30% to a urban cluster of 10,000 to 49,999\"" 65
"\"Rural areas: primary flow to a tract outside a urbanized area of 50,000 and greater or UC\"" 1389
"\"Secondary flow 30% to <50% to a larger urbanized area of 50,000 and greater\"" 3131
"\"Secondary flow 30% to <50% to a urban cluster of 10,000 to 49,999\"" 164

"\"Secondary flow 30% to <50% to a urban cluster of 2,500 to 9,999\"" 21
"\"Secondary flow 30% to <50% to a urbanized area of 50,000 and greater\"" 1075
"\"Small town core: primary flow within an urban cluster of 2,500 to 9,999\"" 4426
"\"Small town high commuting: primary flow 30% or more to a urban cluster of 2,500 to 9,999\"" 190
"\"Small town low commuting: primary flow 10% to <30% to a urban cluster of 2,500 to 9,999\"" 65

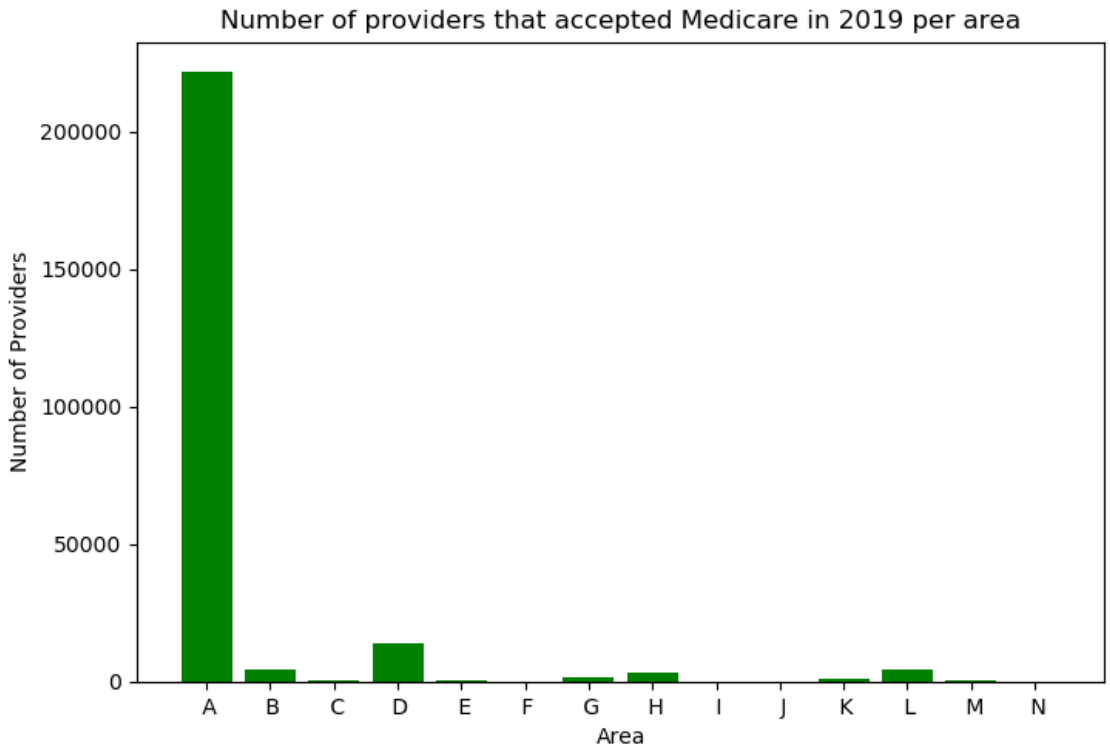
Processed Output (area_any_providers.csv):

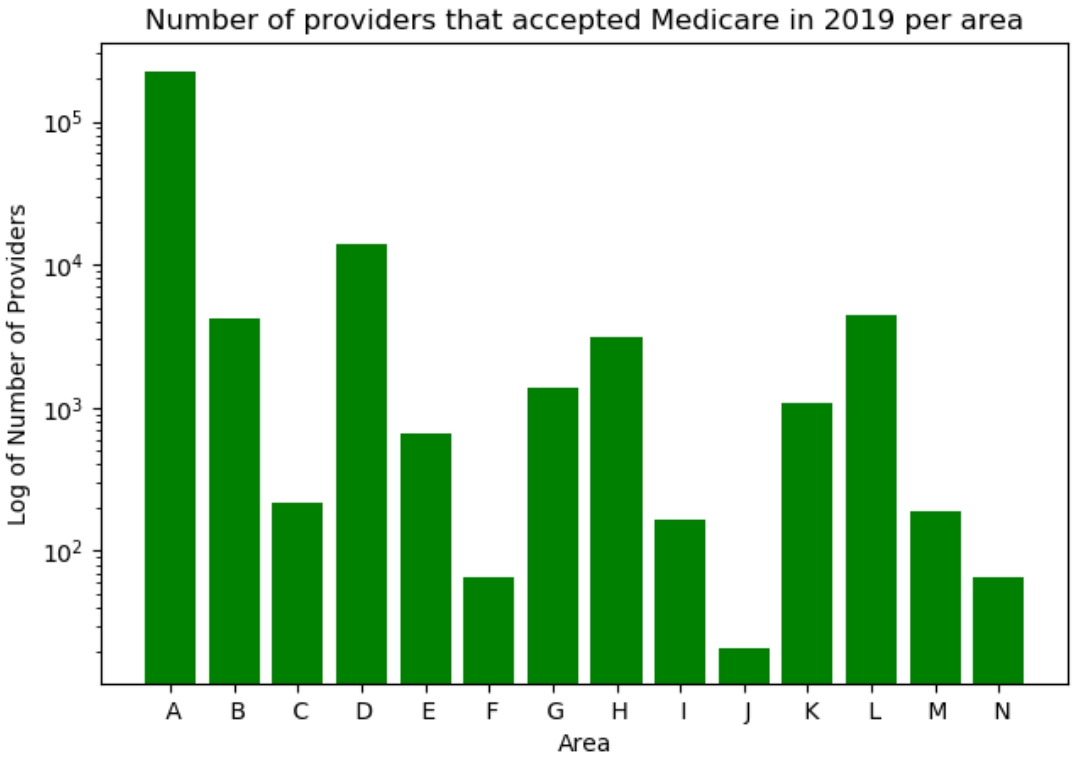
,,
Metropolitan area core: primary flow within an urbanized area of 50000 and greater, A, 221750
Metropolitan area high commuting: primary flow 30% or more to a urbanized area of 50000 and greater, B, 4241
Metropolitan area low commuting: primary flow 10% to <30% to a urbanized area of 50000 and greater, C, 215
Micropolitan area core: primary flow within an urban cluster of 10000 to 49999, D, 13786
Micropolitan high commuting: primary flow 30% or more to a urban cluster of 10000 to 49999, E, 658
Micropolitan low commuting: primary flow 10% to <30% to a urban cluster of 10000 to 49999, F, 65
Rural areas: primary flow to a tract outside a urbanized area of 50000 and greater or UC, G, 1389
Secondary flow 30% to <50% to a larger urbanized area of 50000 and greater, H, 3131
Secondary flow 30% to <50% to a urban cluster of 10000 to 49999, I, 164
Secondary flow 30% to <50% to a urban cluster of 2500 to 9999, J, 21
Secondary flow 30% to <50% to a urbanized area of 50000 and greater, K, 1075
Small town core: primary flow within an urban cluster of 2500 to 9999, L, 4426
Small town high commuting: primary flow 30% or more to a urban cluster of 2500 to 9999, M, 190
Small town low commuting: primary flow 10% to <30% to a urban cluster of 2500 to 9999, N, 65

