Exam GRA4142
Data Management and Python Programming
Part II

November 23, 2018

Start: 23.11.18 17:00
Finish: 26.11.18 17:00

Introduction

This exam consists of three parts, and all answers should be uploaded as a zip-file containing a single Jupyter Notebook file (`.ipynb`-file). Python code and SQL queries should be put in "Code" cells, and text answers should be put in either "Code" or "Markdown" cells. Please use comments if you make any assumptions when solving the exercises. Remember to mark each answer with its part and exercise number. The numbers in parenthesis denote the number of points one can obtain for that exercise, and the total maximum score is 100.

You can work in groups of up to three students. Please name the delivery file as `Exam41422_Group_Num#.ipynb` where you should replace `#` by the number of students in your group. Please, also write in the exam the ID number for each student in the group.

Questions about the exercises can be sent to Leif Harald Karlsen at `leifhka@ifi.uio.no`.

Part 1: Retrieving information (50)

These exercises should all be answered over the `northwind`-schema in the `publicDB`-database.

Exercise 1 (2)

Write a `SELECT`-query that finds the price and quantity per unit of the product with name 'Geitost'.

Solution

```
SELECT price, quantityperunit
FROM northwind.products
WHERE productname = 'Geitost'
```

Exercise 2 (2)

Write a SELECT-query that finds name and how many units there are in stock for all products that either have quantities per unit measures in jars or have a name ending in the word 'Sauce'.

Solution

```
SELECT productname, unitsinstock
FROM northwind.products
WHERE productname LIKE '%Sauce' OR
      quantityperunit LIKE '%jars%'
```

Exercise 3 (3)

Write a SELECT-query that finds the name of all supplier companies from USA having either a fax number or a homepage.

Solution

```
SELECT companyname
FROM northwind.suppliers
WHERE country = 'USA' AND
      (fax IS NOT NULL OR homepage IS NOT NULL)
```

Exercise 4 (4)

Write a SELECT-query that finds the first and last name of all employees handling an order with a freight greater than 200.

Solution

```
SELECT DISTINCT e.firstname, e.lastname
FROM northwind.employees AS e,
     northwind.orders AS o
WHERE e.employeeid = o.employeeid AND
      o.freight > 200
```

Exercise 5 (4)

Write a SELECT-query that finds the name, address and city of all customer companies that are located in the same country as the customer company with name 'Eastern Connection'.

Solution

```
SELECT c2.companyname, c2.address, c2.city
FROM northwind.customers AS c1,
     northwind.customers AS c2
WHERE c1.companyname = 'Eastern Connection' AND
      c1.country = c2.country
```

Exercise 6 (5)

Write a SELECT-query that finds the product name and unit price of all beverages that was ordered with a unit price greater than 20.

Solution

```
SELECT DISTINCT p.productname, c.categoryname, d.unitprice
FROM northwind.products AS p,
     northwind.categories AS c,
     northwind.order_details AS d
WHERE p.categoryid = c.categoryid AND
      p.productid = d.productid AND
      c.categoryname = 'Beverages' AND
      d.unitprice > 20
```

Exercise 7 (4)

Write a SELECT-query that finds out how many orders where placed in year 1996.

Solution

```
SELECT count(*)
FROM northwind.orders
WHERE orderdate < '1997-01-01' AND
      orderdate >= '1996-01-01'
```

Exercise 8 (6)

Write a SELECT-query that finds for each category, its name and the number of products in that category. The result should be ordered by the number of

products from most to least.

Solution

```
SELECT c.categoryname, count(productname) AS numberofproducts
FROM northwind.products AS p,
     northwind.categories AS c
WHERE p.categoryid = c.categoryid
GROUP BY c.categoryid
ORDER BY numberofproducts DESC
```

Exercise 9 (6)

Write a `SELECT`-query that finds the average number of products per order.

Solution

```
SELECT avg(c.ordercount) AS avgproductsperorder
FROM (SELECT o.orderid, count(*) AS ordercount
      FROM northwind.orders AS o,
           northwind.order_details AS d
      WHERE o.orderid = d.orderid
      GROUP BY o.orderid) AS c
```

Exercise 10 (7)

Write a `SELECT`-query that finds the first and last name of the youngest employee that has handled orders for a total value greater than 200000.

Solution

```
SELECT e.firstname, e.lastname
FROM (SELECT e.employeeid, sum(unitprice*quantity) AS sold
       FROM northwind.employees AS e,
            northwind.orders AS o,
            northwind.order_details AS d
       WHERE e.employeeid = o.employeeid AND
             o.orderid = d.orderid
       GROUP BY e.employeeid) AS s,
      northwind.employees AS e
WHERE s.sold > 200000 AND
      e.employeeid = s.employeeid
ORDER BY e.birthdate DESC
LIMIT 1
```

Exercise 11 (7)

Write a SELECT-query that returns the following table (with only one row):

hiredfirst	hiredlast
<name1>	<name2>

where <name1> is the full name of the employee that was hired first, and <name2> is the full name of the employee that was hired last.

Solution

```
SELECT e1.firstname || ' ' || e1.lastname AS hiredfirst,
       e2.firstname || ' ' || e2.lastname AS hiredlast
FROM northwind.employees AS e1,
     northwind.employees AS e2,
     (SELECT employeeid FROM northwind.employees
      ORDER BY hiredate LIMIT 1) AS f,
     (SELECT employeeid FROM northwind.employees
      ORDER BY hiredate DESC LIMIT 1) AS l
WHERE e1.employeeid = f.employeeid AND
      e2.employeeid = l.employeeid
```

or

```
SELECT e1.firstname || ' ' || e1.lastname AS hiredfirst,
       e2.firstname || ' ' || e2.lastname AS hiredlast
FROM northwind.employees AS e1, northwind.employees AS e2
ORDER BY e1.hiredate, e2.hiredate DESC
LIMIT 1
```

Part 2: Database design and creation (20)

Exercise 1 (5)

Consider the following table with name MovieActorCinema:

moviename (text)	actor (text)	actorbirthdate (date)	cinema (text)	played (date)
Jurassic Park	Sam Neill	1947-09-14	Ringen Cinema	1993-07-01
Jurassic Park	Sam Neill	1947-09-14	Colloseum	1993-07-03
Free Willy	Lori Petty	1963-10-14	Ringen Cinema	1993-07-13
Free Willy	Jason Richter	1980-01-29	Ringen Cinema	1993-07-13

where the first column contains the name of a movie, the second column contains the name of an actor playing in that movie, the third column contains the birthdate of that actor, the fourth column contains the name of a cinema

the movie was played at, and the last column contains the date which the movie was played at that cinema.

Write down the schema of a collection of normalized tables together containing the same information as this table. You only have to list table names, and the name and types of each column of each table. You should state which columns are keys, and which columns references other columns.

Solution

Need four tables:

- `Movies`:
  - `mid (int)`, primary key
  - `moviename (text)`
- `Actors`:
  - `aid (int)`, primary key
  - `actorname (text)`,
  - `birthdate (date)`,
- `Playsin`:
  - `aid (int)`, references `Actors(aid)`,
  - `mid (int)`, references `Movies(mid)`
- `Cinemas`:
  - `cid (int)`, primary key
  - `cinema (text)`
- `Screenings`:
  - `mid (int)`, references `Movies(mid)`
  - `cid (int)`, references `Cinemas(cid)`
  - `played (date)`

Exercise 2 (5)

Write SQL-queries constructing the normalized tables, with (natural) constraints, and use `SERIAL` for all primary keys.

Solution

```
CREATE TABLE Movies (
    mid SERIAL PRIMARY KEY,
    moviename text NOT NULL
);
CREATE TABLE Actors (
    aid SERIAL PRIMARY KEY,
    actorname text NOT NULL,
    birthdate date
);
CREATE TABLE Playsin (
    aid int REFERENCES Actors(aid),
    mid int REFERENCES Movies(mid)
);
CREATE TABLE Cinema (
    cid SERIAL PRIMARY KEY,
    cinemaname text NOT NULL
);
CREATE TABLE Screenings (
    mid int REFERENCES Movies(mid),
    cid int REFERENCES Cinemas(cid),
    played date NOT NULL
);
```

Exercise 3 (10)

Write SQL-queries that inserts the information from the `MovieActorCinema` table into the normalized tables using `INSERT`-queries with `SELECT`-subqueries over `MovieActorCinema` and the tables you have made. Thus, there should be one `INSERT`-query for each table you have created.

Solution

```
INSERT INTO Movies(moviename)
SELECT DISTINCT moviename
FROM MovieActorCinema

INSERT INTO Actors(actorname, birthdate)
SELECT DISTINCT mac.actor, mac.actorbirthdate, m.mid
FROM MovieActorCinema AS mac, Movies AS m
WHERE m.moviename = mac.moviename

INSERT INTO Playsin
SELECT DISTINCT a.aid, m.mid
```

```
FROM MovieActorCinema AS mac,
     Movies AS m,
     Actors AS a
WHERE mac.moviename = m.moviename AND
      mac.actorname = a.actorname AND
      mac.actorbirthdate = a.actorbirthdate

INSERT INTO Cinemas(cinemaname)
SELECT DISTINCT cinema
FROM MovieActorCinema

INSERT INTO Screenings(mid, cid, played)
SELECT DISTINCT m.mid, c.cid, mac.played
FROM Movies AS m, Cinemas AS c, MovieActorCinema AS mac
WHERE m.moviename = mac.moviename AND
      c.cinema = mac.cinema
```

Part 3: Python and SQL (30)

Given the following skeleton of an analysis program (attached with this exam as a `.py`-file, which can be opened by Jupyter) for our online store that gives the user two options:

```
import psycopg2
import matplotlib.pyplot as plt

connection = "dbname='<db>' port='5432' user='<user>' " + \
             "host='postgresql-lb-1715985356.eu-west-1.elb.amazonaws.com' " + \
             "password='<pwd>'"

def analysis():
    conn = psycopg2.connect(connection)

    ch = 0
    while (ch != 3):
        print(" -- ANALYZER --")
        print("Please choose an option:")
        print("1. Today's stats")
        print("2. Plot orders")
        print("3. Exit")
        ch = int(input("Option: "))

        if (ch == 1):
            todays_stats(conn)
        elif (ch == 2):
            plot_orders(conn)
```

where `<db>` should be subtituted for your own database (e.g. `s123456DB`), `<user>` with your username (e.g. `s123456`) and `<pwd>` with your password.

Exercise 1 (15)

Implement the `todays_stats(conn)` function that should output the following:

> The store has sold a total of `<totaltoday>` today.
>
> The most popular product today is `<todaysproduct>`.

where `<totaltoday>` is the total sum of the prices of all products sold today, `<todaysproduct>` is the name of the product that is sold the most today.

Solution

```
def todays_stats(conn):
    cur = conn.cursor()

    mostsoldq = \
         "SELECT p.name, sum(o.num) AS total " + \
         "FROM .products AS p, orders AS o " + \
         "WHERE p.pid = o.pid  AND o.date = current_date " + \
         "GROUP BY p.pid " + \
         "ORDER BY total LIMIT 1;"

    cur.execute(mostsoldq)

    mostsold = cur.fetchall()

    print("Today's most sold product is", mostsold[0][0])

    totalsoldq = \
         "SELECT sum(p.price * o.num) AS total " + \
         "FROM products AS p, orders AS o " + \
         "WHERE p.pid = o.pid  AND o.date = current_date;"

    cur.execute(totalsoldq)

    totalsold = cur.fetchall()

    print("Total sold today is", totalsold[0][0])
```

Exercise 2 (15)

Implement the `plot_orders(conn)` function that should ask the user for a product name, and do the following:

- If no name was given (i.e. input is the empty string) plot total sold per day (for all products),
- otherwise, plot the same as above, but restrict the orders to only those consisting of the product with the name given by the user.

