1) Easy

Import json file and extract features

In [31]:
```
import json
import pandas as pd
```

In [167]:
```
f = open('ny_times_movies.json','r')
data=json.load(f)
```

In [164]:
```
df=pd.DataFrame(data['results'])
df.head(2)
```

Out[164]:

	byline	critics_pick	date_updated	display_title	headline	link	mpaa_rating	multimedia	opening_date	publi
0	A.O. SCOTT	1	2018-10-17 02:44:23	Can You Ever Forgive Me	Review: Melissa McCarthy Is Criminally Good in...	{'type': 'article', 'url': 'http://www.nytimes...	R	{'type': 'mediumThreeByTwo210', 'src': 'https:...	2018-10-19	
1	BEN KENIGSBERG	1	2018-10-16 11:04:03	Charm City	Review: ‘Charm City’ Vividly Captures the Stre...	{'type': 'article', 'url': 'http://www.nytimes...		{'type': 'mediumThreeByTwo210', 'src': 'https:...	2018-04-22	

In [45]:
```
df['url']=df.link.map(lambda x: x['url'])
df['type']=df.link.map(lambda x: x['type'])
df=df.drop(['link','multimedia'], axis = 1)
```

In [47]:
```
df.head(2)
```

Out[47]:

	byline	critics_pick	date_updated	display_title	headline	mpaa_rating	opening_date	publi	url	type
0	A.O. SCOTT	1	2018-10-17 02:44:23	Can You Ever Forgive Me	Review: Melissa McCarthy Is Criminally Good in...	R	2018-10-19		http://www.nytimes...	article
1	BEN KENIGSBERG	1	2018-10-16 11:04:03	Charm City	Review: ‘Charm City’ Vividly Captures the Stre...		2018-04-22		http://www.nytimes...	article

2) Easy

Given an array and an integer A, find the maximum for each contiguous subarray of size A.

Input: array = [1, 2, 3, 1, 4, 5, 2, 3, 6], A = 3

Output: 3 3 4 5 5 5 6

In [60]:
```
def function_(array_:list, size:int)->list:
    output_=[]
    for i in range(len(array_)-size+1):
        output_.append(max(array_[i:i+size]))
    return output_
function_([1, 2, 3, 1, 4, 5, 2, 3, 6],3)
```

Out[60]:
```
[3, 3, 4, 5, 5, 5, 6]
```

3) Hard

Suppose you're given a matrix of 1s and 0s that represents a map of rivers. You can assume that the grid cells in your map are only connected horizontally and vertically (e.g. no diagonal connections). You can assume that 1 represents water (your river) and 0 represents land/your river bank. Each cell has a length of 1 and is square in your map. Given this, write code to determine the perimeter of your river.

Examples:

```
Input: [[1,0]]
Output: 4

Input: [[1,0,1],
        [1,1,1]]
Output: 12
```

In [73]:
```
#Thought, a river must be connected.
# Some river with the same number of grids can have different perimeter
# A naive way to solve this problem is to calculate the peremeter along the way
# For example, starting with the first 1, if 1 adjacent with 3 other 1, then its accumulated parameter will b
e 1.
# Remember that this is a m by n matrix
# Pseudo code: a function that find the number of neighbors of a river grid, and
# a function that caculate the perimeter

import numpy as np

def neighbor(x,y, river_matrix):
    river_matrix=np.array(river_matrix)
    m,n=river_matrix.shape
    count=0
    #There are 4 cases, up, down, right, left, and for each case need to consider if they exist
    #Up
    if (x > 0 and river_matrix[x - 1][y]):
        count+= 1;

    # DOWN
    if (x < m-1 and river_matrix[x + 1][y]): #Not the last row
        count+= 1
    # RIGHT
    if (y < n-1 and river_matrix[x][y + 1]):
        count+= 1;

    # LEFT
    if (y > 0 and river_matrix[x][y - 1]):
        count+= 1;

    return count


A=[[1,0,1], [1,1,1]]
x,y = 1,1
#Expected output: 2
neighbor(x,y, A)
```