Barchart code for area_any_providers.py: (barchart_area_any_providers.py)

```
'''
Makes a barchart for the number of providers per area type.
'''

import matplotlib.pyplot as plt
import pandas as pd

def bar_plot(log_scale):
    data = pd.read_csv('area_any_providers.csv')

    df = pd.DataFrame(data)
    X = list(df.iloc[:, 1])
    Y = list(df.iloc[:, 2])

    if log_scale:
        plt.yscale("log")

    plt.bar(X, Y, color='g')

    plt.title("Number of providers that accepted Medicare in 2019 per area")
    plt.xlabel("Area")
    if log_scale:
        plt.ylabel("Log of Number of Providers")
    else:
        plt.ylabel("Number of Providers")
```

```
plt.show()
```


Number of providers that accepted Medicare in 2019 per area
Number of Providers
200000
150000
100000
50000
0
A
B
C
D
E
F
G
H
I
J
K
L
M
N
Area



Key

A: Metropolitan area core: primary flow within an urbanized area of 50000 and greater (221750)
B: Metropolitan area high commuting: primary flow 30% or more to a urbanized area of 50000 and greater (4241)
C: Metropolitan area low commuting: primary flow 10% to <30% to a urbanized area of 50000 and greater (215)
D: Micropolitan area core: primary flow within an urban cluster of 10000 to 49999 (13786)
E: Micropolitan high commuting: primary flow 30% or more to a urban cluster of 10000 to 49999 (658)
F: Micropolitan low commuting: primary flow 10% to <30% to a urban cluster of 10000 to 49999 (65)
G: Rural areas: primary flow to a tract outside a urbanized area of 50000 and greater or UC (1389)
H: Secondary flow 30% to <50% to a larger urbanized area of 50000 and greater (3131)
I: Secondary flow 30% to <50% to a urban cluster of 10000 to 49999 (164)
J: Secondary flow 30% to <50% to a urban cluster of 2500 to 9999 (21)
K: Secondary flow 30% to <50% to a urbanized area of 50000 and greater (1075)
L: Small town core: primary flow within an urban cluster of 2500 to 9999 (4426)
M: Small town high commuting: primary flow 30% or more to a urban cluster of 2500 to 9999 (190)
N: Small town low commuting: primary flow 10% to <30% to a urban cluster of 2500 to 9999 (65)

3. **Proportion of men/women providers who take Medicare (gender_take_medicare.py)**
    a. **We are investigating if men or women are more likely to take Medicare, comparing the national averages of men/women who practice**

```
'''
Analyzing proportion of men/women who take Medicare
'''

from mrjob.job import MRJob
import re

class MRGenderTakeMedicare(MRJob):

   def mapper(self, _, line):
       COMMA_MATCHER = re.compile(r",(?=(?:[^\"']*[\"'][^\"']*[\"'])*[^\"']*$)")
       line_cols = COMMA_MATCHER.split(line)
       gender = line_cols[5]
       if gender == "M" or gender == "F":
           yield (gender, 1)

   def combiner(self, gender, counts):
       yield gender, sum(counts)

   def reducer(self, gender, counts):
       yield gender, sum(counts)

if __name__ == '__main__':
   MRGenderTakeMedicare.run()
```

Output:
"F" 76082
"M" 175632

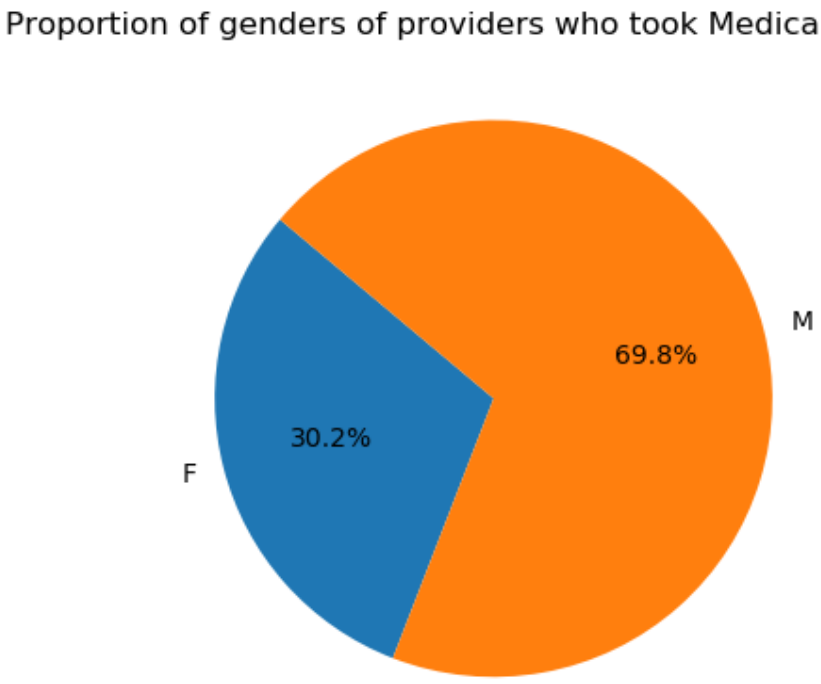
Processed Output (gender_take_medicare.csv):