The x-axis in the plot is time and y-axis is total value sold for. You do not need to display dates on the x-axis, and you can let the first date be day 0, the second date be date 1, and so on.

Solution

```
def plot_orders(conn):

    cur = conn.cursor()

    product = input("Product: ")

    q = \
        "SELECT sum(p.price * o.num) AS sold " + \
        "FROM orders AS o, products AS p " + \
        "WHERE o.pid = p.pid "

    if (product != ""):
        q += "AND p.name = %s "

    q += "GROUP BY o.date "
    q += "ORDER BY o.date;"

    cur.execute(q, [product])

    data = cur.fetchall()

    plt.plot(data)
    plt.show()
```

Final Exam GRA 4142

In [78]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt

%matplotlib inline
```

Part I. (50 points)

Problem 1. (5 points)

Take the following Python code that stores a string:

```
my_str = 'X-DSPAM-Confidence:0.8475'
```

Use `find` and string slicing to extract the portion of the string after the colon character and then convert the extracted string into a floating point number. Please print a line of text as follow:

```
I find a number, 0.8475, in the string.
```

In [73]:
```
my_str = 'X-DSPAM-Confidence:0.8475'
ind=my_str.find(":")+1
```

In [74]:
```
print("I find a number, %3.4f, in the string." %float(my_str[ind:]))
```

```
I find a number, 0.8475, in the string.
```

Problem 2. (5 points)

Consider the following two strings:

```
s1 = "spam"
s2 = "ni!"
```

Write a Python code, that perform string operations on `s1` and `s2` , and produces the following outcome:

a. "NI"

b. "ni!spamni!"

c. "Spam Ni! Spam Ni! Spam Ni!"

d. "maps"

e. ["sp","m"]

f. "spm"

In [1]:
```
s1 = "spam"
s2 = "ni!"
#a
print(s2[0:2].upper())
#b
print(s2+s1+s2)

#c
merge=s1.title()+" "+s2.title()
print(merge*3)

#d
print(s1[::-1])

#e
print([s1[:2],s1[-1]])

#f
print(s1.replace('a',''))
```

```
NI
ni!spamni!
Spam Ni!Spam Ni!Spam Ni!
maps
['sp', 'm']
spm
```

Problem 3. (5 points)

A professor gives 5-point quizzes that are graded on the scale 5-A, 4-B, 3-C, 2-D, 1- F, 0-F. Write a program that accepts a quiz score as an input and prints out the corresponding grade. For example: if the point-score is 4, the code should give 'B' as answer.

In [1]:
```
score = input("Enter score: ")

if int(score)==1 and int(score)==0:
    print("F")
elif int(score)==2:
    print("D")
elif int(score)==3:
    print("C")
elif int(score)==4:
    print("B")
elif int(score)==5:
    print("A")
elif int(score)>5:
    print("Bad score: >5")
```

```
Enter score: 3
C
```

Problem 4. (5 points)

The body mass index (BMI) is calculated as a person's weight (in kilograms), divided by the square of the person's height (in metres). A BMI in the range 19–25, inclusive, is considered healthy. Write a program that calculates a person's BMI and prints a message telling whether they are above, within, or below the healthy range.

In [77]:
```
height=float(input("Height in m: "))
weight=float(input("Weigth in kg: "))

bmi = weight/height**2
if bmi<19:
    print("below health range ",bmi)
elif bmi>=19 and bmi<=25:
    print("within health range ",bmi)
elif bmi>=25:
    print("above health range ",bmi)
```

```
Height in m: 86
Weigth in kg: 8
below health range  0.001081665765278529
```

Problem 5. (8 points)

Initialize a list containing the following elements: 4, 3.1415, 1.0, 'Hello', and 'World'. Then, do the following:

a. Delete 1.0 assuming you know the position on the list? What if you didn't know its position?

b. Append the elements 1.5, 4, 'Hello' to the existing list, to get an extended list as follows:

```
[4,3.1415,1.0,'Hello','World',1.5, 4, 'Hello']
```

c. Arrange the list in reverse order.

d. Count the occurrence of 'Hello' in the extended list.

In [2]:
```
my_lst=[4,3.1415,1.0,'Hello','World']
my_lst.pop(2)
my_lst
```

Out[2]:
```
[4, 3.1415, 'Hello', 'World']
```

In [3]:
```
my_lst=[4,3.1415,1.0,'Hello','World']
my_lst.remove(1.0)
my_lst
```

Out[3]:
```
[4, 3.1415, 'Hello', 'World']
```

In [13]:
```
my_lst=[4,3.1415,1.0,'Hello','World']
my_lst.extend((1.5, 4, 'Hello'))
my_lst
```

Out[13]:
```
[4, 3.1415, 1.0, 'Hello', 'World', 1.5, 4, 'Hello']
```

In [14]:
```
my_lst=[4,3.1415,1.0,'Hello','World']
add=[1.5, 4, 'Hello']
for i in add:
    my_lst.append(i)

my_lst
```

Out[14]:
```
[4, 3.1415, 1.0, 'Hello', 'World', 1.5, 4, 'Hello']
```

In [15]:
```
my_lst[::-1]
```

Out[15]:
```
['Hello', 4, 1.5, 'World', 'Hello', 1.0, 3.1415, 4]
```

In [7]:
```
my_lst.count("Hello")
```

Out[7]:
```
2
```

Problem 6. (6 points)

Ask the user to enter 5 test scores with numerical value between 0 and 100 (integers). Write a program to do the following after all the scores have been entered:

a. Print out the highest and lowest scores.

b. Print out the average of the scores.

c. Print out the second largest score.

In [8]:
```
import numpy as np
```

In [10]:
```
my_list=[]
for i in range(1,6):
    num=float(input(str(i)+". Enter a number between 0 and 100: "))
    my_list.append(num)
print(my_list)
```

```
1. Enter a number between 0 and 100: 3
2. Enter a number between 0 and 100: 4
3. Enter a number between 0 and 100: 5
4. Enter a number between 0 and 100: 6
5. Enter a number between 0 and 100: 7
[3.0, 4.0, 5.0, 6.0, 7.0]
```

In [12]:
```
#### print("Max:", max(my_list))
print("Min:", min(my_list))
print("Max:", max(my_list))

print("Mean",np.mean(my_list))

max2=my_list.copy()
max2.remove(max(my_list))
print("2nd Max",max(max2))
```

```
Min: 3.0
Max: 7.0
Mean 5.0
2nd Max 6.0
```

Problem 7. (6 points)

Consider the following dictionary:

```
commodity = {
    'oil' : 501,
    'copper' : ['upper grade', 'lower grade'],
    'soybean' : ['asia','south america', 'north america']
}
```

Perform the following operations:

a. Add a key to commodity called 'natural gas'.

b. Set the value of 'natural gas' to be a list consisting of the strings 'H1', 'H2', and 'H3'.

c. Sort the items in the list stored under the 'soybean' key in alphabetical order (Make sure the new ordering is updated in the dictionary "commodity").

d. Remove ('upper grade') from the list of items stored under the 'copper' key.

e. Add the number 50 to the number stored under the 'oil' key (to get 551).

In [89]:
```
commodity = {
    'oil' : 501,
    'copper' : ['upper grade', 'lower grade'],
    'soybean' : ['asia','south america', 'north america']
}

commodity['natural gas']=['H1', 'H2', 'H3']
print(commodity)

commodity['soybean'].sort()
print(commodity)

commodity['copper'].remove('upper grade')
commodity

commodity['oil']=commodity['oil']+50
print(commodity)
```

```
{'oil': 501, 'copper': ['upper grade', 'lower grade'], 'soybean': ['asia', 'south
america', 'north america'], 'natural gas': ['H1', 'H2', 'H3']}
{'oil': 501, 'copper': ['upper grade', 'lower grade'], 'soybean': ['asia', 'north
america', 'south america'], 'natural gas': ['H1', 'H2', 'H3']}
{'oil': 551, 'copper': ['lower grade'], 'soybean': ['asia', 'north america', 'sou
th america'], 'natural gas': ['H1', 'H2', 'H3']}
```

In [ ]:
```

```

Problem 8. (10 points)

Write a piece of code to print the lyrics of the song “Old MacDonald.” using a `for` loop. Your program should print the lyrics for three different animals, similar to the example verse below. (Hint: iterate over (cow,moo), (duck,quack), (pig, oink) using dicts)

```
Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!
And on that farm he had a cow, Ee-igh, Ee-igh, Oh!
With a moo, moo here and a moo, moo there.
Here a moo, there a moo, everywhere a moo, moo.
Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!
```

In [2]:
```
# One solution:
animal = ['cow', 'duck', 'pig']
sound = ['moo', 'quack', "oink"]
for i in range(len(animal)):
    print("""
    Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!
    And on that farm he had a %(animals)s, Ee-igh, Ee-igh, Oh!
    With a %(sounds)s, %(sounds)s here and %(sounds)s, %(sounds)s there.
    Here a %(sounds)s, there a %(sounds)s, everywhere a %(sounds)s, %(sounds)s.
    Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!"""%{'animals': animal[i], 'sounds
```

```
    Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!
    And on that farm he had a cow, Ee-igh, Ee-igh, Oh!
    With a moo, moo here and moo, moo there.
    Here a moo, there a moo, everywhere a moo, moo.
    Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!

    Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!
    And on that farm he had a duck, Ee-igh, Ee-igh, Oh!
    With a quack, quack here and quack, quack there.
    Here a quack, there a quack, everywhere a quack, quack.
    Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!

    Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!
    And on that farm he had a pig, Ee-igh, Ee-igh, Oh!
    With a oink, oink here and oink, oink there.
    Here a oink, there a oink, everywhere a oink, oink.
    Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!
```

In [3]:
```
# Alternative solution
x={'animal' :["cow","duck","pig"] ,
    'sound' : ["moo", "quack","oink"],
    'gram' : ["a", "a","an"],}

for i in range(0,3):
    animal=x["animal"][i]
    sound=x["sound"][i]
    gram=x["gram"][i]

    print("Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!")
    print("And on that farm he had a %s, Ee-igh, Ee-igh, Oh!" % animal)
    print("With %s %s, %s here and %s %s, %s there." % (gram,sound,sound,gram,sound
    print("Here %s %s, there %s %s, everywhere %s %s, %s." % (gram,sound,gram,sound
    print("Old MacDonald had a farm, Ee-igh, Ee-igh, Oh! \n")
```

```
Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!
And on that farm he had a cow, Ee-igh, Ee-igh, Oh!
With a moo, moo here and a moo, moo there.
Here a moo, there a moo, everywhere a moo, moo.
Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!

Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!
And on that farm he had a duck, Ee-igh, Ee-igh, Oh!
With a quack, quack here and a quack, quack there.
Here a quack, there a quack, everywhere a quack, quack.
Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!

Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!
And on that farm he had a pig, Ee-igh, Ee-igh, Oh!
With an oink, oink here and an oink, oink there.
Here an oink, there an oink, everywhere an oink, oink.
Old MacDonald had a farm, Ee-igh, Ee-igh, Oh!
```

Part II. (20 points)

Problem 9. (10 points)

We know that money deposited in a bank account earns interest, and this interest accumulates as the years pass. The amount of money $Principle_t$ in each year t follows the specification:

$$Principle_t = Principle_{t-1} * (1 + interest)$$

where $interest$ denotes the annual percentage rate expressed as a decimal number.

Now write an function to determine the future value of an investment. There are 3 inputs in the function: the amount of the investment, the annualized interest rate, and the number of years of the investment.

a. Evaluate the function for the following parameters: interest=0.1, number of years=7, and initial principal = 2000.

b. Evaluate the function for 5 values of the interest rate {.01,.03,.05,.08,.1} (assume that number of years=7, and initial principal = 2000)

In [73]:
```
def future_value(principal,apr,year):
    #print("Year    Value" )
    #print("---------------" )
    #print(" %d    $%5.2f" %(0, principal) )
    for i in range(year):
        principal = principal*(1+apr)
        #print(" %d    $%5.2f" %(i+1, principal) )
    print("The future value is %4.2f" %principal)
    return principal
```

In [74]:
```
future_value(2000,0.1,7)
```

```
The future value is 3897.43
```

Out[74]:
```
3897.434200000001
```

In [76]:
```
for i in [.01,.03,.05,.08,.1]:
    print("The annual interest rate is: ",i)
    future_value(2000,i,7)
    print(" ")
```

```
The annual interest rate is:  0.01
The future value is 2144.27

The annual interest rate is:  0.03
The future value is 2459.75

The annual interest rate is:  0.05
The future value is 2814.20

The annual interest rate is:  0.08
The future value is 3427.65

The annual interest rate is:  0.1
The future value is 3897.43
```

Problem 10. (10 points)

Write your own function called `my_corr` which will compute the correlation coefficient of two arrays. The sample correlation coefficient between arrays x and y is defined as

$$\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2\sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

where x and y are both (nx1) arrays, and i=1,2,3,...,n and $\bar{x}$ and $\bar{y}$ are the sample means of x and y respectively.

Your function should take (nx1) arrays for x and y as inputs and compute the correlation as the output.

You can check your function using numbers from the "math is fun" web site:
https://www.mathsisfun.com/data/correlation.html (https://www.mathsisfun.com/data/correlation.html)

In [1]:
```
import numpy as np
```

In [2]:
```
def my_corr(x,y):
    xbar = np.mean(x)
    ybar = np.mean(y)

    partI   = sum((x-xbar)*(y-ybar))
    partII  = sum((x-xbar)**2)
    partIII = sum((y-ybar)**2)

    corr_xy = partI/np.sqrt(partII*partIII)
    return corr_xy
```

In [3]:
```
np.random.seed(123)
x=np.random.randn(10)
y=np.random.randn(10)
```

In [4]:
```
y
```

Out[4]:
```
array([-0.67888615, -0.09470897,  1.49138963, -0.638902  , -0.44398196,
       -0.43435128,  2.20593008,  2.18678609,  1.0040539 ,  0.3861864 ])
```

In [5]:
```
corr_xy=my_corr(x,y)
corr_xy
```

Out[5]:
```
-0.18198153384151292
```

In [6]:
```
# check with corrcoef function in numpy
np.corrcoef(x,y)[0,1]
```

Out[6]:
```
-0.18198153384151297
```

In [27]:
```
x = np.array([1,2,3])
y = np.array([9,7,8])
np.corrcoef(x,y)
```

Out[27]:
```
array([[ 1. , -0.5],
       [-0.5,  1. ]])
```

Part III. (30 points)

a. Load the data `housing_price.csv` .

In [79]:
```
# Load the data housing_price.csv
path = './data/'
df_long = pd.read_csv('./data/housing_price.csv')

df_long.head()
```

Out[79]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Bathroo
0	Abbotsford	25 Bloomburg St	2	h	1035.0	S	Biggin	4/02/2016	2.5	3067	...	1
1	Abbotsford	5 Charles St	3	h	1465.0	SP	Biggin	4/03/2017	2.5	3067	...	2
2	Abbotsford	55a Park St	4	h	1600.0	VB	Nelson	4/06/2016	2.5	3067	...	1
3	Abbotsford	124 Yarra St	3	h	1876.0	S	Nelson	7/05/2016	2.5	3067	...	2
4	Abbotsford	98 Charles St	2	h	1636.0	S	Nelson	8/10/2016	2.5	3067	...	1

5 rows × 21 columns

b. Select the following variables: `Rooms` , `Type` , `Price` , `Distance` , `Bathroom` , `Bedroom2` . Call the new dataframe "df".

In [80]:
```
df = df_long[["Rooms","Type","Price","Distance","Bathroom","Bedroom2"]]
df.head()
```

Out[80]:

	Rooms	Type	Price	Distance	Bathroom	Bedroom2
0	2	h	1035.0	2.5	1.0	2.0
1	3	h	1465.0	2.5	2.0	3.0
2	4	h	1600.0	2.5	1.0	3.0
3	3	h	1876.0	2.5	2.0	4.0
4	2	h	1636.0	2.5	1.0	2.0

c. Display a table of descriptive statistics of all the variables with numeric values.

In [81]:
```
# display table of statistics of all the variables with numeric values
df.describe()
```

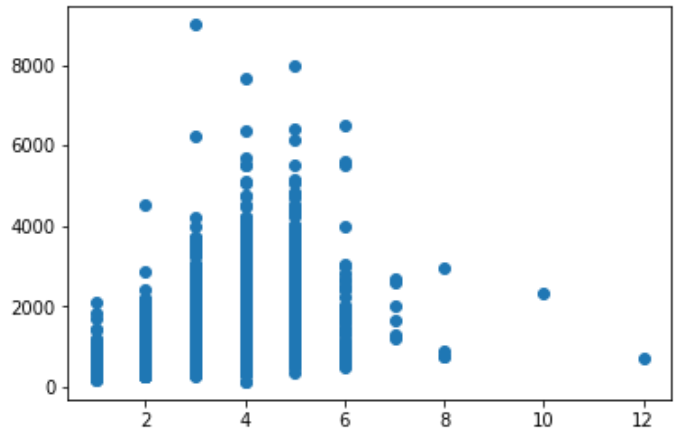
Out[81]:

	Rooms	Price	Distance	Bathroom	Bedroom2
count	7996.000000	7996.000000	7996.000000	7996.000000	7996.000000
mean	3.041396	1088.133535	10.683442	1.626313	3.017884
std	0.976734	680.791381	6.457401	0.728704	0.978214
min	1.000000	131.000000	0.000000	1.000000	0.000000
25%	2.000000	637.750000	6.300000	1.000000	2.000000
50%	3.000000	897.250000	9.700000	2.000000	3.000000
75%	4.000000	1340.000000	13.600000	2.000000	4.000000
max	12.000000	9000.000000	48.100000	9.000000	12.000000

d. Plot in a scatter figure the variable `Rooms` (number of rooms) in the x-axis against `Price` in the y-axis.

In [82]:
```
# plot a scatter plot of `Rooms` (number of rooms) against `Price`
plt.scatter(df["Rooms"],df["Price"])
```

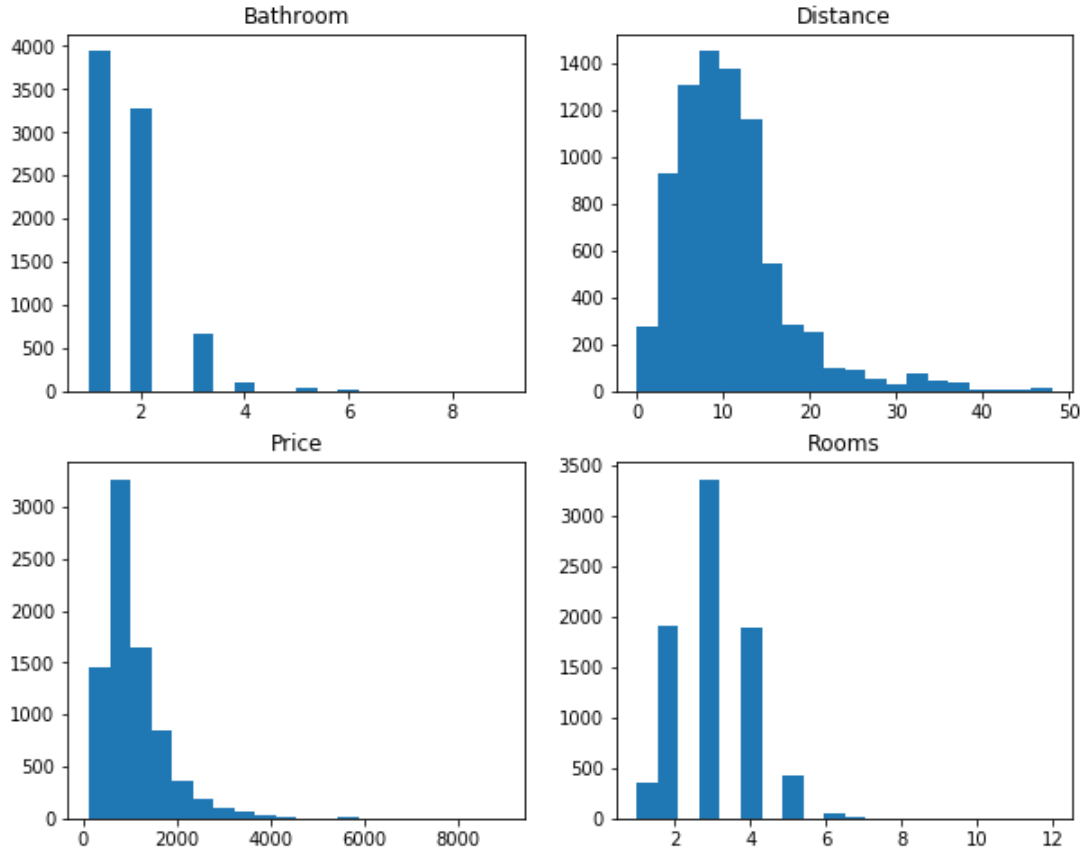
Out[82]:
```
<matplotlib.collections.PathCollection at 0x117a992e8>
```



e. Create a graph with 4 subplots, 2 rows and 2 columns. In each subplot use an histogram to show the distribution of `Bathroom`, `Distance`, `Price`, `Rooms` separately. Please set the title for each subplot.

In [83]:
```
# Plot a distributions of `Bathroom`, `Distance`, `Price`, `Rooms` in a 2-by-2 subp

fig, axes = plt.subplots(nrows=2, ncols=2,figsize=(10, 8))

N=20
axes[0,0].hist(df["Bathroom"],bins=N)
axes[0,0].set_title('Bathroom')

axes[0,1].hist(df["Distance"],bins=N)
axes[0,1].set_title('Distance')

axes[1,0].hist(df["Price"],bins=N)
axes[1,0].set_title('Price')

axes[1,1].hist(df["Rooms"],bins=N)
axes[1,1].set_title('Rooms')
```

Out[83]:
```
Text(0.5, 1.0, 'Rooms')
```



f. Display a table with average values for the variables `Price` and `Rooms` for different types of house.

In [84]:
```
# Display a table of mean of all the variables in the Dataframe `df` grouped by the
df.groupby("Type")[["Price","Rooms"]].mean()
```

Out[84]:

Type	Price	Rooms
h	1248.927370	3.365777
t	909.644374	2.855742
u	592.663567	1.963522

g. Generate a variable for the logarithm of price and run a regression of log price on the number of rooms `Rooms` and a constant term. Please show in a table your estimates.

In [85]:
```
# generate variable log price
df['LogPrice']    = np.log(df["Price"])

# run regression of log price on number of rooms `Rooms` and dummies of the housing
X = df[["Rooms"]]
Y = df["LogPrice"]

X = sm.add_constant(X)
reg1 = sm.OLS(Y, X, missing='drop')

results=reg1.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                LogPrice   R-squared:                       0.311
Model:                            OLS   Adj. R-squared:                  0.311
Method:                 Least Squares   F-statistic:                     3614.
Date:                Fri, 05 Oct 2018   Prob (F-statistic):               0.00
Time:                        12:00:28   Log-Likelihood:                -4921.8
No. Observations:                7996   AIC:                             9848.
Df Residuals:                    7994   BIC:                             9862.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.9019      0.016    360.291      0.000       5.870       5.934
Rooms          0.3083      0.005     60.119      0.000       0.298       0.318
==============================================================================
Omnibus:                       76.329   Durbin-Watson:                   1.325
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               93.403
Skew:                           0.166   Prob(JB):                     5.22e-21
Kurtosis:                       3.412   Cond. No.                         11.4
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.