Out[73]: 2

In [76]:
```
def river_perimeter(river_matrix):
    #Count number of neighbors of a matrix
    river_matrix=np.array(river_matrix)
    m,n=river_matrix.shape
    perimeter=0

    for row in range(m):
        for col in range(n):
            if river_matrix[row,col]==1:
                num_neighbor=neighbor(row,col,river_matrix)
                perimeter+=4-num_neighbor # Note that each neighbor will take 1 perimeter from the river grid
    return perimeter
```

In [77]:
```
river_perimeter(A)
```

Out[77]: 12

4) Easy

Suppose you're trying to optimize a mailing campaign where physical mail is sent to potential customers. There are two options to send advertising mail: either first class or third class. First-class costs $0.50 per parcel with a delivery rate of 99% and third-class costs \$0.35 per parcel with a delivery rate of 65%. Third-class mail also receives a volume discount, if a batch of parcels is greater than 500, for first 500 are $0.35 and anything additional costs \$0.32 per parcel.

Questions:

Which option is the best for the company? Assume our goal is to reach as many people with a cap of $1000, which option would be the best? Assume our goal is to reach 100 people, what option is best?

In [83]:
```
#Summary
import pandas as pd

pd.DataFrame({'Class':["1st","3rd"], 'Price (n>500)':["0.5*n","0.35*500+(n-500)*0.32"],
              "Price (n<= 500)" : ["0.5*n","0.35*n"], "Deliver rate": [99,65]})
```

Out[83]:

	Class	Price (n>500)	Price (n<= 500)	Deliver rate
0	1st	0.5*n	0.5*n	99
1	3rd	0.35*500+(n-500)*0.32	0.35*n	65

In [94]:
```
# Reach as many people with a cap of $1000

#1st class
print("# people receive 1st-class package:",(1000/0.5)*0.99)

#3rd class: solve 0.35*500+(n-500)*0.32=1000
print("# people receive 3rd-class package:",int(((1000-0.35*500)/0.32+500)*0.65))

#3rd is better
```

```
# people receive 1st-class package: 1980.0
# people receive 3rd-class package: 2000
```

In [100]:
```
# Reach 100 people, what option is best?
#Note that we need to send more than 100 parcels to reach 100 people

#1st class
n=np.ceil((100*1)/0.99)
print("# Number of 1st-class package need to send:",n)
print("Total price:",n*0.5)
print("\n")

#3rd class
n=np.ceil((100*1)/0.65)
print("# Number of 3rd-class package need to send:",n)
print("Total price:",n*0.35)

#1st class is better
```

```
# Number of 1st-class package need to send: 102.0
Total price: 51.0


# Number of 3rd-class package need to send: 154.0
Total price: 53.9
```

5) Moderate

Suppose you're an analyst for an e-commerce store. You're trying to identify the top selling products in Q4 2017 by region, and you have 2 tables that you can query:

Table: all_products

Column Name	Data Type	Description
product_id	integer	id of the product
product_name	string	name of the product
sku	integer	universal stock keeping unit number
distributor_id	integer	id for distributor

Table: orders

Column Name	Data Type	Description
date	string	format is "YYYY-MM-DD"
user_id	integer	id of purchaser
order_id	integer	id of order number
productid	integer	id of product
no_units	integer	number of units sold in the order
price	integer	price per item
shipping_id	integer	id of shipment
region	string	region being shipped to

Using the tables above, write a SQL query to find the top 5 selling products (in terms of total units sold) by region in Q4 of 2017. Include both the distributor id as well as the name of the product in your results.