,
F, 76082
M, 175632

Pie chart of gender_take_medicare.py (pichart_gender_take_medicare.py):

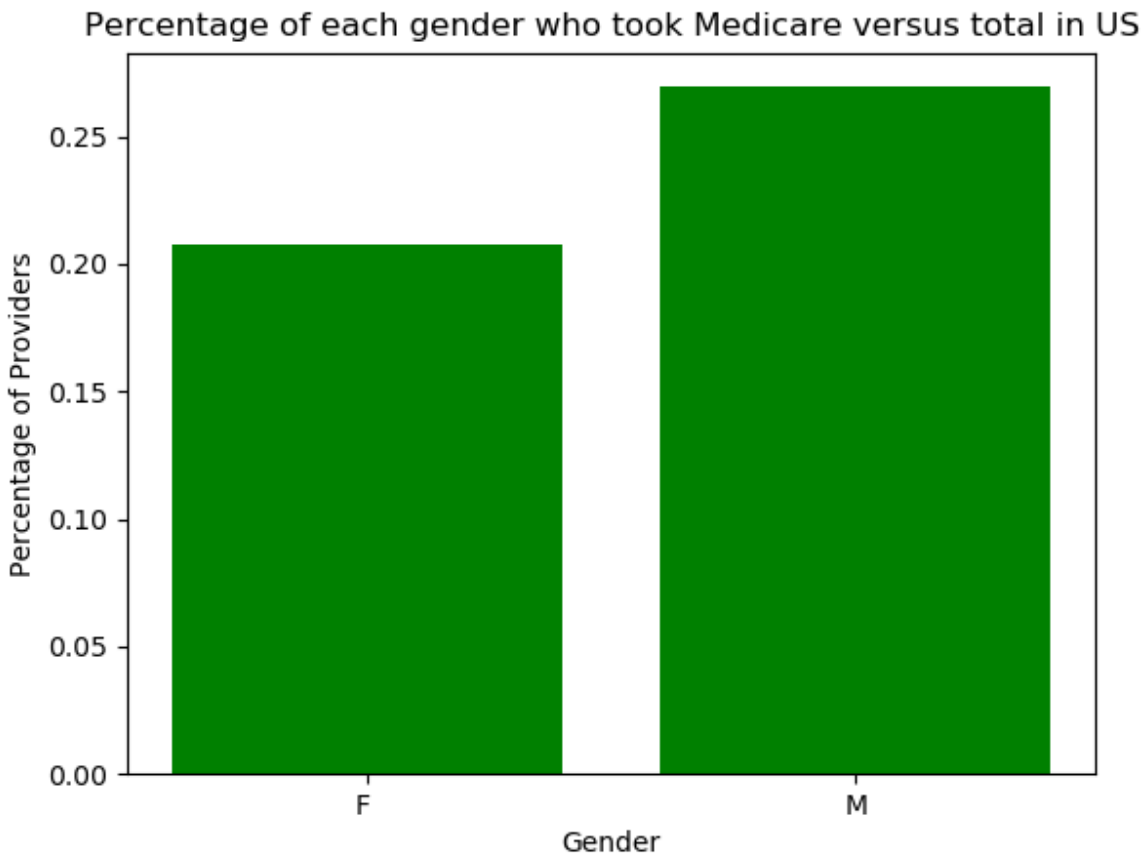
```
'''
Visualizing the proportion of gender that take Medicare
'''
```

```
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv('gender_take_medicare.csv')

x = list(df.iloc[:, 0])
y = list(df.iloc[:, 1])

colors = ["#1f77b4", "#ff7f0e"]
explode = (0, 0)
plt.pie(y, labels=x, explode=explode, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)
plt.title("Proportion of genders of individuals who took Medicare in 2019")
plt.show()
```



Barplot: Percentage of each gender who took Medicare versus total in the US

- According to Becker Hospital Review, there are 366,759 (36%) practicing female physicians and 652,017 (64%) practicing male physicians, for a total of 1,018,776 practicing physicians in the US total



4. **Proportion of individuals vs. corporations who take Medicare (corporation_vs_individual.py)**

```
'''
Analyzing number of corporations vs. individuals who take Medicare
'''

from mrjob.job import MRJob
import re

class MRCorpAndIndividual(MRJob):

   def mapper(self, _, line):
       COMMA_MATCHER = re.compile(r",(?=(?:[^\"']*[\"'][^\"']*[\"'])*[^\"']*$)")
       line_cols = COMMA_MATCHER.split(line)
       individual = line_cols[5]
       if individual == "F" or individual == "M":
           yield ("individual", 1)
       else:
           yield ("corporation", 1)

   def combiner(self, type, counts):
       yield type, sum(counts)

   def reducer(self, type, counts):
       yield type, sum(counts)

if __name__ == '__main__':
   MRCorpAndIndividual.run()
```