/Users/linlin/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
e/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/s
table/indexing.html#indexing-view-versus-copy)
```

In [ ]:

GRA41422 - Data management

Exam 2019 – Evaluation Guidelines

Part 1 SQL - ExamAnswer_Part1.sql

```sql
-- Exercices (1-10) in the northwind database
use Northwind

--Excercise 1
--Write a SELECT-query that finds the company name and city of the customer with contact name Hanna Moos.
SELECT CompanyName, City
FROM Customers
WHERE ContactName = 'Hanna Moos'

--Excercise 2
-- Write a SELECT-query that finds the name and unit price of products with quantity per unit measured in packages of jars (e.g. 12 -
200 ml jars) or where the product name ends with 'Syrup'
SELECT ProductName, UnitPrice
FROM Products
WHERE QuantityPerUnit LIKE '%jars' OR ProductName LIKE '%syrup'


-- Excercice 3
-- Write a SELECT-query that finds the name and city of customers in the UK for which a region exists, or which has a postal code
starting with 'WX'
SELECT  CompanyName, City
FROM customers
WHERE country = 'uk'
       AND ( Region IS NOT NULL OR PostalCode like 'WX%' )


-- Excercise 4
```

```sql
-- Write a SELECT-query that finds the first and last name as well as the hire date of employees handling sales that where shipped
before October 1st 1996. The result set should contain a minimal number of rows and sorted starting with those hired first.
SELECT DISTINCT FirstName, LastName, HireDate
FROM Employees e
JOIN Orders o ON o.EmployeeID = e.EmployeeID
WHERE  o.ShippedDate < '1996-10-01'
ORDER BY HireDate


-- Excercise 5
-- Write a SELECT-query that lists product name, quantity per unit and unit price for all products with the same category as the
product with name 'Queso Cabrales', sorted by product name
-- With self-relation ...
SELECT pr.ProductName, pr.QuantityPerUnit, pr.UnitPrice
FROM Products p
JOIN Products pr on pr.CategoryID = p.CategoryID
WHERE p.ProductName='Queso Cabrales'
ORDER BY pr.ProductName

-- or with singel value sub-query
SELECT ProductName, QuantityPerUnit, UnitPrice
FROM Products p
WHERE CategoryID =  (
       SELECT CategoryID
       FROM Products
       WHERE ProductName= 'Queso Cabrales'
       )
ORDER BY ProductName

-- Excercise 6
-- Write a SELECT-query that lists product name and nominal discount for products sold by sales representatives and for which the
nominal discount is more than 10 percent for.
-- (Tip: The nominal discount is a table column named 'Discount')
SELECT DISTINCT  p.ProductName, od.Discount [nominal discount]
FROM [Order Details] od
JOIN Orders o ON o.OrderID = od.OrderID
JOIN Employees e  ON e.EmployeeID = o.EmployeeID and e.Title = 'sales representative'
JOIN Products p ON p.ProductID = od.ProductID
WHERE od.Discount > 0.10
ORDER BY p.ProductName
```

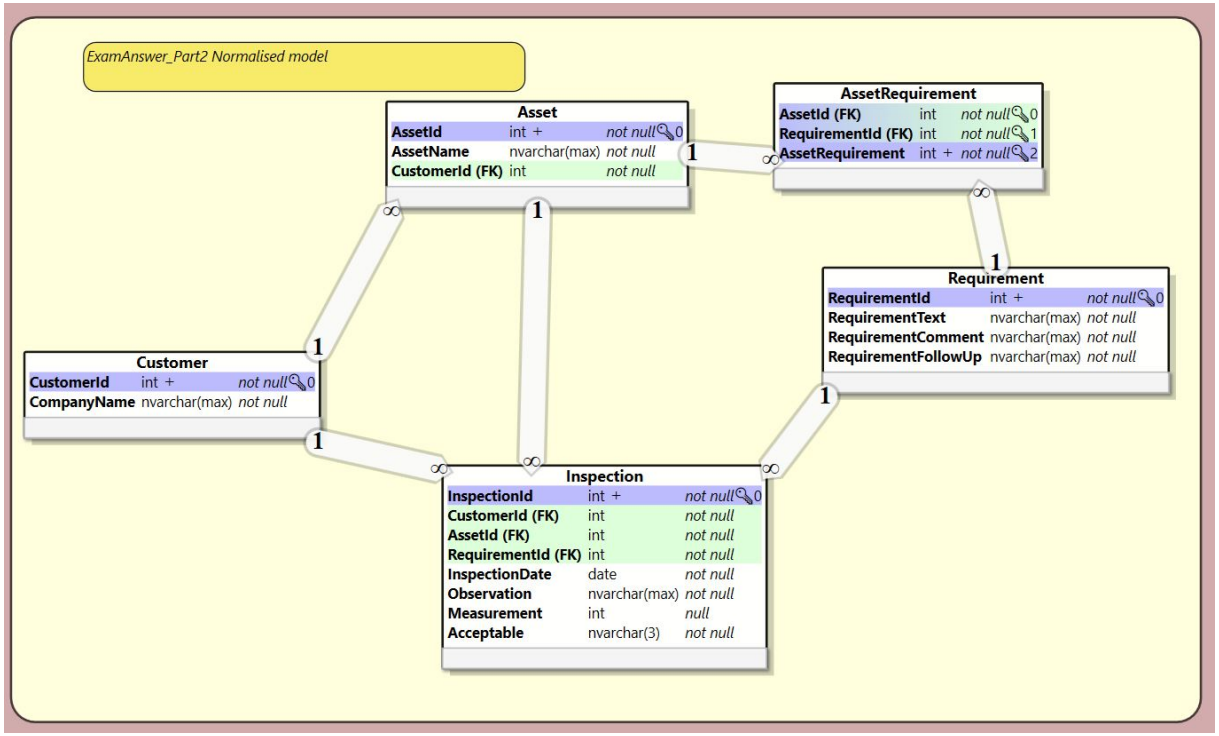
```sql
-- Excercise 7
-- Write a SELECT-query that establishes how many orders were placed for seafood
-- With subquery
SELECT count(*) seafoodOrders
FROM Orders
WHERE OrderID IN (
       SELECT od.OrderID
       FROM [Order Details] od
       JOIN Products p on p.ProductID = od.ProductID
       JOIN Categories c on c.CategoryID = p.CategoryID
       WHERE c.CategoryName = 'seafood'
)

-- or, directly:
SELECT count(DISTINCT o.OrderID) seafoodOrders
FROM Orders  o
JOIN [Order Details] od ON o.OrderID =  od.OrderID
JOIN Products p ON p.ProductID = od.ProductID
JOIN Categories c ON c.CategoryID = p.CategoryID
WHERE c.CategoryName = 'seafood'


-- Excercise 8
-- Write a SELECT-query that lists employee full name and total number of orders placed with that employee, starting with employees
with most orders
SELECT e.FirstName+' '+e.LastName fullName, count(DISTINCT o.OrderID) totalOrders
FROM [Order Details] od
JOIN Orders o ON o.OrderID = od.OrderID
JOIN Employees e  ON e.EmployeeID = o.EmployeeID
GROUP  BY e.FirstName+' '+e.LastName
ORDER BY 2 desc

-- Excercise 9
-- Write a SELECT-query that calculates the average number of columns  per base table in the Northwind database omitting the
sysdiagrams table.
-- Tip : INFORMATION_SCHEMA.COLUMNS and INFORMATION_SCHEMA.TABLES contain information about tables and columns
SELECT AVG(s.columnCount)
FROM (
       SELECT c.TABLE_NAME, count(COLUMN_NAME) columnCount
       FROM INFORMATION_SCHEMA.COLUMNS c
```

```
	join INFORMATION_SCHEMA.TABLES t on t.TABLE_NAME = c.TABLE_NAME and t.TABLE_TYPE='BASE TABLE' and  t.TABLE_NAME <>
'sysdiagrams'
	GROUP BY c.TABLE_NAME
	) s


--Excercise 10
-- Given the product which has the highest total value in stock, write a SELECT-query that gives the company name and product name of
the customer with the highest amount spent for that product. The product should still be allowed to sell (Discontinued=0)
SELECT TOP 1 cu.CompanyName, ProductName
FROM customers cu
JOIN (
	SELECT  o.CustomerID, od.ProductID,  sum(od.Quantity* od.UnitPrice) soldValue
	FROM Orders o
	JOIN [Order Details] od ON od.OrderID=o.OrderID
	GROUP BY o.CustomerID, od.ProductID
) topCustomer ON topCustomer.CustomerID = cu.CustomerID
JOIN (
	SELECT TOP 1 ProductID,  ProductName,  (UnitPrice * UnitsInStock) ValueInStock
	FROM Products
	WHERE Discontinued =0
	ORDER BY (UnitPrice * UnitsInStock) DESC
) topViS ON topViS.ProductID =  topCustomer.ProductID
ORDER BY soldValue DESC


-- Excercise 11
-- Background: A database with to tables, Customer and ExportHistory are used for book-keeping transferring data from the customer
system as indicated in the diagram.
-- The aim is to limit the number of rows to be treated to only the rows in the Customer database that are new or have changed since
the last export.
-- Write the appropriate SQL-Query to SELECT all data in these rows from Customer.
-- Note: After the changes have been applied the LastUpdate is updated for that row in ExportHistory with todays's date

SELECT *
FROM Customer
WHERE customerId IN (
	SELECT customerId
	FROM Customer
	WHERE customerId NOT IN (SELECT customerId FROM ExportHistory)
```

```
       UNION
       SELECT c.customerId
       FROM Customer c
       JOIN ExportHistory h on h.CustomerId = c.CustomerId
       WHERE c.LastUpdate > h.LastUpdate
       )