ANSWER:

Not sure about this one

SQL:

```
SELECT product_id,product_name,SUM(no_units) AS total_unit

FROM all_products JOIN orders ON product_id = productid

WHERE MONTH(date) IN (10,11,12) AND YEAR(date) = 2017

GROUPBY product_id

ORDER BY total_unit DESC

LIMIT 5
```

PANDAS

```
df=all_products.join(lsuffix ='product_id' , rsuffix='productid')
df[(df.date[:4]=='2017') & df.date[5:7].isin(['10','11','12'])]
.groupby(product_id)
.sum()
.sort_values(by='col1', ascending=False)['product_id','product_name','no_units']
.head(5)
```

6) Moderate

Suppose you're given the following information about a population and its wealth distribution:

- Population size: 2500
- Mean income (in USD, thousands): 50
- Standard deviation of income (in USD, thousands): 12.5

The population wealth is initially normally distributed

Additionally, you're told that economic transactions occur randomly between two individuals in the population. In a transaction, two parties come together and there is an exchange of wealth. For the purposes of this question, the transactions can be modeled in the following way:

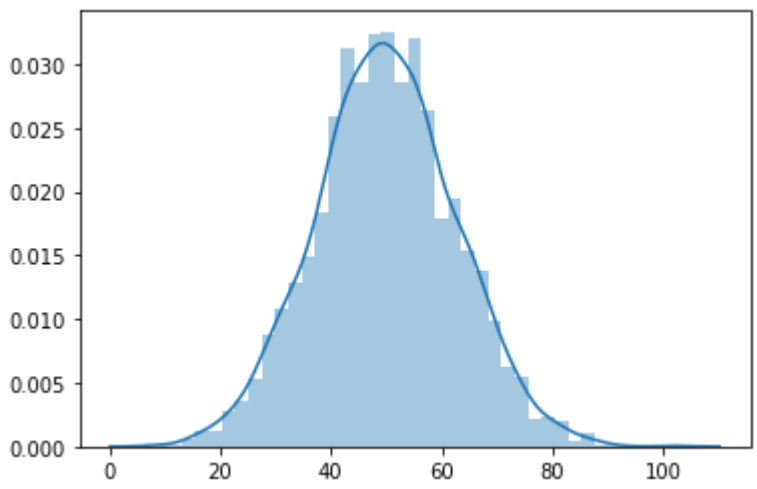
Two individuals come together at random

When they interact, their wealth is put into a pot, and is then split randomly and uniformly between the two parties

Given this information, write a simulation to show how wealth of the population will change over time. You can simulate 50 time steps (or transactions per individual).

In [157]:
```
import seaborn as sns
import matplotlib.pyplot as plt
pop=2500
mu=50
sigma=12.5
pop_pool= np.random.normal(mu, sigma, pop)

sns.distplot(pop_pool)
plt.show()
```

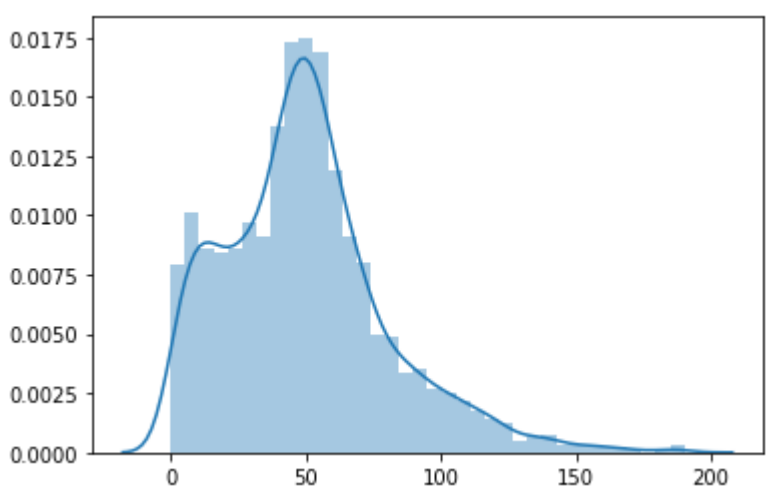


In [163]:
```
num_iter=1000

for i in range(num_iter):
    #Randomize people
    person_1=random.randint(0,pop)
    while True:
        person_2=random.randint(0,pop)
        if(person_2 != person_1):
            break
    fraction=random.random() #Number in [0,1]

    #Redistribution
    total=pop_pool[person_1]+pop_pool[person_2]
    pop_pool[person_1]=total*fraction
    pop_pool[person_2]=total*(1-fraction)

sns.distplot(pop_pool)
plt.show()
```



Interesting: The classes are seriously stratified after 1000 transactions.