Output:
"corporation" 1129
"individual" 251714

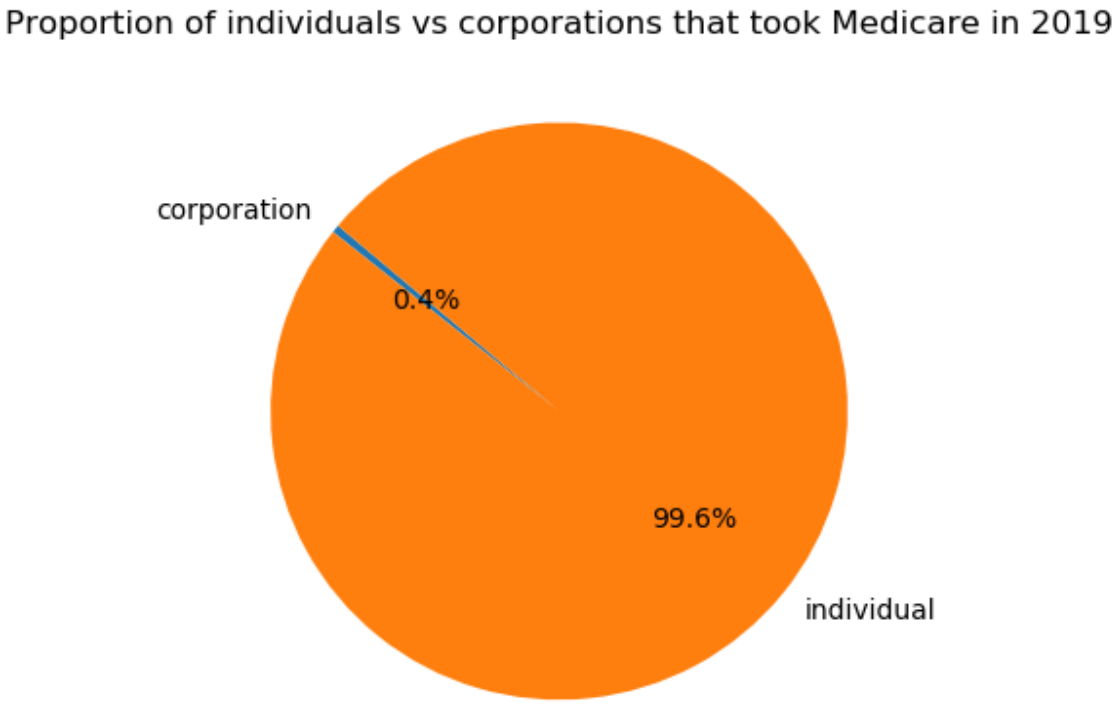
Processed Output (corporation_vs_individual.csv):

,
corporation, 1129
individual, 251714

Pie chart for corporation_vs_individual.py (pichart_corporation_vs_individual.py)

```
'''
Visualizing the proportion of gender that take Medicare
'''
```

```
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv('corporation_vs_individual.csv')

x = list(df.iloc[:, 0])
y = list(df.iloc[:, 1])

colors = ["#1f77b4", "#ff7f0e"]
explode = (0, 0)
plt.pie(y, labels=x, explode=explode, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)
plt.title("Proportion of individuals vs corporations that took Medicare in 2019")

plt.show()
```



5. **Number of unique provider types in state that take Medicare (state_take_medicare.py)**

```
'''
Analyzing number of unique types of providers in states who take Medicare
'''

from mrjob.job import MRJob
import re

class MRStateProviders(MRJob):

    def mapper(self, _, line):
        COMMA_MATCHER = re.compile(r",(?=(?:[^\"']*[\"'][^\"']*[\"'])*[^\"']*$)")
        line_cols = COMMA_MATCHER.split(line)
        if len(line_cols[10]) == 2 and line_cols[10] != "AA" and line_cols[10] != "ZZ"
and line_cols[10] != "XX" and line_cols[10] != "US" and not line_cols[10].isnumeric():
            state = line_cols[10]
            provider_type = line_cols[16]
        else:
            state = ""
            provider_type = ""
        if state:
            yield (state, provider_type)

    def combiner(self, state, provider_type):
        types = set(provider_type)
        for unq_type in types:
            yield (state, unq_type)

    def reducer(self, state, provider_type):
        types = set(provider_type)
        yield state, len(types)

if __name__ == '__main__':
    MRStateProviders.run()
```

Output:
"AE" 6
"AK" 42
"AL" 64
"AP" 5

"AR" 59
"AZ" 64
"CA" 69
"CO" 62
"CT" 59
"DC" 50
"DE" 52
"FL" 69
"GA" 63
"GU" 17
"HI" 53
"IA" 56
"ID" 55
"IL" 70
"IN" 62
"KS" 57
"KY" 58
"LA" 58
"MA" 64
"MD" 62
"ME" 54
"MI" 61
"MN" 63
"MO" 61
"MP" 5
"MS" 51
"MT" 54
"NC" 68
"ND" 51
"NE" 56
"NH" 57
"NJ" 64
"NM" 55
"NV" 53
"NY" 73
"OH" 65
"OK" 55
"OR" 61
"PA" 70
"PR" 52
"RI" 52
"SC" 60
"SD" 51
"TN" 62

"TX" 69
"UT" 58
"VA" 64
"VI" 17
"VT" 48
"WA" 65
"WI" 64
"WV" 51
"WY" 39

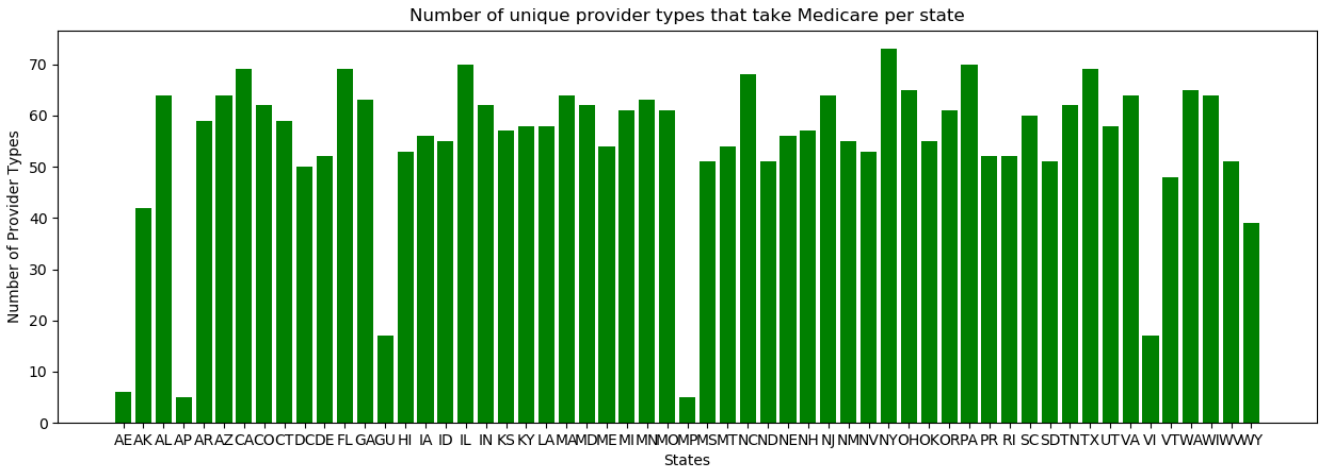
Processed Output (state_take_medicare.csv):

