Question 2

a) - c)

In [1]:
```
in_string = "this deluge of data calls for automated methods of data a
list_of_strings = in_string.split()
print(list_of_strings[::3])  #fix me to answer the question
print(list_of_strings[-16])  #fix me to answer the question
```

```
['this', 'data', 'automated', 'data', 'is', 'learning', 'particular',
'machine', 'a', 'methods', 'automatically', 'in', 'then', 'uncovered'
, 'predict', 'or', 'other', 'decision', 'uncertainty']
uncovered
```

d)

In [3]:
```
tuple_of_strings = tuple(list_of_strings)
print(tuple_of_strings)
```

```
('this', 'deluge', 'of', 'data', 'calls', 'for', 'automated', 'method
s', 'of', 'data', 'analysis', 'which', 'is', 'what', 'machine', 'lear
ning', 'provides', 'in', 'particular', 'we', 'define', 'machine', 'le
arning', 'as', 'a', 'set', 'of', 'methods', 'that', 'can', 'automatic
ally', 'detect', 'patterns', 'in', 'data', 'and', 'then', 'use', 'the
', 'uncovered', 'patterns', 'to', 'predict', 'future', 'data', 'or',
'to', 'perform', 'other', 'kinds', 'of', 'decision', 'making', 'under
', 'uncertainty')
```

When to use a tuple:

To make the collection immutable;

When DRAM space and/or iteration speed is a predominant concern

e)

In [6]:
```
set_of_strings = set(list_of_strings)
print(set_of_strings)
print(len(set_of_strings))
```

```
{'patterns', 'future', 'learning', 'use', 'calls', 'particular', 'or'
, 'making', 'the', 'deluge', 'to', 'a', 'provides', 'then', 'other',
'set', 'automatically', 'as', 'automated', 'detect', 'and', 'define',
'which', 'analysis', 'is', 'decision', 'we', 'methods', 'can', 'uncov
ered', 'in', 'machine', 'this', 'under', 'perform', 'that', 'for', 'w
hat', 'predict', 'of', 'data', 'kinds', 'uncertainty'}
43
```

Sets do not allow duplicates.
Sets are unordered.

f)

In [7]:
```
d = {}  #create an empty dictionary
for i in list_of_strings:
    if i not in d:
        d[i]=0
    d[i] += 1

print(d)
```

```
{'this': 1, 'deluge': 1, 'of': 4, 'data': 4, 'calls': 1, 'for': 1, 'a
utomated': 1, 'methods': 2, 'analysis': 1, 'which': 1, 'is': 1, 'what
': 1, 'machine': 2, 'learning': 2, 'provides': 1, 'in': 2, 'particula
r': 1, 'we': 1, 'define': 1, 'as': 1, 'a': 1, 'set': 1, 'that': 1, 'c
an': 1, 'automatically': 1, 'detect': 1, 'patterns': 2, 'and': 1, 'th
en': 1, 'use': 1, 'the': 1, 'uncovered': 1, 'to': 2, 'predict': 1, 'f
uture': 1, 'or': 1, 'perform': 1, 'other': 1, 'kinds': 1, 'decision':
1, 'making': 1, 'under': 1, 'uncertainty': 1}
```

g)

Entries "of" and "data" appear most frequently at 4 times.

h)

In [10]:
```
list_of_tuples = [(k,v) for k,v in d.items()]  #this line uses a list
print(list_of_tuples)
```

```
[('this', 1), ('deluge', 1), ('of', 4), ('data', 4), ('calls', 1), ('
for', 1), ('automated', 1), ('methods', 2), ('analysis', 1), ('which'
, 1), ('is', 1), ('what', 1), ('machine', 2), ('learning', 2), ('prov
ides', 1), ('in', 2), ('particular', 1), ('we', 1), ('define', 1), ('
as', 1), ('a', 1), ('set', 1), ('that', 1), ('can', 1), ('automatical
ly', 1), ('detect', 1), ('patterns', 2), ('and', 1), ('then', 1), ('u
se', 1), ('the', 1), ('uncovered', 1), ('to', 2), ('predict', 1), ('f
uture', 1), ('or', 1), ('perform', 1), ('other', 1), ('kinds', 1), ('
decision', 1), ('making', 1), ('under', 1), ('uncertainty', 1)]
```