-- or with LEFT OUTER JOIN

SELECT *
FROM Customer
WHERE customerId IN (
       SELECT customerId
       FROM Customer
       LEFT JOIN ExportHistory h on h.CustomerId = c.CustomerId
       WHERE h.CustomerId IS NULL
       UNION
       SELECT c.customerId
       FROM Customer c
       JOIN ExportHistory h on h.CustomerId = c.CustomerId
       WHERE c.LastUpdate > h.LastUpdate
       )

----- End of Part 1
```

Part 2 – Normalisation - Inspection database

ExamAnswer_Part2_Model



The one-to-many relation from Customer to Asset as well as the many-to-many between Asset and Requirement is more implied than explicit in the non-normalised table. These relations are not required by the students.

ExamAnswer_Part2.SQL

```
-- ExamAnswer_Part2_Preamble.sql  -- Preamble provided as part of the Exam input
go
```

```
use Northwind
go
drop database inspection
go

go
create database Inspection
go
use Inspection
go
create table NonNormalisedInspection (CompanyName nvarchar(max) not null, Asset nvarchar(max) not null
, InspectionDate Date , Requirement nvarchar(max) not null, RequirementComment nvarchar(max) not null
, Observation nvarchar(max) not null, Measurement nvarchar(max), Acceptable nvarchar(3), RequirementFollowUp nvarchar(max) )
go
insert into NonNormalisedInspection (CompanyName , Asset, InspectionDate , Requirement , RequirementComment
, Observation , Measurement, Acceptable, RequirementFollowUp) values
 ('Ocean Traveler', 'Atlantic traveler', '2015-05-01', 'Check propeller condition'    , 'It should be smooth'   , 'Looks OK'        ,
'(Not applicable)', 'Yes', '')
,('Ocean Traveler', 'Atlantic traveler', '2015-05-01', 'Measure clearing of propeller', 'Minimum 30 millimeters', '(Not applicable)',
'55'              , 'Yes', '')
,('Ocean Traveler', 'Atlantic traveler', '2017-05-01', 'Check propeller condition'    , 'It should be smooth'   , 'Propeller has
dents', '(Not applicable)', 'No', 'Replace propeller')
,('Ocean Traveler', 'Atlantic traveler', '2017-05-01', 'Measure clearing of propeller', 'Minimum 30 millimeters', '(Not applicable)',
'25'                , 'No', 'Replace bearing')
,('Ocean Traveler', 'Atlantic traveler', '2019-06-01', 'Check propeller condition'    , 'It should be smooth'   , 'Looks OK'        ,
'(Not applicable)', 'Yes', '')
,('Ocean Traveler', 'Atlantic traveler', '2019-06-01', 'Measure clearing of propeller', 'Minimum 30 millimeters', '(Not applicable)',
'60'              , 'Yes', '')

,('Ocean Traveler', 'Pacific traveler', '2015-05-01', 'Check propeller condition'    , 'It should be smooth'   , 'Looks OK'        ,
'(Not applicable)', 'Yes', '')
,('Ocean Traveler', 'Pacific traveler', '2015-05-01', 'Measure clearing of propeller', 'Minimum 30 millimeters', '(Not applicable)',
'53'              , 'Yes', '')
,('Ocean Traveler', 'Pacific traveler', '2017-05-01', 'Check propeller condition'    , 'It should be smooth'   , 'Looks OK'        ,
'(Not applicable)', 'Yes', '')
,('Ocean Traveler', 'Pacific traveler', '2017-05-01', 'Measure clearing of propeller', 'Minimum 30 millimeters', '(Not applicable)',
'50'              , 'Yes','')
,('Ocean Traveler', 'Pacific traveler', '2019-06-01', 'Check propeller condition'    , 'It should be smooth'   , 'Looks OK'        ,
'(Not applicable)', 'Yes', '')
```

```
,('Ocean Traveler', 'Pacific traveler', '2019-06-01', 'Measure clearing of propeller', 'Minimum 30 millimeters', '(Not applicable)',
'49'              , 'Yes', '')
,('Star fisheries', 'Morning star', '2015-06-15', 'Check propeller condition'    , 'It should be smooth'   , 'Looks OK'        , '(Not
applicable)', 'Yes', '')
,('Star fisheries', 'Morning star', '2017-06-10', 'Check propeller condition'    , 'It should be smooth'   , 'Looks OK'        , '(Not
applicable)', 'Yes', '')
,('Star fisheries', 'Morning star', '2019-06-09', 'Check propeller condition'    , 'It should be smooth'   , 'Looks OK'        , '(Not
applicable)', 'Yes', '')

----------- End of ExamAnswer_Part2_Preamble.sql

-- ExamAnswer_Part2  (Student part)

-- Excercise 1

-------------------------------------------------------------------------------------------
go
-- Model New Model
-- Updated 13.11.2019 18:01:04
-- DDL Generated 13.11.2019 18:01:06

--**********************************************************************
--     Tables
--**********************************************************************

-- Table dbo.Asset
create table
       [dbo].[Asset]
(
       [AssetId] int identity(0,1) not null
       , [AssetName] nvarchar(max) not null
       , [CustomerId] int not null
,
constraint [Pk_Asset_AssetId] primary key clustered
(
       [AssetId] asc
)
);

-- Table dbo.AssetRequirement
```

```
create table
	[dbo].[AssetRequirement]
(
	[AssetId] int not null
	, [RequirementId] int not null
	, [AssetRequirement] int identity(0,1) not null
,
constraint [Pk_AssetRequirement_AssetIdRequirementIdAssetRequirement] primary key clustered
(
	[AssetId] asc
	, [RequirementId] asc
	, [AssetRequirement] asc
)
);

-- Table dbo.Customer
create table
	[dbo].[Customer]
(
	[CustomerId] int identity(0,1) not null
	, [CompanyName] nvarchar(max) not null
,
constraint [Pk_Customer_CustomerId] primary key clustered
(
	[CustomerId] asc
)
);

-- Table dbo.Inspection
create table
	[dbo].[Inspection]
(
	[InspectionId] int identity(0,1) not null
	, [CustomerId] int not null
	, [AssetId] int not null
	, [RequirementId] int not null
	, [InspectionDate] date not null
	, [Observation] nvarchar(max) not null
	, [Measurement] int null
	, [Acceptable] nvarchar(3) not null
```

```
,
constraint [Pk_Inspection_InspectionId] primary key clustered
(
	[InspectionId] asc
)
);

-- Table dbo.Requirement
create table
	[dbo].[Requirement]
(
	[RequirementId] int identity(0,1) not null
	, [RequirementText] nvarchar(max) not null
	, [RequirementComment] nvarchar(max) not null
	, [RequirementFollowUp] nvarchar(max) not null
,
constraint [Pk_Requirement_RequirementId] primary key clustered
(
	[RequirementId] asc
)
);
--************************************************************************
--	Data
--************************************************************************
--************************************************************************
--	Relationships
--************************************************************************

-- Relationship Fk_Customer_Asset_CustomerId
alter table [dbo].[Asset]
add constraint [Fk_Customer_Asset_CustomerId] foreign key ([CustomerId]) references [dbo].[Customer] ([CustomerId]);


-- Relationship Fk_Asset_AssetRequirement_AssetId
alter table [dbo].[AssetRequirement]
add constraint [Fk_Asset_AssetRequirement_AssetId] foreign key ([AssetId]) references [dbo].[Asset] ([AssetId]);


-- Relationship Fk_Requirement_AssetRequirement_RequirementId
alter table [dbo].[AssetRequirement]
```

```
add constraint [Fk_Requirement_AssetRequirement_RequirementId] foreign key ([RequirementId]) references [dbo].[Requirement]
([RequirementId]);


-- Relationship Fk_Customer_Inspection_CustomerId
alter table [dbo].[Inspection]
add constraint [Fk_Customer_Inspection_CustomerId] foreign key ([CustomerId]) references [dbo].[Customer] ([CustomerId]);


-- Relationship Fk_Asset_Inspection_AssetId
alter table [dbo].[Inspection]
add constraint [Fk_Asset_Inspection_AssetId] foreign key ([AssetId]) references [dbo].[Asset] ([AssetId]);


-- Relationship Fk_Requirement_Inspection_RequirementId
alter table [dbo].[Inspection]
add constraint [Fk_Requirement_Inspection_RequirementId] foreign key ([RequirementId]) references [dbo].[Requirement]
([RequirementId]);


----------------------------------------
-- Exercise 2 - populating the normalised tables
insert into Customer
select distinct CompanyName
from NonNormalisedInspection

-- select * from Customer

insert into Asset
select distinct Asset, c.CustomerId
from NonNormalisedInspection n
join Customer c on c.CompanyName = n.CompanyName

-- select * from Asset

insert into Requirement
select distinct Requirement, RequirementComment, RequirementFollowUp
from NonNormalisedInspection n
where RequirementFollowUp <> ''
```

```
insert into Inspection
select a.CustomerId, a.AssetId, r.RequirementId, n.InspectionDate, n.Observation
       , case when n.Measurement='(Not Applicable)' then null else n.Measurement end
       , n.Acceptable
from NonNormalisedInspection n
-- join Customer c on c.CompanyName = n.CompanyName
join Asset a on a.AssetName = n.Asset
join Requirement r on r.RequirementText = n.Requirement
go
-----------------------------------------
-- Exercise 3 - create the InspectionView
go
create view InspectionView as
       select c.CompanyName, a.AssetName Asset
       , i.InspectionDate
       , r.RequirementText Requirement, r.RequirementComment, i.Observation
       , case when i.Measurement is null then '(Not applicable)' else convert(varchar(10), i.measurement ) end Measurement
       , i.Acceptable
       , case when i.Acceptable='No' then  r.RequirementFollowUp else '' end RequirementFollowUp -- select *
       from Inspection i
       left join Customer c on c.CustomerId = i.CustomerId
       left join Asset a  on a.AssetId = i.AssetId
       left join Requirement r on r.RequirementId = i.RequirementId
go
--------------------
-- Just testing, not required for the answer
select * from NonNormalisedInspection order by 1,2,3
select * from InspectionView order by 1,2,3

---------- End of Part 2
```

ExamAnswer_Part3.ipynb

In [1]:
```
# Excercise 1
import pyodbc
conn = pyodbc.connect('Driver={SQL SERVER};Trusted_Connection=yes;'
                      'Database=Inspection;Server=(local);')
cursor = conn.cursor()
cursor.execute("SELECT DISTINCT CompanyName, Asset FROM NonNormalisedInspection ORDER BY  CompanyName, Asset ")

for row in cursor:
    print(row)
```

```
('Ocean Traveler', 'Atlantic traveler')
('Ocean Traveler', 'Pacific traveler')
('Star fisheries', 'Morning star')
```

In [2]:
```
# Excercise 2
sql = """\
    insert into NonNormalisedInspection
    (CompanyName , Asset, InspectionDate , Requirement , RequirementComment,
    Observation , Measurement, Acceptable, RequirementFollowUp)
    values(?, ?, ?, ?, ?, ?, ?, ?, ?)
    """
params = [  ('Moon', 'Moonlight', '2010-02-01', 'Check propeller condition' \
                , 'It should be smooth'   , 'Looks OK'        , '(Not applicable)', 'Yes', '') \
            , ('Moon', 'Beam', '2010-04-04', 'Check propeller condition' \
                , 'It should be smooth'   , 'Looks OK'        , '(Not applicable)', 'Yes', '')]

cursor.executemany(sql, params)
```

In [3]:

```
#Exercise 3
try:
    conn.autocommit = False

    sql = """\
         insert into NonNormalisedInspection
                (CompanyName , Asset, InspectionDate , Requirement , RequirementComment,
                Observation , Measurement, Acceptable, RequirementFollowUp)
            values(?, ?, ?, ?, ?, ?, ?, ?, ?)
    """
    params = [  ('Sun', 'Shine', '2012-12-12', 'Check propeller condition' \
                , 'It should be smooth'   , 'Looks OK'        , '(Not applicable)', 'Yes', '') ]
    cursor.executemany(sql, params)
    sql = """\
    update t set CompanyName = 'Global Traveler'
    from NonNormalisedInspection t
    where CompanyName = 'Ocean Traveler'
    """
    cursor.execute(sql)
except pyodbc.DatabaseError as err:
    conn.rollback()
else:
    conn.commit()
finally:
    conn.autocommit = True
```

```
#Exercise 3
try:
    conn.autocommit = False

    sql = """\
         insert into NonNormalisedInspection
                (CompanyName , Asset, InspectionDate , Requirement , RequirementComment,
                Observation , Measurement, Acceptable, RequirementFollowUp)
            values(?, ?, ?, ?, ?, ?, ?, ?, ?)
    """
    params = [  ('Sun', 'Shine', '2012-12-12', 'Check propeller condition' \
                , 'It should be smooth'   , 'Looks OK'        , '(Not applicable)', 'Yes', '') ]
    cursor.executemany(sql, params)
    sql = """\
    update t set CompanyName = 'Global Traveler',yyy
    from NonNormalisedInspection t
    where CompanyName = 'Ocean Traveler'
    """
    cursor.execute(sql)
except pyodbc.DatabaseError as err:
    #Excercise 4 - start
    # Exception handling e.g. https://stackoverflow.com/questions/11392709/how-to-catch-specific-pyodbc-error-message
    print ("Rollback for error:"+err.args[1])
    #Excercise 4 - end
    conn.rollback()
else:
    conn.commit()
finally:
    conn.autocommit = True
```

```
Rollback for error:[42000] [Microsoft][ODBC SQL Server Driver][SQL Server]Incorrect syntax near the keyword 'from'. (156) (SQLE
xecDirectW)
```

```
#Exercise 5
cursor.execute("SELECT DISTINCT CompanyName, Asset,  InspectionDate  FROM NonNormalisedInspection ORDER BY InspectionDate, Asset

for row in cursor:
    print(row)
```

```
('Moon', 'Moonlight', '2010-02-01')
('Moon', 'Beam', '2010-04-04')
('Sun', 'Shine', '2012-12-12')
('Ocean Traveler', 'Atlantic traveler', '2015-05-01')
('Ocean Traveler', 'Pacific traveler', '2015-05-01')
('Star fisheries', 'Morning star', '2015-06-15')
('Ocean Traveler', 'Atlantic traveler', '2017-05-01')
('Ocean Traveler', 'Pacific traveler', '2017-05-01')
('Star fisheries', 'Morning star', '2017-06-10')
('Ocean Traveler', 'Atlantic traveler', '2019-06-01')
('Ocean Traveler', 'Pacific traveler', '2019-06-01')
('Star fisheries', 'Morning star', '2019-06-09')
```

Exam GRA4142-Solutions

Add group member here:

ID 1: XXXX

ID 2: XXXX

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.

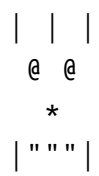
In [2]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from IPython.display import Image
from scipy import linalg

%matplotlib inline
```

In [ ]:
```

```

Question 1 (40 points)

Problem 1

Write a Python script that prints the following figure using **only one line of code**! (so don't use triple quotes)



In [61]:
```
print('\t  | | | \n \t   @ @  \n \t    *  \n          |"""| ')
```

```
          | | |
           @ @
            *
          |"""|
```

Problem 2

Declare a delete_keys function that accepts two arguments: a dictionary and a list of strings. For each string in the list, if the string exists as a dictionary key, delete the key-value pair from the dictionary. If the string does not exist as a dictionary key, avoid an error. The return value should be the modified dictionary object.

EXAMPLE: my_dict = { "A": 1, "B": 2, "C": 3 }

strings = ["A", "C"]

delete_keys(my_dict, strings) => {'B': 2}

In [21]:
```
my_dict = { "A": 1, "B": 2, "C": 3 }
strings = ["A", "C"]
```

In [35]:
```
def delete_keys(mydict,mystring):
    for s in mystring:
        if s in mydict.keys():
            del mydict[s]
    return mydict
```

In [33]:
```
delete_keys(my_dict,strings)
```

In [ ]:
```

```

Problem 3

Define a function called "clean_string" that accepts a single string argument. The function should clean up the white spaces on both sides of the argument. It should also replace every ocurrence of the letter "g" with the letter "Z" and every ocurrence of a space with an exclamation point "!".

For example:

clean_string(" gordon is in Oslo ") => "zordon!is!Oslo"

In [40]:
```
string = "         gordon is in Oslo     "
```

Out[40]:
```
'         gordon is in Oslo     '
```

In [54]:
```
def clean_string(mystring):
    return mystring.strip().replace(" ","!").replace("g","Z")
```

In [53]:
```
clean_string(string)
```

Out[53]:
```
'Zordon!is!in!Oslo'
```

In [ ]:
```

```

Problem 4

Count how many of each vowel (a,e,i,o,u) there are in the text string in the next cell, and use the appropriate string method to print the count for each vowel with a single formatted string. Remember that vowels can be both lower and uppercase.

In [4]:
```
text = """But I must explain to you how all this mistaken idea of d
enouncing pleasure and praising pain was born
and I will give you a complete account of the system, and expound t
he actual teachings of the great explorer of the
truth, the master-builder of human happiness. No one rejects, disli
kes, or avoids pleasure itself, because it is
pleasure, but because those who do not know how to pursue pleasure
rationally encounter consequences that are
extremely painful. Nor again is there anyone who loves or pursues o
r desires to obtain pain of itself, because it is
pain, but because occasionally circumstances occur in which toil an
d pain can procure him some great pleasure.
To take a trivial example, which of us ever undertakes laborious ph
ysical exercise, except to obtain some advantage
from it? But who has any right to find fault with a man who chooses
to enjoy a pleasure that has no annoying
consequences, or one who avoids a pain that produces no resultant p
leasure? On the other hand, we denounce with
righteous indignation and dislike men who are so beguiled and demor
alized by the charms of pleasure of the moment,
so blinded by desire, that they cannot foresee the pain and trouble
that are bound to ensue; and equal blame belongs
to those who fail in their duty through weakness of will, which is
the same as saying through shrinking from toil
and pain."""
```

In [12]:
```
ltext = list(text)
ltext_lower = [x.lower() for x in ltext]

for i in list('aeiou'):
    print("Number of a's:",ltext_lower.count(i))
```

```
Number of a's: 98
Number of a's: 132
Number of a's: 76
Number of a's: 94
Number of a's: 53
```

In [9]:
```
list('aeiou')
```

Out[9]:
```
['a', 'e', 'i', 'o', 'u']
```

Problem 5

Declare a "FindIndex_SameValues" function that accepts two lists. The function should return a list of the index positions in which the two lists have equal elements

EXAMPLES:

FindIndex_SameValues([1, 2, 3], [3, 2, 1]) => [1]

FindIndex_SameValues(["a", "b", "c", "d"], ["c", "b", "a", "d"]) => [1, 3]

In [13]:
```
def FindIndex_SameValues(list1,list2):

    shortest_length = min(len(list1), len(list2))

    list3=[]
    for i in list(range(shortest_length)):
        if list1[i]==list2[i]:
            list3.append(i)
    print(list3)

FindIndex_SameValues([11, 11, 12, 12], [11, 12, 12, 14])
```

```
[0, 2]
```

Problem 6

Simulate R realization of a unbiased coin. Assume p=0.5 and let x be a draw from uniform distribution. If x<p, then it is tail and if x>=p it is head.

a. Create a function flip_Rtimes(R) that flip a fair coin R times. Use R=10.

b. Count the number of times that head occurs k or more times consecutively within the sequence at least once.

In [5]:
```
import random

p = 0.5
```

In [6]:
```
#(a)
def flip(p):
    return 'H' if random.random() < p else 'T'

def flip_Rtimes(R):
    flips = [flip(p) for i in range(R)]
    return flips
```

In [13]:
```
R = 20
flips = flip_Rtimes(R)
#flips
```

In [10]:
```
#(b) Count the number of times that head occurs k or more
#    times consecutively within the sequence at least once.

def Count_more_than_k(flips,k):
    H_occur = 0
    num_H = 0
    for j in flips:
        if j == 'H':
            num_H +=1
            if num_H==k:
                H_occur +=1
        elif j == 'T':
                num_H = 0
    #print(heads)
    return H_occur
```

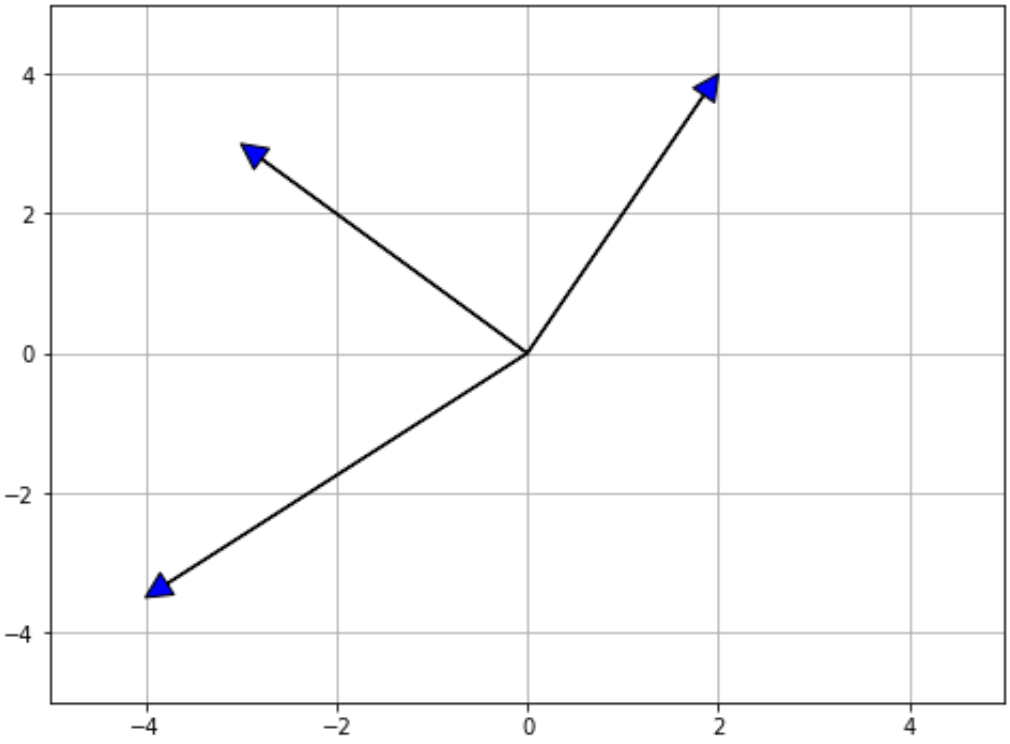
In [11]:
```
num_consec  = Count_more_than_k(flips,2)
num_consec
```

Out[11]: 5

Problem 7

Produce the following figure. (Hint: Use annonate())

In [11]:
```
fig, ax = plt.subplots(figsize=(8, 6))

ax.set(xlim=(-5, 5), ylim=(-5, 5))
ax.grid()
vecs = ((2, 4), (-3, 3), (-4, -3.5))
for v in vecs:
    ax.annotate('', xy=v ,xytext=(0, 0),arrowprops=dict(facecolor='
blue',width=0.5))
plt.show()
```



Problem

Given values $\alpha_1 = 1.5$, and $\alpha_2 = -0.9$, construct the following matrix A using a for loop. The matrix should be for any arbitrary dimensions (NxK) of a matrix. You can test your code with N=10, K=15.