,
AE, 6
AK, 42
AL, 64
AP, 5
AR, 59
AZ, 64
CA, 69
CO, 62
CT, 59
DC, 50
DE, 52
FL, 69
GA, 63
GU, 17
HI, 53
IA, 56
ID, 55
IL, 70
IN, 62
KS, 57
KY, 58
LA, 58
MA, 64
MD, 62
ME, 54
MI, 61
MN, 63
MO, 61
MP, 5
MS, 51
MT, 54
NC, 68

ND, 51
NE, 56
NH, 57
NJ, 64
NM, 55
NV, 53
NY, 73
OH, 65
OK, 55
OR, 61
PA, 70
PR, 52
RI, 52
SC, 60
SD, 51
TN, 62
TX, 69
UT, 58
VA, 64
VI, 17
VT, 48
WA, 65
WI, 64
WV, 51
WY, 39

Barchart of unique provider types in state that take Medicare (barchart_state_take_medicare.py)

```
import matplotlib.pyplot as plt
import pandas as pd

data = pd.read_csv('state_take_medicare.csv')
 df = pd.DataFrame(data)
 X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])
plt.bar(X, Y, color='g')
plt.title("Number of unique provider types that take Medicare per state")
plt.xlabel("States")
plt.ylabel("Number of Provider Types")
 plt.show()
```



TOP 10 STATES (top_ten_states.py)**:**

```
from mrjob.job import MRJob
import re
import heapq

# Question 5
class MRStateProviders_heap(MRJob):
    '''
    Finds top 10 states for unique types of providers (out of states who
take Medicare)
    '''
    def mapper(self, _, line):
        COMMA_MATCHER =
re.compile(r",(?=(?:[^\"']*[\"'][^\"']*[\"'])*[^\"']*$)")
        line_cols = COMMA_MATCHER.split(line)
        if len(line_cols[10]) == 2 and line_cols[10] != "AA" and
line_cols[10] != "ZZ" and \
        line_cols[10] != "XX" and line_cols[10] != "US" and not
line_cols[10].isnumeric():
            state = line_cols[10]
            provider_type = line_cols[16]
        else:
            state = ""
            provider_type = ""
        if state:
            yield (state, provider_type)
    def combiner(self, state, provider_type):
```

```
        types = set(provider_type)
        for unq_type in types:
            yield (state, unq_type)
    def reducer_init(self):
        self.h = []
    def reducer(self, state, provider_type):
        types = set(provider_type)
        num_types = len(types)
        if len(self.h) < 10:
            self.h.append((num_types, state))
            if len(self.h) == 10:
                heapq.heapify(self.h)
        else:
            min_count, _ = self.h[0]
            if num_types > min_count:
                heapq.heapreplace(self.h, (num_types, state))
    def reducer_final(self):
        self.h.sort(reverse=True)
        for num_types, state in self.h:
            yield state, num_types
if __name__ == '__main__':
 MRStateProviders_heap.run()
```

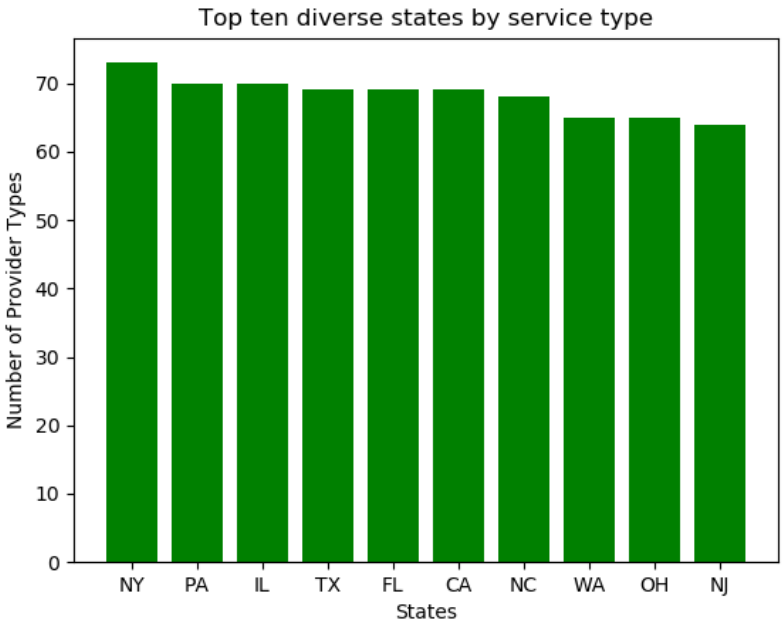
Processed Output (top_ten_states.csv):

,
NY, 73
PA, 70
IL, 70
TX, 69
FL, 69
CA, 69
NC, 68
WA, 65
OH, 65
NJ, 64

Barchart (barchart_top_ten_states.py):

```
import matplotlib.pyplot as plt
import pandas as pd
data = pd.read_csv('top_ten_states.csv')
```

```
df = pd.DataFrame(data)
X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])
plt.bar(X, Y, color='g')
plt.title("Top ten diverse states by service type")
plt.xlabel("States")
plt.ylabel("Number of Provider Types")
plt.show()
```



6. **Number of providers in a state (state_any_take_medicare.py)**

```
'''
Analyzing number of providers in states who take Medicare
'''

from mrjob.job import MRJob
import re

class MRStateAnyTakeMedicare(MRJob):

   def mapper(self, _, line):
       COMMA_MATCHER = re.compile(r",(?=(?:[^\"']*[\"'][^\"']*[\"'])*[^\"']*$)")
       line_cols = COMMA_MATCHER.split(line)
       if len(line_cols[10]) == 2 and line_cols[10] != "AA" and line_cols[10] != "ZZ"
and line_cols[10] != "XX" and line_cols[10] != "US" and not line_cols[10].isnumeric():
           state = line_cols[10]
       else:
           state = ''
       if state:
           yield (state, 1)

   def combiner(self, state, counts):
       yield state, sum(counts)

   def reducer(self, state, counts):
       yield state, sum(counts)

if __name__ == '__main__':
   MRStateAnyTakeMedicare.run()
```