Question 3

a) - c)

In [11]:
```
import numpy as np
import matplotlib.pyplot as plt
A = [[1,1,3],[4,4,4],[5,6,9]]
A = np.array(A)
A_inv = np.linalg.inv(A)

print(A_inv)

### Write code below to answer question
```

```
[[ 1.5    1.125 -1.   ]
 [-2.    -0.75   1.   ]
 [ 0.5   -0.125  0.   ]]
```

d)

In [15]:
```
arr = np.random.rand(10)
print(arr)
```

```
[0.70306528 0.49234068 0.7222745  0.39293689 0.33672632 0.07859635
 0.30616581 0.4001807  0.77434804 0.0330423 ]
```

3)

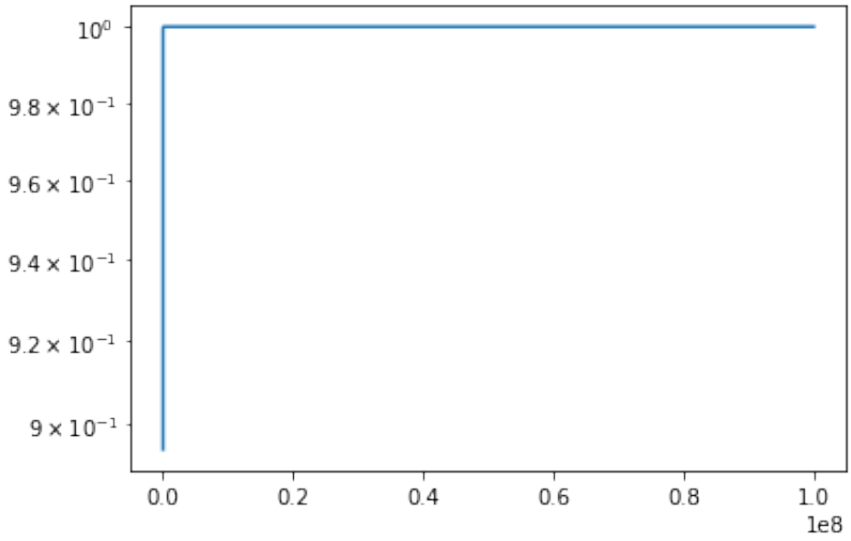
In [42]:
```
mins = np.zeros(8)
for e in range(1, 9):
    arr = np.random.rand(10**e)
    mins[e-1] = min(arr)
    print("n = 10^", e, ", min = ", mins[e-1])
```

```
n = 10^ 1 , min =  0.06364182628682502
n = 10^ 2 , min =  0.003973275989821112
n = 10^ 3 , min =  0.0007439726677122005
n = 10^ 4 , min =  0.00026061296743173923
n = 10^ 5 , min =  1.0708722494090495e-05
n = 10^ 6 , min =  3.564983015014178e-06
n = 10^ 7 , min =  1.7324049650380147e-07
n = 10^ 8 , min =  2.073066718288885e-09
```

In [47]:
```
maxs = np.zeros(8)
for e in range(1, 9):
    arr = np.random.rand(10**e)
    maxs[e-1] = max(arr)
    print("n = 10^", e, ", min = ", maxs[e-1])
```

```
n = 10^ 1 , min =  0.8940236100710377
n = 10^ 2 , min =  0.9939057212986895
n = 10^ 3 , min =  0.9992434215034283
n = 10^ 4 , min =  0.9998750551799596
n = 10^ 5 , min =  0.9999923134167332
n = 10^ 6 , min =  0.9999994155745234
n = 10^ 7 , min =  0.9999999846132266
n = 10^ 8 , min =  0.9999999999476651
```

In [48]:
```
x_vals = np.geomspace(10, 10**8, num=8)
plt.plot(x_vals, maxs)
plt.yscale("log")
plt.show()
```



Result aligns with my expectation. As you increase the number of iterations, there is a higher chance of getting a even greater number bounded by (0,1).

In [ ]:
```
```