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -\alpha_1 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -\alpha_2 & -\alpha_1 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -\alpha_2 & -\alpha_1 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -\alpha_2 & -\alpha_1 & 1 \end{bmatrix}}_{\equiv A}$$

In [12]:
```
α1 = 1.53
α2 = -.9
T = 8

A = np.zeros((T, T))

for i in range(T):
    A[i, i] = 1

    if i-1 >= 0:
        A[i, i-1] = -α1

    if i-2 >= 0:
        A[i, i-2] = -α2

A
```

Out[12]:
```
array([[ 1.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
       [-1.53,  1.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
       [ 0.9 , -1.53,  1.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  0.9 , -1.53,  1.  ,  0.  ,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  0.9 , -1.53,  1.  ,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  0.  ,  0.9 , -1.53,  1.  ,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  0.  ,  0.  ,  0.9 , -1.53,  1.  ,  0.  ],
       [ 0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.9 , -1.53,  1.  ]])
```

Question 2 (15 points)

Read the data `italy_earthquakes.csv` . The data contains information about earthquakes records from Italy.

1. Load the data and transform the column Magnitude to a float.
2. Plot the mean magnitude every two days.
3. Consider earthquakes greater than 3.0. Computer the rolling mean every 25 days.
4. Consider values of "Depth/Km" between 0 and 20. Plot an histogram with those values.
5. Check if earthquakes are more likely to happen at a specific hour of the day. Compute the mean value by hour of earthquakes larger than 4.0 in magnitud.

In [15]:
```
# (1) Load and clean data
eq = pd.read_csv("./data/italy_earthquakes.csv")
eq.rename(columns={"Magnitude\\":'Magnitude'},inplace=True)
eq.head(3)
```

Out[15]:

	Time	Latitude	Longitude	Depth/Km	Magnitude
0	2016-08-24 03:36:32.000	42.6983	13.2335	8.1	6.0\
1	2016-08-24 03:37:26.580	42.7123	13.2533	9.0	4.5\
2	2016-08-24 03:40:46.590	42.7647	13.1723	9.7	3.8\

In [16]:
```
eq_mag = eq['Magnitude'].str.replace("\\", "")
eq_mag = eq_mag.str.replace("}", "")
eq['Magnitude'] = eq_mag.astype(float)
```

In [17]:
```
#(2) Set time as the index and resample the data every two days
eq = eq.set_index('Time')
eq.index = pd.to_datetime(eq.index)
```

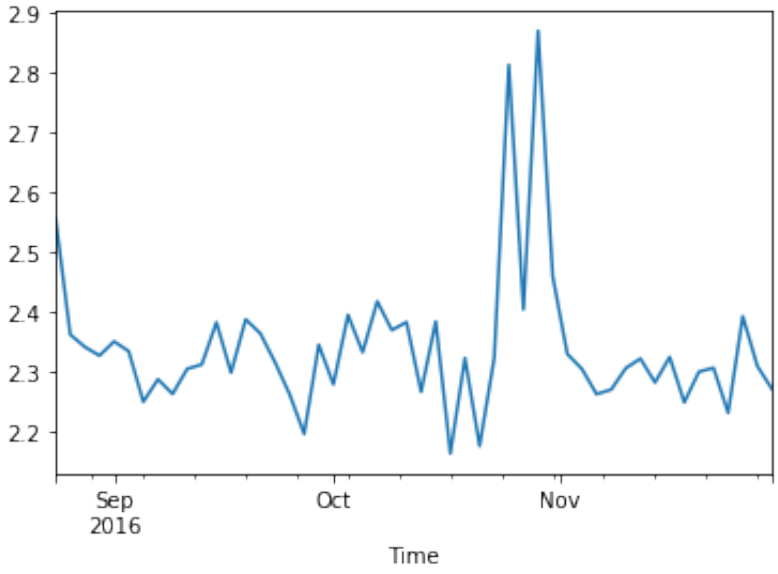
In [18]:
```
eq.head()
```

Out[18]:

Time	Latitude	Longitude	Depth/Km	Magnitude
2016-08-24 03:36:32.000	42.6983	13.2335	8.1	6.0
2016-08-24 03:37:26.580	42.7123	13.2533	9.0	4.5
2016-08-24 03:40:46.590	42.7647	13.1723	9.7	3.8
2016-08-24 03:41:38.900	42.7803	13.1683	9.7	3.9
2016-08-24 03:42:07.170	42.7798	13.1575	9.7	3.6

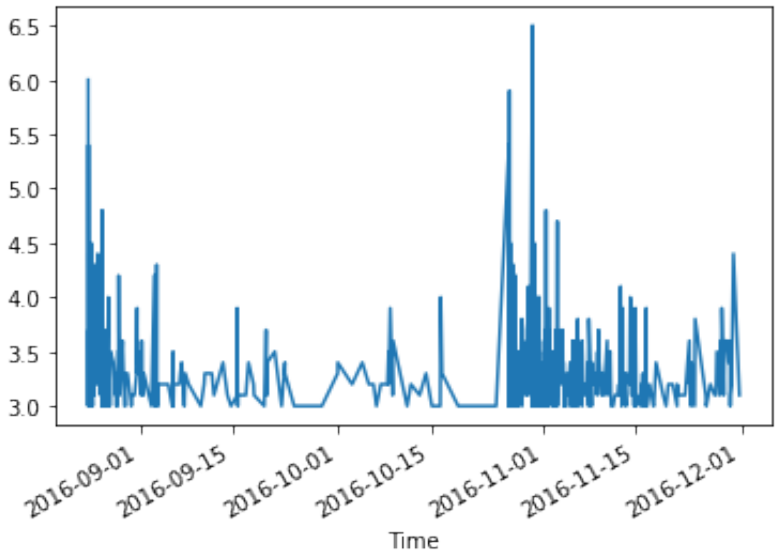
In [19]:
```
eq['Magnitude'].resample('2D').mean().plot()
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe0788f24d0>
```



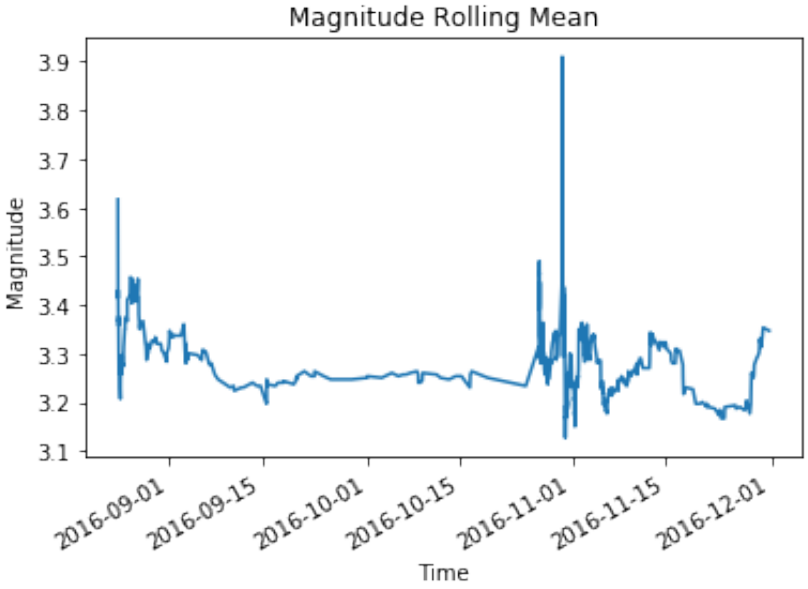
In [20]:
```
# (3) Consider earthquakes greater than 3.0. Computer the rolling m
ean every 25 days.

# Pandas series with magnitudes greater than 3.0
magn3 = eq[eq["Magnitude"] >= 3.0]
magn3['Magnitude'].plot()
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe0a8eec750>
```



In [91]:
```
magn3['Magnitude'].rolling(30).mean().plot()
plt.title("Magnitude Rolling Mean")
plt.ylabel("Magnitude")
```

Out[91]:
```
Text(0, 0.5, 'Magnitude')
```



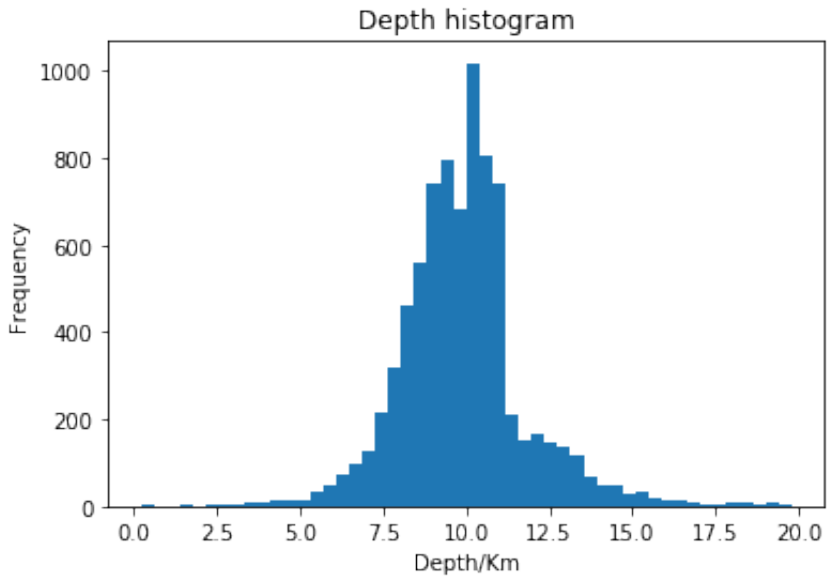
In [94]:
```
eq.head()
```

Out[94]:

Time	Latitude	Longitude	Depth/Km	Magnitude
2016-08-24 03:36:32.000	42.6983	13.2335	8.1	6.0
2016-08-24 03:37:26.580	42.7123	13.2533	9.0	4.5
2016-08-24 03:40:46.590	42.7647	13.1723	9.7	3.8
2016-08-24 03:41:38.900	42.7803	13.1683	9.7	3.9
2016-08-24 03:42:07.170	42.7798	13.1575	9.7	3.6

In [93]:
```
#(4) Consider values of "Depth/Km" between 0 and 20. Plot an histog
ram with those values.

plt.figure()

depth = eq[(eq["Depth/Km"] < 20) & (eq["Depth/Km"] >= 0)]["Depth/Km
"]
depth.plot(kind="hist", stacked=True, bins=50)

plt.title("Depth histogram")
plt.xlabel("Depth/Km")
```

Out[93]:
```
Text(0.5, 0, 'Depth/Km')
```



In [27]:
```
(5)
magn4 = eq[eq["Magnitude"] >= 4.0]['Magnitude']
```

In [28]:
```
dm = magn4.groupby(magn4.index.hour).mean()
dm
```

Out[28]:
```
Time
1     4.000000
2     4.700000
3     4.600000
4     4.700000
5     4.100000
6     4.600000
8     4.542857
9     4.190909
10    4.300000
11    4.000000
12    4.300000
13    4.166667
14    4.450000
15    4.100000
16    4.100000
17    4.200000
18    4.250000
19    4.600000
20    4.000000
21    5.900000
23    4.500000
Name: Magnitude, dtype: float64
```

Question 3 (15 points)

Read the data `operations.csv` . The data contains information about Aerial Bombing Operations in WW2.