Output:

"AE" 6
"AK" 554
"AL" 3330
"AP" 6
"AR" 1994
"AZ" 4605
"CA" 20397
"CO" 4248
"CT" 3801
"DC" 1116
"DE" 827
"FL" 16703
"GA" 6736

"GU" 51
"HI" 764
"IA" 2033
"ID" 1111
"IL" 9307
"IN" 5542
"KS" 2067
"KY" 3284
"LA" 3637
"MA" 8365
"MD" 4603
"ME" 1347
"MI" 7884
"MN" 5618
"MO" 5291
"MP" 6
"MS" 1602
"MT" 933
"NC" 8870
"ND" 911
"NE" 1349
"NH" 1531
"NJ" 7422
"NM" 1537
"NV" 1824
"NY" 19370
"OH" 9756
"OK" 2312
"OR" 3624
"PA" 12984
"PR" 1494
"RI" 1268
"SC" 4065
"SD" 819
"TN" 5560
"TX" 15963
"UT" 1971
"VA" 6390
"VI" 43
"VT" 610
"WA" 6388
"WI" 6045
"WV" 1488
"WY" 327

Processed Output (state_any_take_medicare.csv):

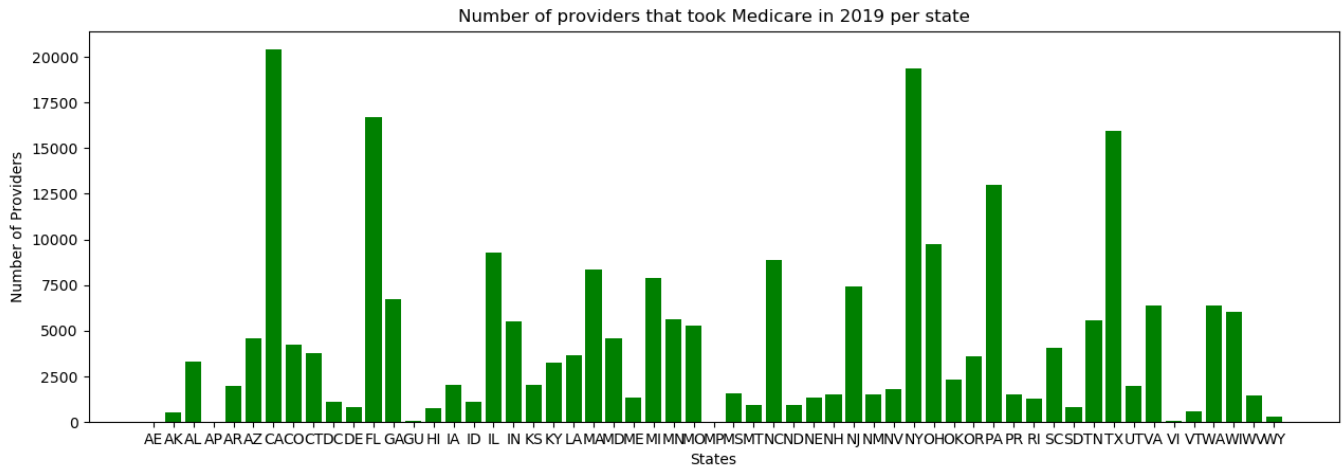
,
AE, 6
AK, 554

AL, 3330
AP, 6
AR, 1994
AZ, 4605
CA, 20397
CO, 4248
CT, 3801
DC, 1116
DE, 827
FL, 16703
GA, 6736
GU, 51
HI, 764
IA, 2033
ID, 1111
IL, 9307
IN, 5542
KS, 2067
KY, 3284
LA, 3637
MA, 8365
MD, 4603
ME, 1347
MI, 7884
MN, 5618
MO, 5291
MP, 6
MS, 1602
MT, 933
NC, 8870
ND, 911
NE, 1349
NH, 1531
NJ, 7422
NM, 1537
NV, 1824
NY, 19370
OH, 9756
OK, 2312
OR, 3624
PA, 12984
PR, 1494
RI, 1268
SC, 4065
SD, 819
TN, 5560
TX, 15963
UT, 1971
VA, 6390

VI, 43
VT, 610
WA, 6388
WI, 6045
WV, 1488
WY, 327

Barchart for state_any_take_medicare.py (barchart_state_any_take_medicare.py)

```
import matplotlib.pyplot as plt
import pandas as pd

data = pd.read_csv('state_any_take_medicare.csv')
 df = pd.DataFrame(data)
 X = list(df.iloc[:, 0])
Y = list(df.iloc[:, 1])
 plt.bar(X, Y, color='g')
plt.title("Number of providers that took Medicare in 2019 per state")
plt.xlabel("States")
plt.ylabel("Number of Providers")
 plt.show()
```



Another question that we wanted to answer: We were looking to create a referral network in each state for internal medicine with cardiology doctors to map out coverage.

- We were not able to get this code to fully run, but after many edits, this should work
- We made a google storage bucket (as explained up top) and got the link to this file

File name: mapper_pairs.py

```
from mrjob.job import MRJob
from mrjob.step import MRStep

'''
Trying to count the number of internal medicine + cardiology pairs
per state
'''

class build_top_pairs(MRJob):
   def mapper_init(self):
       # We made a big-data-project_storage-bucket and got a
       # gs link to access the data set through Google
       gcs_file = gcs.open('gs://big-data-project_storage-bucket/med_data.csv')
       self.data = gcs_file.read()
       gcs_file.close()


   def mapper(self, _, line):
       row1 = line.strip().split(",")
       state1 = row1[10]

       # loop through dataset for every iteration of mapper
       for row in self.data:
           row2 = row.strip().split(",")
           state2 = row2[10]
           if ("Internal Medicine" in row1) and \
               ("Cardiology" in row2) and state1 == state2:
               yield (state1, 1)


   def combiner(self, state, counts):
       yield rows, state, sum(counts)


   def reducer(self, state, counts):
       yield state, sum(counts)

if __name__ == "__main__":
```

```
  build_top_pairs.run()
```

https://stackoverflow.com/questions/48279061/gcs-read-a-text-file-from-google-cloud-storage-directly-into-python

