Exam GRA4142

Add group member here:

ID 1: 1039909

ID 2: 0617308

ID 3: XXXX

Instructions

The exam consists of four parts. To make it more readable the exam is divided in several small questions. The answers should not be more than a few lines of coding (of course there may be more than one solution method).

Please write your code and execute each line in the notebook. You may add comments to your code if neccessary. (Make sure you don't print out long dataframes. Be concise.)

You can work in groups of up to three students. Please rename this python file as `G_ID1_ID2_ID3.ipynb` where ID1,ID2, and ID3 are the ID numbers of each student in the group. Please, also write in the exam the ID number for each student in the group.

Finally, you cannot talk to other groups during the exam.

In [152]:

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib as mp
import matplotlib.pyplot as plt

%matplotlib inline
```

Question 1 (40 points)

Problem 1.

Write a program that asks the user to enter three numbers (use three separate `input` statements). Create variables called `total` and `average` that hold the sum and average of the three numbers and print out the values of total and average.

In [153]:

```
my_list = []
for i in range(1,4):
    num = int(input('Enter number ' + str(i) +' : '))
    my_list.append(num)

total = np.sum(my_list)
average = np.mean(my_list)

print ('Total = %d and Average = %d' %(total, average))
```

```
Enter number 1 : 1
Enter number 2 : 3
Enter number 3 : 5
Total = 9 and Average = 3
```

Problem 2.

Write a program that generates a list of 50 random integers such that the first integer number is between 1 and 2 (including 2), the second is between 1 and 3, the third is between 1 and 4, . . . , and the last is between 1 and 51.

In [154]:

```
import random
rand_list = []

for i in range(2,52):
    rand_list.append(random.randrange(1, i))

print (rand_list)
```

```
[1, 2, 2, 2, 5, 6, 1, 7, 4, 7, 4, 3, 5, 13, 6, 5, 13, 2, 14, 20, 3, 1, 6,
11, 9, 14, 9, 11, 6, 25, 29, 6, 33, 7, 6, 25, 5, 20, 25, 19, 14, 5, 19, 2
6, 12, 45, 2, 32, 32, 19]
```

Problem 3

Write a function that asks the user to enter a positive integer n, and then computes $\left(1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}\right)-\ln(n)$. Plot the the value of your function for different values of n=10,100,500,1000,10000.

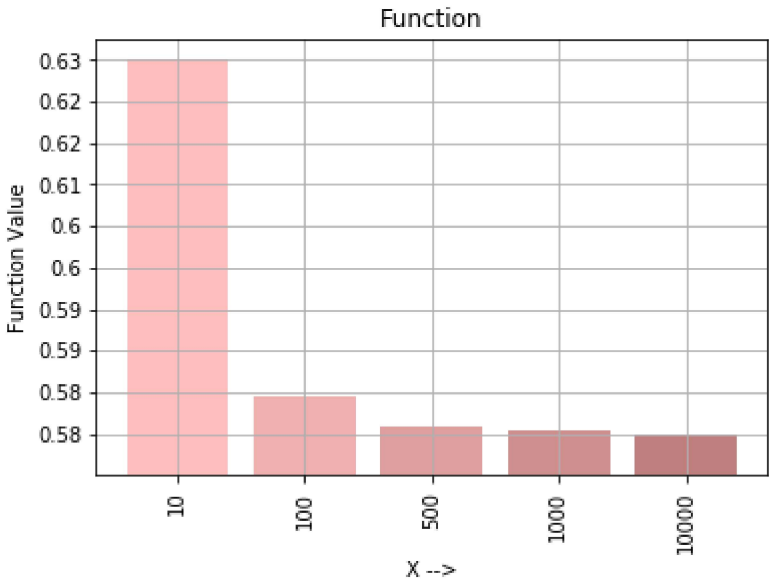
In [156]:

```
import math
inp_num = int(input('Enter a positive number :'))

def y_fun(n):
    sum_y = 0
    for i in range(1,n+1):
        sum_y += 1/i
    return(sum_y - math.log(i))

print (y_fun(inp_num))
```

```
Enter a positive number :8
0.6384156011773068
```

In [157]:

```
# Plot a bar chart
objects = [10,100,500,1000,10000]
y = []
for i in objects:
    y.append(y_fun(i))
y_pos = np.arange(len(objects))
score_y = []

for i in range(len(objects)):
    score_y.append(((y[i] - min(y))/(max(y) - min(y))* max(y))+.07)

y_pos = np.arange(len(objects))
y_lab = np.round(np.linspace(min(y),max(y),10),2)
score = y
plt.yticks(np.linspace(min(score_y), max(score_y),10),y_lab)

like_scores = np.array(range(0,len(objects)))
data_norm = mp.colors.Normalize()
color_map = mp.colors.LinearSegmentedColormap("my_map",{"red": [(0, 1.0, 1.0),(1.0, .5,
.5)],
                                                        "green": [(0, 0.5, 0.5), (1.0,
0, 0)],
                                                        "blue": [(0, 0.50, 0.5),(1.0, 0
, 0)]})

y_pos
plt.bar(y_pos, score_y, align='center', alpha=0.5, color=color_map(data_norm(like_score
s)))
plt.yticks(np.linspace(min(score_y), max(score_y),10),y_lab)

plt.xticks(y_pos, objects)
plt.ylabel('Function Value')
plt.xlabel('X -->')
plt.title('Function')
plt.grid(True)
locs, labels = plt.xticks()
plt.setp(labels, rotation=90)

plt.show()
```



Problem 4.

Write a function that prints out whether a certain word contains any vowels and count the number of vowels. For example, you can test your function with the following words "Antidisestablishmentarianism" and "Menneskerettighetsorganisasjonene".

In [158]:

```
word_list = ['BCdfgh','Antidisestablishmentarianism','Menneskerettighetsorganisasjonen
e']

def vow_fn(word):
    count = 0
    vowels = ['a','e','i','o','u']
    for i in word.lower():
        if i in vowels:
            count += 1
    return(count)

for i in word_list:
    a = vow_fn(i)
    if a == 0:
        print ('No vowels present in word ', i)
    else:
        print ('Vowels present in word %s and number of vowels is %s' %(i,a))
```

```
No vowels present in word  BCdfgh
Vowels present in word Antidisestablishmentarianism and number of vowels i
s 11
Vowels present in word Menneskerettighetsorganisasjonene and number of vow
els is 13
```

Problem 5

At a certain school, student's email addresses end with @student.college.edu, while professor's email addresses end with @prof.college.edu. Write a program that first asks the user how many email addresses she/he will be entering, and then asks the user to enter those addresses. After all the email addresses are entered, the program should print out a message indicating how many students and professor's have been entered.

In [7]:

```
stud_email = '@student.college.edu'
stud_count = 0
prof_email = '@prof.college.edu'
prof_count = 0
num_email = int(input('Please enter number of emails you will be entering :'))

email_list = []
for num in range(num_email):
    email_list.append(input('Please enter student/professor email :'))

for i in email_list:
    if stud_email in i:
        stud_count += 1
    elif prof_email in i:
        prof_count += 1

print ('Number of emails entered for students is %i and for professors is %i' %(stud_co
unt,prof_count))
```

```
Please enter number of emails you will be entering :5
Please enter student/professor email :steve@student.college.edu
Please enter student/professor email :mike@prof.college.edu
Please enter student/professor email :john@student.college.edu
Please enter student/professor email :anders@student.college.edu
Please enter student/professor email :paul@prof.college.edu
Number of emails entered for students is 3 and for professors is 2
```

Problem 6

Companies often try to personalize their offers to make them more attractive. One simple way to do this is just to insert the person's name at various places in the offer. Of course, companies don't manually type in every person's name; everything is computer-generated. Write a program that asks the user for a name and then generates an offer like the one below.

```
Enter name: Steve Jobs

Dear Steve Jobs,

I am pleased to offer you our new Platinum Plus Rewards card
at a special introductory APR of 47.99 percent.  Steve, an offer
like this does not come along every day, so I urge you to call
now toll-free at 1-800-314-1592. We cannot offer such a low
rate for long, Steve, so call right away.
```

In [159]:

```
f_name, l_name = input('Enter name: ').split()

offer_text = '\nDear %s %s,\n\nI am pleased to offer you our new Platinum Plus Rewards
 card\nat a special introductory APR of 47.99 percent.  %s, an offer\nlike this does no
t come along every day, so I urge you to call\nnow toll-free at 1-800-314-1592. We cann
ot offer such a low\nrate for long, %s, so call right away.'

print (offer_text %(f_name,l_name,f_name,f_name))
```

```
Enter name: Steve Jobs

Dear Steve Jobs,

I am pleased to offer you our new Platinum Plus Rewards card
at a special introductory APR of 47.99 percent.  Steve, an offer
like this does not come along every day, so I urge you to call
now toll-free at 1-800-314-1592. We cannot offer such a low
rate for long, Steve, so call right away.
```

Problem 7

Rewrite the following nested loop as a list comprehension

```
ans = []
for i in range(10):
    for j in range(40):
        if (i%3==0) and (j%2==0) and (i>j):
            ans.append((i, j))
print(ans)
```

In [160]:

```
ans = [(i,j) for i in range(10) for j in range(40) if (i%3==0) if (j%2==0) if (i>j)]
print(ans)
```

```
[(3, 0), (3, 2), (6, 0), (6, 2), (6, 4), (9, 0), (9, 2), (9, 4), (9, 6),
(9, 8)]
```

Problem 8

Suppose we want to create an output dictionary which contains only the odd numbers that are present in the input list (see below) as keys and their cubes as values. Show how to do this using for loops and dictionary comprehension.

To test your function, consider the following list :

```
np.random.seed(100)
list_integer = np.random.randint(0, 1000,size= 200)
```

In [161]:

```
np.random.seed(100)
list_integer = np.random.randint(0, 1000,size= 200)
```

In [162]:

```
# using dictionary comprehension
list_cube = {i:i**3 for i in list_integer if (i%2>0)}
print(list_cube)
```

```
{835: 582182875, 871: 660776311, 855: 625026375, 79: 493039, 53: 148877, 6
55: 281011375, 875: 669921875, 507: 130323843, 649: 273359449, 93: 804357,
667: 296740963, 415: 71473375, 897: 721734273, 141: 2803221, 757: 43379809
3, 723: 377933067, 603: 219256227, 955: 870983875, 135: 2460375, 49: 11764
9, 431: 80062991, 705: 350402625, 317: 31855013, 695: 335702375, 967: 9042
31063, 763: 444194947, 889: 702595369, 617: 234885113, 403: 65450827, 63:
250047, 181: 5929741, 283: 22665187, 369: 50243409, 303: 27818127, 679: 31
3046839, 877: 674526133, 437: 83453453, 273: 20346417, 525: 144703125, 17:
4913, 347: 41781923, 475: 107171875, 979: 938313739, 693: 332812557, 13: 2
197, 185: 6331625, 131: 2248091, 643: 265847707, 719: 371694959, 215: 9938
375, 495: 121287375, 873: 665338617, 811: 533411731, 773: 461889917, 839:
590589719, 95: 857375, 749: 420189749, 631: 251239591, 801: 513922401, 91
5: 766060875, 821: 553387661, 419: 73560059, 407: 67419143, 765: 44769712
5, 521: 141420761, 789: 491169069, 409: 68417929, 677: 310288733, 295: 256
72375, 619: 237176659, 819: 549353259, 243: 14348907, 439: 84604519, 813:
537367797, 485: 114084125, 89: 704969, 429: 78953589, 803: 517781627, 621:
239483061, 67: 300763, 223: 11089567, 473: 105823817, 349: 42508549, 887:
697864103, 245: 14706125, 57: 185193, 383: 56181887, 585: 200201625}
```

In [163]:

```
# using loops
list_cube2 = {}
for i in list_integer:
    if (i%2 > 0):
        list_cube2[i] = i**3
    else:
        pass
print(list_cube2)
```

```
{835: 582182875, 871: 660776311, 855: 625026375, 79: 493039, 53: 148877, 6
55: 281011375, 875: 669921875, 507: 130323843, 649: 273359449, 93: 804357,
667: 296740963, 415: 71473375, 897: 721734273, 141: 2803221, 757: 43379809
3, 723: 377933067, 603: 219256227, 955: 870983875, 135: 2460375, 49: 11764
9, 431: 80062991, 705: 350402625, 317: 31855013, 695: 335702375, 967: 9042
31063, 763: 444194947, 889: 702595369, 617: 234885113, 403: 65450827, 63:
250047, 181: 5929741, 283: 22665187, 369: 50243409, 303: 27818127, 679: 31
3046839, 877: 674526133, 437: 83453453, 273: 20346417, 525: 144703125, 17:
4913, 347: 41781923, 475: 107171875, 979: 938313739, 693: 332812557, 13: 2
197, 185: 6331625, 131: 2248091, 643: 265847707, 719: 371694959, 215: 9938
375, 495: 121287375, 873: 665338617, 811: 533411731, 773: 461889917, 839:
590589719, 95: 857375, 749: 420189749, 631: 251239591, 801: 513922401, 91
5: 766060875, 821: 553387661, 419: 73560059, 407: 67419143, 765: 44769712
5, 521: 141420761, 789: 491169069, 409: 68417929, 677: 310288733, 295: 256
72375, 619: 237176659, 819: 549353259, 243: 14348907, 439: 84604519, 813:
537367797, 485: 114084125, 89: 704969, 429: 78953589, 803: 517781627, 621:
239483061, 67: 300763, 223: 11089567, 473: 105823817, 349: 42508549, 887:
697864103, 245: 14706125, 57: 185193, 383: 56181887, 585: 200201625}
```

Question 2 (15 points)

1. Create a function that inputs a word (string) and returns True (boolean) when there is no repeated letter in the word. For example, the word "troublemakings" should return a True value, and the word "sheet" should return a False.
2. Read the text file `ShakespeareSonets.txt` which contains an extract of THE SONNETS by William Shakespeare in text format. Remove the punctuation marks (such as ",", ".", ":", "'", "?", etc.), and also remove the numbers from the text (such as "1", "2", etc).
3. Select from the text those words which do not contain a repeated letter. Compute the length of each word and plot a histogram with the frequency of these words' length in the text.

In [164]:

```
# Question 2.1

def dupl_fun(input_string):
    output_string = ''

    string_len = len(input_string)

    duplicate_flag = 0
    var_position = 0
    while var_position < string_len:
        for next_pos in range(var_position + 1,string_len):
            if input_string[var_position].lower()  == input_string[next_pos].lower():
                duplicate_flag += 1
                break
            else:
                pass
        var_position += 1
    return (duplicate_flag)

enter_string = input ('Enter a word :')

dup_string = dupl_fun(enter_string)

if dup_string:
    print ('Repeated letter present in the word :', enter_string )
else:
    print ('No repeated letter present in the word :', enter_string)
```

```
Enter a word :Elephant
Repeated letter present in the word : Elephant
```

In [165]:

```
# Question 2.2
f = open('ShakespeareSonets.txt')
file = f.readlines()
f.close()

file_lines = len(file)

punct_list = ['[', ']', '[', '!', '"', '#', '$', '%', '&', '(', ')', '*', '+', ',', '.'
, '/', ':',
              ';', '<', '=', '>', '?', '@', '\'', '^', '_', '`', '|', '{', '}', '~', '-'
, ']']

num_list = ['0', '1', '2', '3', '4', '5', '6', '7' , '8', '9']

f_new = ''

for line in range(0,file_lines):
    for string_char in file[line]:
        if string_char in punct_list or string_char in num_list:
            pass
        else:
            f_new += string_char

print (f_new)
```

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase
That thereby beautys rose might never die
But as the riper should by time decease
His tender heir might bear his memory
But thou contracted to thine own bright eyes
Feedst thy lights flame with selfsubstantial fuel
Making a famine where abundance lies
Thy self thy foe to thy sweet self too cruel
Thou that art now the worlds fresh ornament
And only herald to the gaudy spring
Within thine own bud buriest thy content
And tender churl makst waste in niggarding
  Pity the world or else this glutton be
  To eat the worlds due by the grave and thee

When forty winters shall besiege thy brow
And dig deep trenches in thy beautys field
Thy youths proud livery so gazed on now
Will be a tattered weed of small worth held
Then being asked where all thy beauty lies
Where all the treasure of thy lusty days
To say within thine own deep sunken eyes
Were an alleating shame and thriftless praise
How much more praise deserved thy beautys use
If thou couldst answer This fair child of mine
Shall sum my count and make my old excuse
Proving his beauty by succession thine
  This were to be new made when thou art old
  And see thy blood warm when thou feelst it cold

Look in thy glass and tell the face thou viewest
Now is the time that face should form another
Whose fresh repair if now thou not renewest
Thou dost beguile the world unbless some mother
For where is she so fair whose uneared womb
Disdains the tillage of thy husbandry
Or who is he so fond will be the tomb
Of his selflove to stop posterity
Thou art thy mothers glass and she in thee
Calls back the lovely April of her prime
So thou through windows of thine age shalt see
Despite of wrinkles this thy golden time
  But if thou live remembered not to be
  Die single and thine image dies with thee

In [166]:

```
# Question 2.3
#Select from the text those words which do not contain a repeated letter.

# List (f_list) created for words which do not comtain repelted letter
f_list = []

word = ''
for f_char in f_new:
    if f_char == ' ' or f_char == '\n':
        dup_string = dupl_fun(word)
        if dup_string or word == '' :
            pass
        else:
            f_list.append(word)
        word = ''
    else:
        word += f_char
```

In [167]:

```
#Compute the length of each word and plot a histogram with the frequency of these word
s' length in the text.
f_dict = {}
x = []
for words in f_list:
    f_words_len = len(words)
    x.append(len(words))
    if f_words_len not in f_dict:
        f_dict[f_words_len] = 1
    else:
        f_dict[f_words_len] += 1

x_dict = {}

for i in range(1,len(f_dict) + 1):
    x_dict[i] = f_dict[i]

# (length of each word):(frequency)
print (x_dict)
```

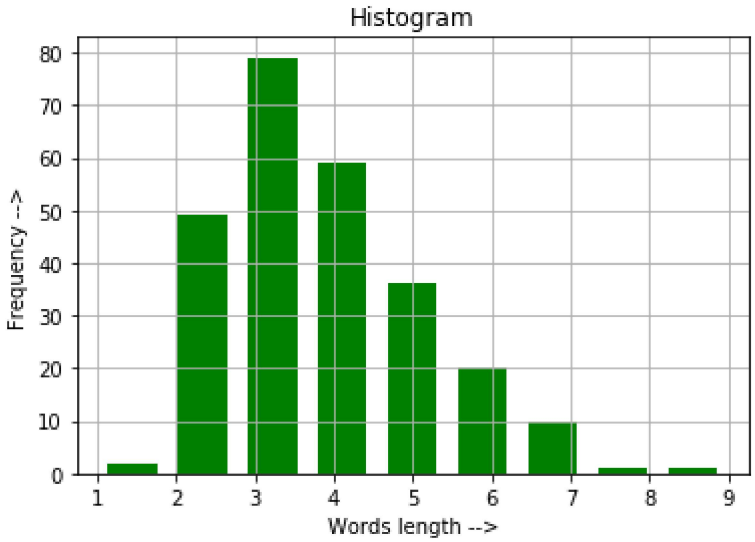
```
{1: 2, 2: 49, 3: 79, 4: 59, 5: 36, 6: 20, 7: 10, 8: 1, 9: 1}
```

In [168]:

```
bins = len(x_dict)

plt.grid(True)
plt.hist(x, bins, histtype='bar',rwidth=0.7, color = 'green')
plt.xlabel('Words length -->')
plt.ylabel('Frequency -->')
plt.title('Histogram')
```

Out[168]:

```
Text(0.5, 1.0, 'Histogram')
```



Question 3 (15 points)

Read the data `data_ECommerce_sample.csv` . The data contains information about customers shopping using a web platform.

1. Show the type and number of observations for each variable.
2. Drop observations for which `CustomerID` is missing. Name the dataframe as `ec`
3. Change the column 'InvoiceData' into a `datetime` . Create a month variable and compute the total monthly quantity sold by the platform.
4. Compute the revenue per observation (call it `Revenue`) as the product of `Quantity` and `UnitPrice` . Display the ten highest invoice orders in terms of revenue.
5. Compute the highest five invoice orders in terms of revenue for the UK, Germany, and France.

In [169]:

```
# 3.1
# Show the type and number of observations for each variable.