Another question that we aimed to solve was also finding the average age of mothers and fathers at the time of a baby's birth so as to gage our target coverage audience.
Here was the dataset we used: (avg_mom_age.py, avg_dad_age.py)
https://console.cloud.google.com/bigquery?p=bigquery-public-data&d=samples&t=natality&page=table&_ga=2.86294480.999355211.1653932284-1312598613.1652540771&project=lab2-351619&ws=!1m5!1m4!4m3!1sbigquery-public-data!2ssamples!3snatality

- The code is shown below

```
'''
Average age of mothers who gave birth
'''
from mrjob.job import MRJob
import re

class MRAvgMotherAge(MRJob):

   def mapper(self, _, line):
       COMMA_MATCHER = re.compile(r",(?=(?:[^\"']*[\"'][^\"']*[\"'])*[^\"']*$)")
       line_cols = COMMA_MATCHER.split(line)
       mother_age = line_cols[14]
       yield None, mother_age

   def reducer(self, _, mother_ages):
       sum = 0
       count = 0
       for age in mother_ages:
         sum += age
         count += 1
       yield None, sum/count

if __name__ == '__main__':
 MRAvgMotherAge.run()
```

```
'''
Average age of fathers at time of birth
'''
```

```
from mrjob.job import MRJob
import re

class MRAvgFatherAge(MRJob):

   def mapper(self, _, line):
       COMMA_MATCHER = re.compile(r",(?=(?:[^\"']*[\"'][^\"']*[\"'])*[^\"']*$)")
       line_cols = COMMA_MATCHER.split(line)
       father_age = line_cols[29]
       yield None, father_age

   def reducer(self, _, father_ages):
       sum = 0
       count = 0
       for age in father_ages:
         sum += age
         count += 1
       yield None, sum/count

if __name__ == '__main__':
 MRAvgFatherAge.run()
```

Extraneous GoogleCloud terminal code from lab 3:
scp -i ~/.ssh/google-cloud-cs123 med_data.csv bkatsnelson@34.68.254.233:~/

CODE FOR ZIP CODES:

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import heapq
import csv


class build_top_pairs(MRJob):
   def mapper_init():
       with open('med_data.csv') as dataset:
           self.data = dataset.read_lines()

   def mapper(self, _, line):
       row1 = line.strip().split(",")
```

```
       state1 = row1[10]
          # loop through dataset for every iteration of mapper
           for row in self.data:
               row2 = row.strip().split(",")
               state2 = row2[10]
               if ("Internal Medicine" in row1) and ("Cardiology" in row2)
and state1 == state2:
                   yield (row1 + row2), (state1, 1)

   def combiner(self, rows, state, counts):
       yield rows, state, sum(counts)

   def reducer_init(self):
       self.h = []
   def reducer(self, rows, state, counts):
      row1 = rows[0:29]
      row2 = rows[29:]
      yield rows, state, sum(counts)
          if len(self.h) < 3: # find closest 3 for every person in the
list
              self.h.append((dist, (row1, row2)))
              if len(self.h) == 3:
                  heapq.heapify(self.h)
          else:
              min_count, _ = self.h[0]
              if dist > min_count:
                  heapq.heapreplace(self.h, (dist, (row1, row2)))
   def reducer_final(self):
      self.h.sort(reverse = True):
      for dist, (row1, row2) in self.h:
          state = row1[10]
          first_name1 = row1[1]
          last_name1 = row1[2]
          profession1 = row1[16]
          zip_code1 = row1[12]
          first_name2 = row2[1]
          last_name2 = row2[2]
          profession2 = row2[16]
          zip_code2 = row2[12]
          distance = dist*(-1)
```

```
          yield (("SEPARATE_FIELDS").join([state, first_name1, last_name1,
profession1, zip_code1, \
          first_name2, last_name2, profession2, zip_code2, distance])),
None
if __name__ == "__main__":
  build_top_pairs.run()
## MAP_TO_CSV IMPLEMENTATION:
import csv
## pre_csv = task_p.py < filename.csv > output.csv
fieldnames = ["state", "first_name1", "last_name1", "profession1",
"zip_code1", \
          "first_name2", "last_name2", "profession2", "zip_code2",
"distance"]
with open("final_output.csv", "w") as f:
  writer = csv.DictWriter(f, fieldnames=fieldnames)
  writer.writeheader()
  # filename stands for csv file generated from mapper
  with open(filename) as f_old:
      for line in f_old:
          row = line.split("\t")
          row_keep = row[0]
          fields = row_keep.split("SEPARATE_FIELDS")
          row_to_add = {"state": fields[0],
              "first_name1": fields[1],
              "last_name1": fields[2],
              "profession1": fields[3],
              "zip_code1": fields[4],
              "first_name2": fields[5],
              "last_name2": fields[6],
              "profession2": fields[7],
              "zip_code2": fields[8],
              "distance": fields[9]}
          writer.writerow(row_to_add)

'''
search = SearchEngine(simple_zipcode=False)
data = []
with open(filename) as f:
  header = f.readline() # Skip the header row
  for row in f:
```

```
      data.append(row)
# Don't want to iterate twice through the same person
used_ids = set()
class build_top_pairs(MRJob):
  def mapper(self, _, line):
      h = []
      row1 = line.strip().split(",")
      if row1 not in used_ids:
          zip1 = search.by_zipcode(row1[12])
          lat1 = zip1.lat
          long1 = zip1.lng
          used_ids.add(row1[0])
          used_ids2 = set()
          # loop through dataset for every iteration of mapper
          for row in data:
              row2 = row.strip().split(",")
              if (row2[0] not in used_ids2) and (row1[0] != row2[0]):
                  used_ids2.add(row2[0])
                  zip2 = search.by_zipcode(row2[12])
                  lat2 = zip2.lat
                  long2 = zip2.lng
                  # Multiply distance by -1 since want to find 3 smallest
                  dist = mpu.haversine_distance((lat1,long1),(lat2,long2))
* (-1)
                  yield (row1 + row2), dist
  def reducer_init(self):
      self.h = []
  def reducer(self, rows, dist):
      row1 = rows[0:29]
      row2 = rows[29:]
      if (row1[16] == "Internal Medicine" or row2[16] == "Internal
Medicine") \
      and (row1[16] == "Cardiologist" or row2[16] == "Cardiologist") and \
      (row1[10] == row2[10]): # same state
          if len(self.h) < 3: # find closest 3 for every person in the
list
              self.h.append((dist, (row1, row2)))
              if len(self.h) == 3:
                  heapq.heapify(self.h)
          else:
```

```
            min_count, _ = self.h[0]
            if dist > min_count:
                heapq.heapreplace(self.h, (dist, (row1, row2)))
  def reducer_final(self):
      self.h.sort(reverse = True):
      for dist, (row1, row2) in self.h:
          state = row1[10]
          first_name1 = row1[1]
          last_name1 = row1[2]
          profession1 = row1[16]
          zip_code1 = row1[12]
          first_name2 = row2[1]
          last_name2 = row2[2]
          profession2 = row2[16]
          zip_code2 = row2[12]
          distance = dist*(-1)
          yield (("SEPARATE_FIELDS").join([state, first_name1, last_name1,
profession1, zip_code1, \
          first_name2, last_name2, profession2, zip_code2, distance])),
None
if __name__ == "__main__":
  build_top_pairs.run()
## MAP_TO_CSV IMPLEMENTATION:
import csv
## pre_csv = task_p.py < filename.csv > output.csv
fieldnames = ["state", "first_name1", "last_name1", "profession1",
"zip_code1", \
          "first_name2", "last_name2", "profession2", "zip_code2",
"distance"]
with open("final_output.csv", "w") as f:
  writer = csv.DictWriter(f, fieldnames=fieldnames)
  writer.writeheader()
  # filename stands for csv file generated from mapper
  with open(filename) as f_old:
      for line in f_old:
          row = line.split("\t")
          row_keep = row[0]
          fields = row_keep.split("SEPARATE_FIELDS")
          row_to_add = {"state": fields[0],
              "first_name1": fields[1],
```

```
                "last_name1": fields[2],
                "profession1": fields[3],
                "zip_code1": fields[4],
                "first_name2": fields[5],
                "last_name2": fields[6],
                "profession2": fields[7],
                "zip_code2": fields[8],
                "distance": fields[9]}
            writer.writerow(row_to_add)
'''
```

EXTRANEOUS CODE:

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import heapq
import csv


class build_top_pairs(MRJob):

    def mapper_init(self):
        gcs_file = gcs.open(filename)
        contents = gcs_file.read()
        gcs_file.close()

    def mapper(self, _, line):

        gcs_file = gcs.open(filename)
        contents = gcs_file.read()

        row1 = line.strip().split(",")
        state1 = row1[10]
        # loop through dataset for every iteration of mapper
        for row in self.data:
            row2 = row.strip().split(",")
            state2 = row2[10]
            if ("Internal Medicine" in row1) and ("Cardiology" in row2) and state1 == state2:
                yield state1, 1
```

```
        gcs_file.close()

    def combiner(self, state, counts):
        yield state, sum(counts)

    def reducer(self, state, counts):
        yield state, sum(counts)


if __name__ == "__main__":
   build_top_pairs.run()
```

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import heapq
import csv


class build_top_pairs(MRJob):
    def mapper_init(self):
        with open('med_data.csv') as dataset:
            self.data = dataset.read_lines()

    def mapper(self, _, line):
        row1 = line.strip().split(",")
        state1 = row1[10]
        # loop through dataset for every iteration of mapper
        for row in self.data:
            row2 = row.strip().split(",")
            state2 = row2[10]
            if ("Internal Medicine" in row1) and ("Cardiology" in row2) and state1 == state2:
                yield state1, 1

    def combiner(self, state, counts):
        yield state, sum(counts)

    def reducer(self, state, counts):
        yield state, sum(counts)
```

```
if __name__ == "__main__":
  build_top_pairs.run()
```

```
reader=csv.reader(open('med_data.csv', 'r'), delimiter=',')
writer=csv.writer(open('med_data_paired.csv', 'w'), delimiter=',')
for row1 in reader:
   row1_string = ",".join(row1)
   COMMA_MATCHER1 =
re.compile(r",(?=(?:[^\"']*[\"'][^\"']*[\"'])*[^\"']*$)")
   line1 = COMMA_MATCHER1.split(row1_string)
   state1 = line1[10]
   for row2 in reader:
       row2_string = ",".join(row2)
       COMMA_MATCHER2 =
re.compile(r",(?=(?:[^\"']*[\"'][^\"']*[\"'])*[^\"']*$)")
       line2 = COMMA_MATCHER2.split(row2_string)
       if len(line2) > 11:
           state2 = line2[10]
       else:
           state2 = "None"

       if ("Internal Medicine" in row1) and ("Cardiology" in row2) and
state1 == state2:
           writer.writerow(row2)
```

```
import logging
import os
import cloudstorage as gcs
import webapp2

from google.appengine.api import app_identity

def get(self):
   bucket_name = "big-data-project_storage-bucket"

   self.response.headers['Content-Type'] = 'text/plain'
   self.response.write('Demo GCS Application running from Version: '
                       + os.environ['CURRENT_VERSION_ID'] + '\n')
   self.response.write('Using bucket name: ' + bucket_name + '\n\n')


def create_file(self, filename):
   """Create a file.

   The retry_params specified in the open call will override the default
   retry params for this particular file handle.

   Args:
     filename: filename.
   """
   self.response.write('Creating file %s\n' % filename)

   write_retry_params = gcs.RetryParams(backoff_factor=1.1)
   gcs_file = gcs.open(filename,
                       'w',
                       content_type='text/plain',
                       options={'x-goog-meta-foo': 'foo',
                                'x-goog-meta-bar': 'bar'},
                       retry_params=write_retry_params)
   gcs_file.write('abcde\n')
   gcs_file.write('f'*1024*4 + '\n')
   gcs_file.close()
   self.tmp_filenames_to_clean_up.append(filename)
```

```
def read_file(self, filename):
  self.response.write('Reading the full file contents:\n')

  gcs_file = gcs.open(filename)
  contents = gcs_file.read()
  gcs_file.close()
  self.response.write(contents)
```