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

df_desc = pd.DataFrame(columns = ['Columns', 'Type', 'Observations'])
for i in df.columns:
    df_desc = df_desc.append({'Columns' : i, 'Type' : df[i].dtypes, 'Observations' : df
[i].count()}, ignore_index=True)

# Show the type and number of observations for each variable.
df_desc
```

Out[169]:

	Columns	Type	Observations
0	InvoiceNo	object	8000
1	StockCode	object	8000
2	Description	object	7984
3	Quantity	int64	8000
4	InvoiceDate	object	8000
5	UnitPrice	float64	8000
6	CustomerID	float64	6027
7	Country	object	8000

In [170]:

```
# 3.2
# Drop observations for which CustomerID is missing. Name the dataframe as ec

ec = df.dropna(subset = ['CustomerID'])
ec.head()
```

Out[170]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Countr
0	C544414	22960	JAM MAKING SET WITH JARS	-2	2/18/2011 14:54	3.75	13408.0	Unite Kingdo
2	575656	22952	60 CAKE CASES VINTAGE CHRISTMAS	48	11/10/2011 14:29	0.55	13319.0	Unite Kingdo
3	571636	20674	GREEN POLKADOT BOWL	16	10/18/2011 11:41	1.25	13509.0	Unite Kingdo
4	576657	22556	PLASTERS IN TIN CIRCUS PARADE	12	11/16/2011 11:03	1.65	12720.0	Germar
5	569823	23298	SPOTTY BUNTING	1	10/6/2011 12:15	4.95	16895.0	Unite Kingdo

In [173]:

```
# 3.3
# Change the column 'InvoiceData' into a datetime.
ec.loc[:,'InvoiceDate'] = pd.to_datetime(ec.iloc[:,4])
ec.head()
```

Out[173]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Countr
0	C544414	22960	JAM MAKING SET WITH JARS	-2	2011-02-18 14:54:00	3.75	13408.0	Unite Kingdo
2	575656	22952	60 CAKE CASES VINTAGE CHRISTMAS	48	2011-11-10 14:29:00	0.55	13319.0	Unite Kingdo
3	571636	20674	GREEN POLKADOT BOWL	16	2011-10-18 11:41:00	1.25	13509.0	Unite Kingdo
4	576657	22556	PLASTERS IN TIN CIRCUS PARADE	12	2011-11-16 11:03:00	1.65	12720.0	Germar
5	569823	23298	SPOTTY BUNTING	1	2011-10-06 12:15:00	4.95	16895.0	Unite Kingdo

In [174]:

```
# Create a month variable and compute the total monthly quantity sold by the platform.
ec.loc[:,'Year'] = ec.loc[:,'InvoiceDate'].dt.year
ec.loc[:,'Month'] = ec.loc[:,'InvoiceDate'].dt.month
ec.head()
```

Out[174]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Countr
0	C544414	22960	JAM MAKING SET WITH JARS	-2	2011-02-18 14:54:00	3.75	13408.0	Unite Kingdo
2	575656	22952	60 CAKE CASES VINTAGE CHRISTMAS	48	2011-11-10 14:29:00	0.55	13319.0	Unite Kingdo
3	571636	20674	GREEN POLKADOT BOWL	16	2011-10-18 11:41:00	1.25	13509.0	Unite Kingdo
4	576657	22556	PLASTERS IN TIN CIRCUS PARADE	12	2011-11-16 11:03:00	1.65	12720.0	Germar
5	569823	23298	SPOTTY BUNTING	1	2011-10-06 12:15:00	4.95	16895.0	Unite Kingdo

In [175]:

```
ek = ec.groupby(['Year', 'Month'])['Quantity'].count()
print ('Total monthly quantity sold by the platform : \n', ek)
```

```
Total monthly quantity sold by the platform :
 Year  Month
2010  12       406
2011  1        344
      2        322
      3        426
      4        318
      5        484
      6        375
      7        377
      8        380
      9        586
      10       773
      11       984
      12       252
Name: Quantity, dtype: int64
```

In [176]:

```
# 3.4
#Compute the revenue per observation (call it Revenue) as the product of Quantity and U
nitPrice.
# we keep the credit note for Invoice No. C544414 considering this to be a negative rev
enue
ec.loc[:,'Revenue'] = ec['Quantity'] * ec['UnitPrice']
ec.head()
```

Out[176]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Countr
0	C544414	22960	JAM MAKING SET WITH JARS	-2	2011-02-18 14:54:00	3.75	13408.0	Unite Kingdo
2	575656	22952	60 CAKE CASES VINTAGE CHRISTMAS	48	2011-11-10 14:29:00	0.55	13319.0	Unite Kingdo
3	571636	20674	GREEN POLKADOT BOWL	16	2011-10-18 11:41:00	1.25	13509.0	Unite Kingdo
4	576657	22556	PLASTERS IN TIN CIRCUS PARADE	12	2011-11-16 11:03:00	1.65	12720.0	Germar
5	569823	23298	SPOTTY BUNTING	1	2011-10-06 12:15:00	4.95	16895.0	Unite Kingdo

In [177]:

```
# Delete duplicate entries before selection top 10 revenue items. We can do it on Invoi
ceNo and InvoiceDate
e_top10 = ec.drop_duplicates(subset=['InvoiceDate'])
e_top10[['InvoiceNo','InvoiceDate']].duplicated().head()
```

Out[177]:

```
0    False
2    False
3    False
4    False
5    False
dtype: bool
```

In [178]:

```
#Display the ten highest invoice orders in terms of revenue.
e_top10.sort_values('Revenue', ascending=False).head(10)[['InvoiceNo','Revenue']]
```

Out[178]:

	InvoiceNo	Revenue
3578	540789	835.20
2283	543989	700.80
1070	552172	640.00
3189	559465	622.50
4657	568512	550.00
5241	569815	515.52
850	547708	489.60
1594	551062	432.00
2428	551314	432.00
5339	546464	380.80

In [179]:

```
# 3.5
#Compute the highest five invoice orders in terms of revenue for the UK, Germany, and F
rance.
e_country = ec.drop_duplicates(subset=['InvoiceDate'])
cntry = ['France', 'Germany', 'United Kingdom']

# below sorts the countries with revenue
cntry_list = e_country.groupby(['Country']).apply(lambda x: x.sort_values(['Revenue'],
ascending = False)).reset_index(drop=True).groupby(['Country']).head()

# below displays highest five invoice orders for UK, Germany and France
cntry_list.loc[(cntry_list['Country'].isin(cntry))].reset_index(drop=True)
```

Out[179]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cour
0	540789	21381	MINI WOODEN HAPPY BIRTHDAY GARLAND	576	2011-01-11 11:48:00	1.45	12643.0	Fra
1	573153	23209	LUNCH BAG VINTAGE DOILY	80	2011-10-28 07:39:00	1.65	12678.0	Fra
2	548410	22992	REVOLVER WOODEN RULER	60	2011-03-31 10:29:00	1.95	12731.0	Fra
3	552315	POST	POSTAGE	6	2011-05-08 16:10:00	18.00	12700.0	Fra
4	541405	POST	POSTAGE	5	2011-01-17 15:17:00	18.00	12683.0	Fra
5	568987	47590A	BLUE HAPPY BIRTHDAY BUNTING	21	2011-09-29 15:58:00	5.45	12709.0	Germ
6	539957	22777	GLASS CLOCHE LARGE	12	2010-12-23 12:58:00	7.65	12585.0	Germ
7	571227	20750	RED RETROSPOT MINI CASES	10	2011-10-14 13:54:00	7.95	12477.0	Germ
8	560230	20679	EDWARDIAN PARASOL RED	12	2011-07-17 10:53:00	5.95	12600.0	Germ
9	564337	22326	ROUND SNACK BOXES SET OF4 WOODLAND	24	2011-08-24 14:40:00	2.95	12649.0	Germ
10	552172	16014	SMALL CHINESE STYLE SCISSOR	2000	2011-05-06 13:03:00	0.32	16308.0	Un Kingc
11	559465	47566	PARTY BUNTING	150	2011-07-08 13:26:00	4.15	14680.0	Un Kingc
12	568512	22900	SET 2 TEA TOWELS I LOVE LONDON	200	2011-09-27 12:28:00	2.75	14101.0	Un Kingc
13	569815	22866	HAND WARMER SCOTTY DOG DESIGN	288	2011-10-06 11:53:00	1.79	15838.0	Un Kingc
14	547708	82551	LAUNDRY 15C METAL SIGN	288	2011-03-24 18:34:00	1.70	17450.0	Un Kingc

Question 4 (30 points)

We are going to use three datasets for this question. The description of the variables is as follows:

- The dataset `national_exam_5th_year_level_grade.csv` contains the grade of the 5th-year National Exam in Norway.

Variable	Description
Unnamed: 0	School code
2015_English_Boy	English grade for boys in year 2015
2015_English_girl	English grade for girls in year 2015
...	...

- The dataset `national_exam_5th_year_level_school_info.csv` contains information on schools' location in Norway.

Variable	Description
county code	County code
municipal code	Municipality code
county	County name
municipal	Municipality name
public	True if the school is public and False if the school is private
school	School name
school code	School code

- The dataset `income_nor.csv` contains average household income for each municipality in Norway.

Variable	Description
municipal code	Municipality code
2017	Average household income of each municipality

---

1. Load the data `national_exam_5th_year_level_grade.csv` as a dataFrame `df` . Rename the first column as `school_code` and set it as the row index.
2. Create a hierarchical index for the columns. The index levels should be year, subject, and gender.
3. Show a table with average grades for the whole country by gender and subject in 2017.
4. Select Math and English grades for boys and girls for the year 2017, and call it `dfs` . Plot an histogram for each grade in a two-by-two figure (Please label the x- and y-axis and show the title for each subplot).
5. Load `national_exam_5th_year_level_school_info.csv` as a dataFrame `dfk` . You may notice that there are some strange strings in the column of county, municipal and school. Please replace those values ["¿","\x8c","¾","\x9f","¯","\x81","\x87","\x01"] with ["o", "aa","ae","u","o","aa","a","s"], respectively.
6. Merge DataFrames dfk with dfs on `school_code` . Call the merged dataset `dfall` .
7. Remove the schools with missing grade values and the municipalities with less than five schools.
8. Compute the mean of test grades for each municipality by subject and gender. Call teh dataframe `mean_grade` . Plot a bar chart for the ten municipalities with the highest Math scores for boys.
9. Load `income_nor.csv` as a dataFrame `df_income` . Keep the observations if the year is 2017. Set 'municipal code' as the index.

10. Merge the DataFrame `income` with `mean_grade`.
11. Plot the municipality's income against the average grade of boys and girls in Math using a scatter plot. Please show the legend and label the axis and title.
12. Run two regressions for boys and girls' Math grade on income.

$$\text{grade}_i = \alpha_0 + \alpha_1 \times \text{income}_i$$

where grade is the average math $grade$ for municipality i, and $income$ is the average household income of municipality i.

In [180]:

```
# 4.1
# Load the data `national_exam_5th_year_level_grade.csv` as a dataFrame `df`.
df = pd.read_csv('national_exam_5th_year_level_grade.csv')

# Rename the first column as `school_code` and set it as the row index.
df.rename(columns={df.columns[0]: 'school_code'}, inplace = True)
df.set_index('school_code', inplace = True)

df.head()
```

Out[180]:

school_code	2015_English_Boy	2015_English_girl	2015_Math_Boy	2015_Math_girl	2015_rea
973601997	52.0	51.0	49.0	49.0	
974578263	54.0	54.0	58.0	51.0	
979200293	55.0	52.0	51.0	50.0	
974578212	50.0	47.0	52.0	48.0	
974578220	50.0	50.0	50.0	53.0	

5 rows × 24 columns

In [181]:

```
# 4.2
# Create a hierarchical index for the columns. The index levels should be year, subjec
t, and gender.

list_index = [['2015', '2016', '2017', '2018'],['English', 'Math', 'Reading'], ['Boy',
'Girl']]
ind_col = pd.MultiIndex.from_product(list_index, names = ['Year', 'Subject', 'Gender'])
df.columns = ind_col
df.head()
```

Out[181]:

Year	2015						2016				...	2017		
Subject	English		Math		Reading		English		Math		...	Math		Readi
Gender	Boy	Girl	Boy	Girl	Boy	Girl	Boy	Girl	Boy	Girl	...	Boy	Girl	Boy
school_code														
973601997	52.0	51.0	49.0	49.0	48.0	53.0	54.0	47.0	54.0	48.0	...	50.0	47.0	51.0
974578263	54.0	54.0	58.0	51.0	58.0	53.0	57.0	53.0	55.0	53.0	...	56.0	51.0	54.0
979200293	55.0	52.0	51.0	50.0	51.0	53.0	49.0	47.0	52.0	50.0	...	51.0	48.0	52.0
974578212	50.0	47.0	52.0	48.0	52.0	50.0	51.0	55.0	55.0	56.0	...	50.0	47.0	49.0
974578220	50.0	50.0	50.0	53.0	48.0	51.0	53.0	48.0	51.0	48.0	...	55.0	52.0	51.0

5 rows × 24 columns

In [182]:

```
# 4.3
# Show a table with average grades for the whole country by gender and subject in 2017.
df['2017'].mean()
```

Out[182]:

```
Subject  Gender
English  Boy       50.962246
         Girl      49.464217
Math     Boy       51.022923
         Girl      49.002871
Reading  Boy       49.160232
         Girl      50.844595
dtype: float64
```

In [183]:

```
# 4.4
# Select Math and English grades for boys and girls for the year 2017, and call it `dfs
`.
dfs = df['2017'].iloc[:, 0:4]
dfs.head()
```

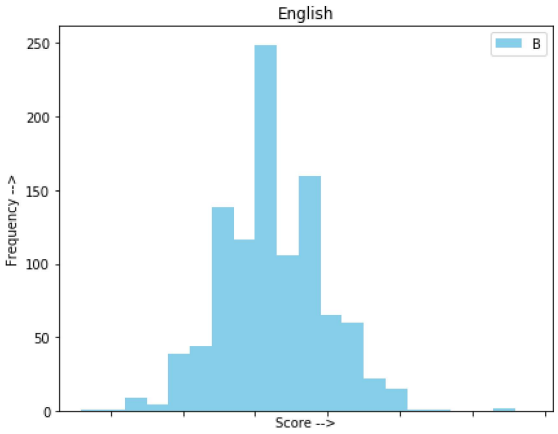
Out[183]:

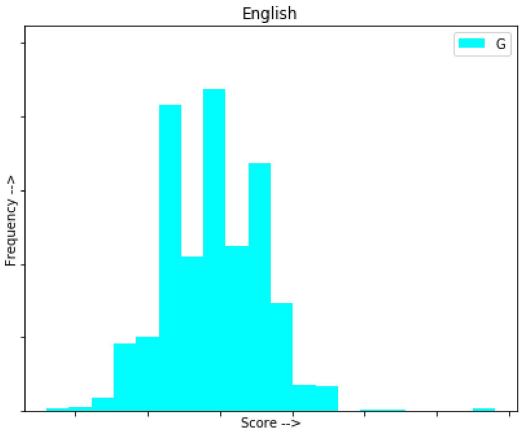
Subject	English		Math	
Gender	Boy	Girl	Boy	Girl
school_code				
973601997	56.0	51.0	50.0	47.0
974578263	56.0	54.0	56.0	51.0
979200293	52.0	51.0	51.0	48.0
974578212	49.0	49.0	50.0	47.0
974578220	51.0	48.0	55.0	52.0

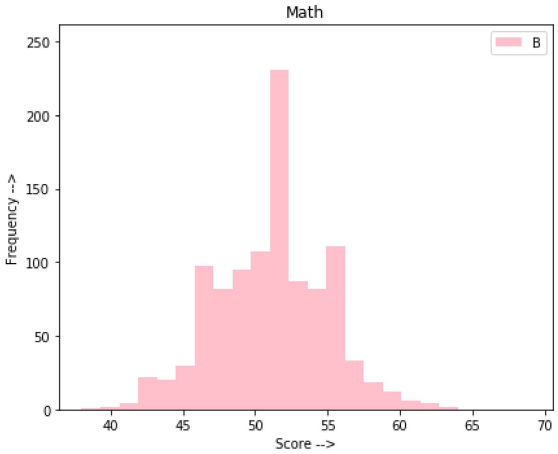
In [185]:

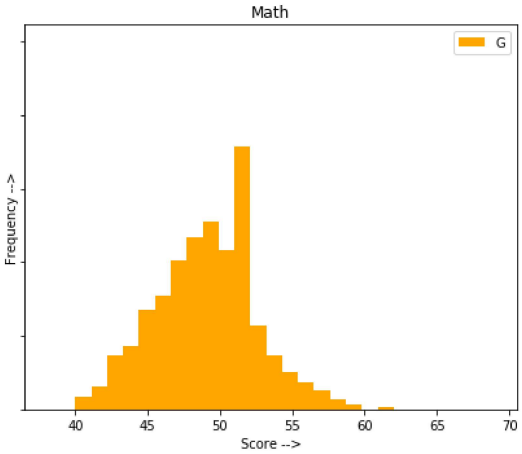
```
# Plot an histogram for each grade in a two-by-two figure (Please label the x- and y-ax
is and show the title for each subplot).
plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=None, hspace=N
one)
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True, figsize=(15,12))

a = 0
x = ['Boy','Girl']
Title = ['English', 'Math']
colors = ['skyblue','cyan','pink','orange']

for i in range(2):
    for j in range(2):
        axes[i,j].hist(dfs.iloc[:,a], bins = 20, color = colors[a])
        axes[i,j].legend(x[j])
        axes[i,j].set_xlabel('Score -->')
        axes[i,j].set_ylabel('Frequency -->')
        axes[i,j].set_title(Title[i])
        a += 1
```

```
<Figure size 432x288 with 0 Axes>
```









In [186]:

```
# 4.5
#Load national_exam_5th_year_level_school_info.csv as a dataFrame dfk.
#You may notice that there are some strange strings in the column of county, municipal
 and school.
#Please replace those values ["¿","\x8c","¾","\x9f","¯","\x81","\x87","\x01"]
#with ["o", "aa","ae","u","o","aa","a","s"], respectively.

dfk = pd.read_csv('national_exam_5th_year_level_school_info.csv')
old_strings = ["¿","\x8c","¾","\x9f","¯","\x81","\x87","\x01"]
new_strings = ["o", "aa","ae","u","o","aa","a","s"]
for i in range(len(old_strings)):
    dfk.replace(to_replace=[old_strings[i]],value=[new_strings[i]], regex = True, inpla
ce = True)

dfk.head()
```

Out[186]:

	county code	municipal code	county	municipal	public	school	school code
0	2.0	220.0	Akershus	Asker	True	Arnestad skole	973601997
1	2.0	220.0	Akershus	Asker	True	Billingstad skole	974578263
2	2.0	220.0	Akershus	Asker	True	Blakstad skole	979200293
3	2.0	220.0	Akershus	Asker	True	Bondi skole	974578212
4	2.0	220.0	Akershus	Asker	True	Drengsrud skole	974578220

In [188]:

```
# 4.6
# Merge DataFrames dfk with dfs on school_code. Call the merged dataset dfall.
dfall = pd.merge(dfs, dfk, left_on = 'school_code', right_on = 'school code')
dfall.head()
```

Out[188]:

	(English, Boy)	(English, Girl)	(Math, Boy)	(Math, Girl)	county code	municipal code	county	municipal	public	scl
0	56.0	51.0	50.0	47.0	2.0	220.0	Akershus	Asker	True	Arne s
1	56.0	54.0	56.0	51.0	2.0	220.0	Akershus	Asker	True	Billing s
2	52.0	51.0	51.0	48.0	2.0	220.0	Akershus	Asker	True	Blak s
3	49.0	49.0	50.0	47.0	2.0	220.0	Akershus	Asker	True	B s
4	51.0	48.0	55.0	52.0	2.0	220.0	Akershus	Asker	True	Dreng s

In [189]:

```
# 4.7
# Remove the schools with missing grade values and the municipalities with less than fi
ve schools.
dfall.dropna(inplace = True)
dfall = dfall.groupby('municipal').filter(lambda x: len(x) >=5)
dfall.head()
```

Out[189]:

	(English, Boy)	(English, Girl)	(Math, Boy)	(Math, Girl)	county code	municipal code	county	municipal	public	scl
0	56.0	51.0	50.0	47.0	2.0	220.0	Akershus	Asker	True	Arne s
1	56.0	54.0	56.0	51.0	2.0	220.0	Akershus	Asker	True	Billing s
2	52.0	51.0	51.0	48.0	2.0	220.0	Akershus	Asker	True	Blak s
3	49.0	49.0	50.0	47.0	2.0	220.0	Akershus	Asker	True	B s
4	51.0	48.0	55.0	52.0	2.0	220.0	Akershus	Asker	True	Dreng s

In [190]:

```
# 4.8

col_list = ['English_Boy', 'English_Girl', 'Math_Boy', 'Math_Girl']
for i in range(4):
    dfall[col_list[i]] = dfall.iloc[:,i]
dfall.drop(dfall.columns[0:4], axis = 1, inplace = True)
dfall.head()
```

Out[190]:

	county code	municipal code	county	municipal	public	school	school code	English_Boy	Engli
0	2.0	220.0	Akershus	Asker	True	Arnestad skole	973601997	56.0	
1	2.0	220.0	Akershus	Asker	True	Billingstad skole	974578263	56.0	
2	2.0	220.0	Akershus	Asker	True	Blakstad skole	979200293	52.0	
3	2.0	220.0	Akershus	Asker	True	Bondi skole	974578212	49.0	
4	2.0	220.0	Akershus	Asker	True	Drengsrud skole	974578220	51.0	

In [191]:

```
# Compute the mean of test grades for each municipality by subject and gender. Call the
dataframe mean_grade.

col_list = ['English_Boy','English_Girl','Math_Boy','Math_Girl']
mean_grade = dfall.groupby('municipal')[col_list].mean()
mean_grade.head()
```

Out[191]:

municipal	English_Boy	English_Girl	Math_Boy	Math_Girl
Alta	50.000000	48.600000	52.400000	48.400000
Arendal	50.363636	47.727273	49.818182	46.181818
Asker	53.692308	51.307692	52.769231	50.230769
Askoy	50.777778	49.333333	52.000000	49.000000
Baerum	53.000000	51.600000	52.960000	50.560000

In [192]:

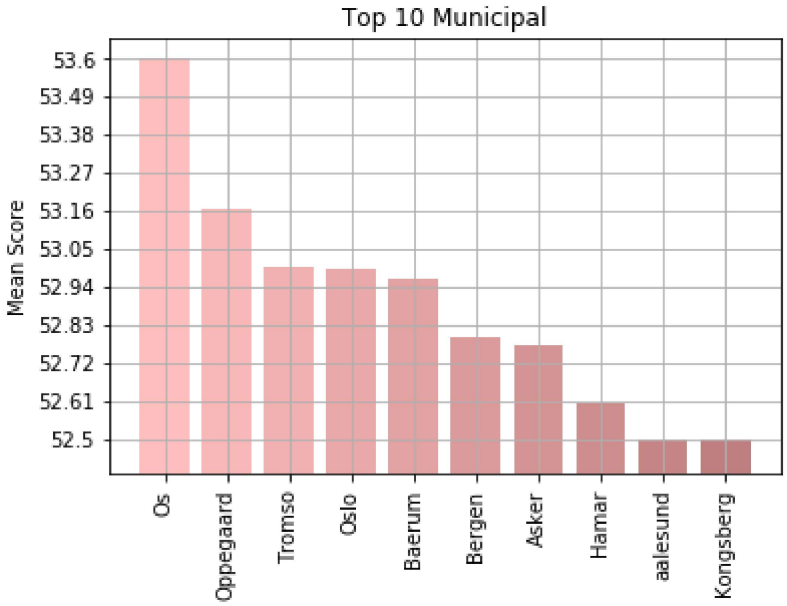
```
# Plot a bar chart for the ten municipalities with the highest Math scores for boys.

top_math_boy = mean_grade.sort_values(by = 'Math_Boy', ascending = False).head(10)
top_math_boy['Math_Boy']
```

Out[192]:

```
municipal
Os           53.600000
Oppegaard    53.166667
Tromso       53.000000
Oslo         52.990099
Baerum       52.960000
Bergen       52.796610
Asker        52.769231
Hamar        52.600000
aalesund     52.500000
Kongsberg    52.500000
Name: Math_Boy, dtype: float64
```

In [193]:

```
# Plot a bar chart
objects = top_math_boy.index
score = top_math_boy.iloc[:,2]

score_y = (score - score.min())/(score.max() - score.min())* score.max()+5
objects = top_math_boy.index
y_pos = np.arange(len(objects))
y_lab = np.round(np.linspace(score.min(),score.max(),11),2)
score = top_math_boy.iloc[:,2]
like_scores = np.array(range(0,len(objects)))
data_norm = mp.colors.Normalize()
color_map = mp.colors.LinearSegmentedColormap("my_map",{"red": [(0, 1.0, 1.0),(1.0, .5,
.5)],
                                                        "green": [(0, 0.5, 0.5), (1.0,
0, 0)],
                                                        "blue": [(0, 0.50, 0.5),(1.0, 0
, 0)]})

plt.bar(y_pos, score_y, align='center', alpha=0.5, color=color_map(data_norm(like_score
s)))
plt.yticks(np.linspace(score_y.min(), score_y.max(),11),y_lab)

plt.xticks(y_pos, objects)
plt.ylabel('Mean Score')
plt.title('Top 10 Municipal')
plt.grid(True)
locs, labels = plt.xticks()
plt.setp(labels, rotation=90)

plt.show()
```



In [194]:

```
# 4.9
# Load income_nor.csv as a dataFrame df_income.
df_income = pd.read_csv('income_nor.csv')
df_income.head()
```

Out[194]:

	06944: Households' income	by region	type of household	contents and year;;;
0	;;;	NaN	NaN	NaN
1	municipal code;2015;2016;2017	NaN	NaN	NaN
2	101;555000;562000;580000	NaN	NaN	NaN
3	104;561000;568000;581000	NaN	NaN	NaN
4	105;566000;578000;593000	NaN	NaN	NaN

In [195]:

```
# Keep the observations if the year is 2017.
# Set 'municipal code' as the index.

df_income[['municipal code','2015','2016','2017']] = df_income.iloc[:,0].str.split(";",
expand=True)
df_income.drop(df_income.columns[0:4], axis = 1, inplace = True)
df_income.drop(df_income.columns[1:3], axis = 1, inplace = True)
df_income = df_income.drop([0,1])
df_income.set_index('municipal code', inplace = True)
df_income.head()
```

Out[195]:

	2017
municipal code	
101	580000
104	581000
105	593000
106	607000
111	703000

In [196]:

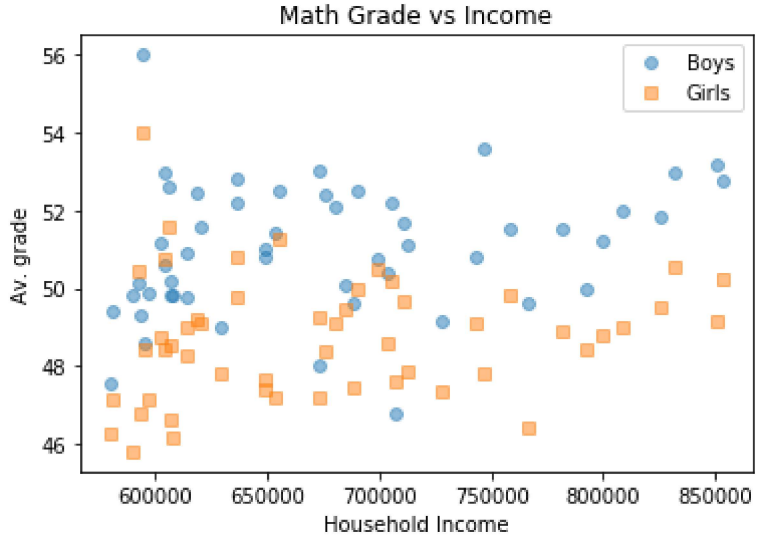
```
# 4.10. Merge the DataFrame `income` with `mean_grade`.
mean_grade2 = dfall.groupby(['municipal code','municipal'])[col_list].mean()
df_income.index = df_income.index.astype(int)
df_inc_grade = pd.merge(df_income, mean_grade2, left_index=True, right_index = True)
df_inc_grade['2017'] = pd.to_numeric(df_inc_grade['2017'],errors = 'coerce')
df_inc_grade.head()
```

Out[196]:

		2017	English_Boy	English_Girl	Math_Boy	Math_Girl
municipal code	**municipal**					
101	**Halden**	580000.0	50.142857	47.428571	47.571429	46.285714
104	**Moss**	581000.0	50.142857	49.285714	49.428571	47.142857
105	**Sarpsborg**	593000.0	50.200000	47.600000	49.300000	46.800000
106	**Fredrikstad**	607000.0	51.705882	50.470588	49.823529	48.529412
213	**Ski**	782000.0	52.250000	50.125000	51.500000	48.875000

In [198]:

```
# 4.11 Plot the municipality's income against the average grade of boys and girls in Ma
th using a scatter plot.
# Please show the legend and label the axis and title.
fig = plt.figure()
plt.plot(df_inc_grade['2017'],df_inc_grade['Math_Boy'], 'o', label='Boys',alpha=0.5)
plt.plot(df_inc_grade['2017'],df_inc_grade['Math_Girl'], 's', label='Girls',alpha=0.5)
plt.xlabel('Household Income')
plt.ylabel('Av. grade')
plt.title('Math Grade vs Income')

plt.legend()
plt.show()
```



In [200]:

```
# 4.12
# Run two regressions for boys and girls' Math grade on income.
# grade i = α0 + α1×income í grade i=a0+a1×income i where grade is the average math grad
e grade for municipality i ïi, and
# income income is the average household income of municipality i.
y_boy = df_inc_grade['Math_Boy']
y_girl = df_inc_grade['Math_Girl']
x = df_inc_grade['2017']
x_addc = sm.add_constant(x)
```

In [201]:

```
reg_boy = sm.OLS(y_boy,x_addc,missing = 'drop')
reg_girl = sm.OLS(y_girl,x_addc,missing = 'drop')
results_boy = reg_boy.fit()
results_girl = reg_girl.fit()
```

In [202]:

```
print (results_boy.summary())
```

```
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                Math_Boy   R-squared:
0.062
Model:                            OLS   Adj. R-squared:
0.043
Method:                 Least Squares   F-statistic:
3.285
Date:                Mon, 14 Oct 2019   Prob (F-statistic):             0.
0759
Time:                        12:06:18   Log-Likelihood:                -9
8.755
No. Observations:                  52   AIC:                             2
01.5
Df Residuals:                      50   BIC:                             2
05.4
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const         47.4178      2.000     23.704      0.000      43.400      5
1.436
2017        5.317e-06   2.93e-06      1.812      0.076   -5.75e-07    1.12
e-05
==============================================================================
====
Omnibus:                        4.911   Durbin-Watson:
2.249
Prob(Omnibus):                  0.086   Jarque-Bera (JB):
5.411
Skew:                           0.231   Prob(JB):                       0.
0668
Kurtosis:                       4.511   Cond. No.                    5.97
e+06
==============================================================================
====

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The condition number is large, 5.97e+06. This might indicate that ther
e are
strong multicollinearity or other numerical problems.
```

In [203]:

```
print (results_girl.summary())
```

```
                            OLS Regression Results
==============================================================================
====
Dep. Variable:              Math_Girl   R-squared:
0.020
Model:                            OLS   Adj. R-squared:
0.001
Method:                 Least Squares   F-statistic:
1.042
Date:                Mon, 14 Oct 2019   Prob (F-statistic):
0.312
Time:                        12:06:20   Log-Likelihood:                -9
6.486
No. Observations:                  52   AIC:                             1
97.0
Df Residuals:                      50   BIC:                             2
00.9
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const         46.8007      1.915     24.439      0.000      42.954      5
0.647
2017        2.867e-06   2.81e-06      1.021      0.312   -2.77e-06    8.51
e-06
==============================================================================
====
Omnibus:                       10.239   Durbin-Watson:
1.780
Prob(Omnibus):                  0.006   Jarque-Bera (JB):                1
0.596
Skew:                           0.820   Prob(JB):                      0.0
0500
Kurtosis:                       4.482   Cond. No.                      5.97
e+06
==============================================================================
====

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
[2] The condition number is large, 5.97e+06. This might indicate that ther
e are
strong multicollinearity or other numerical problems.
```