1. Load the data. List the number of variables and print the information about types of each variable.
2. Clean some data. Drop countries that are NaN. Drop if target longitude is NaN. Drop if takeoff longitude is NaN.
3. Drop some variables. Drop some
4. Plot the 10 most bombed countries in sample. Make a horizontal bar plot.
5. Compute the number of missions (Mission ID) by year and country

Variables

`Mission Date` : Date of mission. `Theater of Operations` : Region in which active military operations are in progress; "the army was in the field awaiting action"; Example: "he served in the Vietnam theater for three years"

`Country` : Country that makes mission or operation like USA `Air Force` : Name or id of air force unity like 5AF `Aircraft Series` : Model or type of aircraft like B24

`Callsign` : Before bomb attack, message, code, announcement, or tune that is broadcast by radio. `Takeoff Base` : Takeoff airport name like Ponte Olivo Airfield `Takeoff Location` : takeoff region Sicily `Takeoff Latitude` : Latitude of takeoff region `Takeoff Longitude` : Longitude of takeoff region

`Target Country` : Target country like Germany `Target City` : Target city like Berlin `Target Type` : Type of target like city area `Target Industry` : Target industy like town or urban `Target Priority` : Target priority like 1 (most) `Target Latitude` : Latitude of target `Target Longitude` : Longitude of target

In [31]:

In [3]:
```
# 1. Load the data. List the number of variables and print the info
rmation about types of
# each variable.

# bombing data
ae = pd.read_csv("./data/operations.csv",low_memory=False)

# Print number of variables
ae.columns
```

Out[3]:
```
Index(['Mission ID', 'Mission Date', 'Theater of Operations', 'Cou
ntry',
       'Air Force', 'Unit ID', 'Aircraft Series', 'Callsign', 'Mis
sion Type',
       'Takeoff Base', 'Takeoff Location', 'Takeoff Latitude',
       'Takeoff Longitude', 'Target ID', 'Target Country', 'Target
City',
       'Target Type', 'Target Industry', 'Target Priority', 'Targe
t Latitude',
       'Target Longitude', 'Altitude (Hundreds of Feet)', 'Airborn
e Aircraft',
       'Attacking Aircraft', 'Bombing Aircraft', 'Aircraft Returne
d',
       'Aircraft Failed', 'Aircraft Damaged', 'Aircraft Lost',
       'High Explosives', 'High Explosives Type',
       'High Explosives Weight (Pounds)', 'High Explosives Weight
(Tons)',
       'Incendiary Devices', 'Incendiary Devices Type',
       'Incendiary Devices Weight (Pounds)',
       'Incendiary Devices Weight (Tons)', 'Fragmentation Devices'
,
       'Fragmentation Devices Type', 'Fragmentation Devices Weight
(Pounds)',
       'Fragmentation Devices Weight (Tons)', 'Total Weight (Pound
s)',
       'Total Weight (Tons)', 'Time Over Target', 'Bomb Damage Ass
essment',
       'Source ID'],
      dtype='object')
```

In [4]:
```
# print type
# ae.info()
```

In [5]:
```
#2. Clean some data. Drop countries that are NaN. Drop if target lo
ngitude is NaN.
# Drop if takeoff longitude is NaN.

# drop countries that are NaN
ae = ae[pd.isna(ae.Country)==False]
# drop if target longitude is NaN
ae = ae[pd.isna(ae['Target Longitude'])==False]
# Drop if takeoff longitude is NaN
ae = ae[pd.isna(ae['Takeoff Longitude'])==False]
```

In [6]:
```
drop_list = ['Mission ID','Unit ID','Target ID','Altitude (Hundreds
of Feet)','Airborne Aircraft',
             'Attacking Aircraft', 'Bombing Aircraft', 'Aircraft Re
turned',
             'Aircraft Failed', 'Aircraft Damaged', 'Aircraft Lost'
,
             'High Explosives', 'High Explosives Type','Mission Typ
e',
             'High Explosives Weight (Pounds)', 'High Explosives We
ight (Tons)',
             'Incendiary Devices', 'Incendiary Devices Type',
             'Incendiary Devices Weight (Pounds)',
             'Incendiary Devices Weight (Tons)', 'Fragmentation Dev
ices',
             'Fragmentation Devices Type', 'Fragmentation Devices W
eight (Pounds)',
             'Fragmentation Devices Weight (Tons)', 'Total Weight (
Pounds)',
             'Total Weight (Tons)', 'Time Over Target', 'Bomb Damag
e Assessment','Source ID']
ae.drop(drop_list, axis=1,inplace = True)
ae.head(3)
```

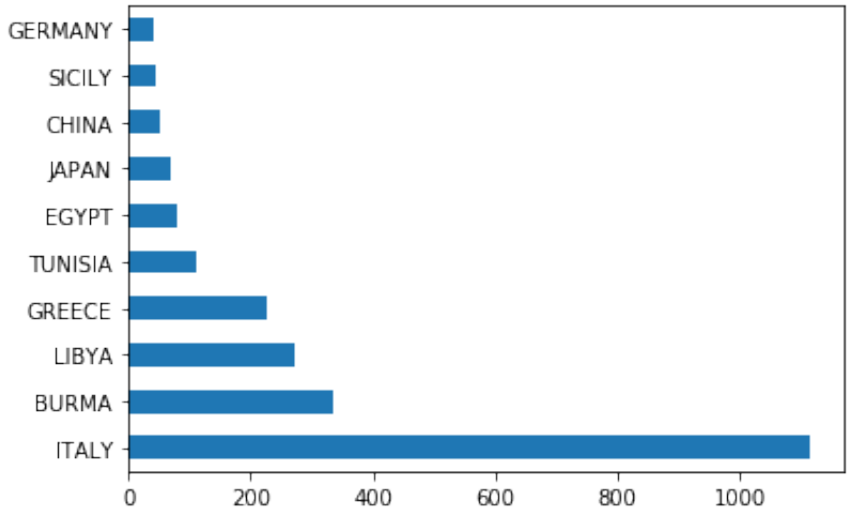
Out[6]:

	Mission Date	Theater of Operations	Country	Air Force	Aircraft Series	Callsign	Takeoff Base	Takeoff Location	Takeoff Latitude
0	8/15/1943	MTO	USA	12 AF	A36	NaN	PONTE OLIVO AIRFIELD	SICILY	37.131022
2	8/15/1943	MTO	USA	12 AF	A36	NaN	PONTE OLIVO AIRFIELD	SICILY	37.131022
3	8/15/1943	MTO	USA	12 AF	A36	NaN	PONTE OLIVO AIRFIELD	SICILY	37.131022

In [7]:
```
# (3). Plot the 10 most bombed countries in sample. Make an horizon
tal bar plot.

# Top target countries
top10_cty = ae['Target Country'].value_counts()[:10]
print(top10_cty)
top10_cty.plot(kind='barh', stacked=True)
```

```
ITALY      1114
BURMA       335
LIBYA       272
GREECE      228
TUNISIA     113
EGYPT        80
JAPAN        71
CHINA        52
SICILY       46
GERMANY      42
Name: Target Country, dtype: int64
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc090c4b550>



In [46]:
```
# (4) Compute the number of missions (Mission ID) by year and count
ry
ae['MissionDate']=pd.to_datetime(ae['Mission Date'])
ae.groupby(['Country',ae['MissionDate'].dt.year])['Mission ID'].cou
nt()
```

Out[46]:
```
Country         MissionDate
AUSTRALIA       1941               4
                1942             312
GREAT BRITAIN   1939              41
                1940            6700
                1941           10216
                1942            4139
                1943            2705
                1944            5205
                1945            2355
NEW ZEALAND     1943               9
                1944             314
                1945             310
SOUTH AFRICA    1940              10
                1941               9
USA             1940              32
                1941              45
                1942            1369
                1943           14911
                1944           46040
                1945           31768
Name: Mission ID, dtype: int64
```

In [ ]:

In [ ]:

Question 4 (30 points)

Read the data `SCFP2016.csv` . The data contains information from the Survey of Consumer Finance in the US. More details in here : https://www.federalreserve.gov/econres/scf_2016.htm (https://www.federalreserve.gov/econres/scf_2016.htm).

The codebook for the survey can be found here.
https://sda.berkeley.edu/data/scfcomb2013/Doc/hcbk.htm
(https://sda.berkeley.edu/data/scfcomb2013/Doc/hcbk.htm).

1. Compute the following summary statistics. Net worth in `NETWORTH` . Keep only data with positive networth.

1. Divide the data in buckets by 'AGE'. Compute a boxplot with information of networth by age category.

1. Compute the fraction of risky assets by individual as
$$FracRisky = \frac{risky - assets}{assets}$$
where $risky - assets$ is the sum of `BOND` , `EQUITY` and `OTHFIN` . Total assets is `ASSET` . Compute the mean proportion of risky asset held by income percentile group. To this end, divide the income into 5 income percentile. Then, compute the mean proportion by category.

1. Suppose you want to predict the fraction of risky assets that each invidual will hold. The data scientist propose the following regression model
$FracRisky_i = \beta_0 + \beta_1 age_i + \beta_2 gen_i + \beta_3 hou_i + \beta_4 mar_i + \beta_5 emp_i + \beta_6 inc_i + \beta_7 netw_i + \epsilon_i$
where the variable age (`AGE`) is the age of the invidual, gen (`HHSE`) is the gender, hou (`HOUSECL`) is whether the indivual has a house, mar (`MARRIED`) is the marital status, emp (`LF`) is whether the indivual is employed, inc is the income percentile group, netw is the net worth percentile group and ϵ_i is the error term.
2. Compare the parameters when you estimate the same regression model above using the machine learning procedure Lasso (Hint: Use package sklearn)

In [9]:
```
scf = pd.read_csv( "data/SCFP2016.csv" )
scf
```

Out[9]:

	YY1	Y1	WGT	HHSEX	AGE	AGECL	EDUC	EDCL	MARRIED	KIDS	.
0	1	11	6427.136675	2	71	5	10	3	2	0	.
1	1	12	6428.350592	2	71	5	10	3	2	0	.
2	1	13	6414.477294	2	71	5	10	3	2	0	.
3	1	14	6428.487972	2	71	5	10	3	2	0	.
4	1	15	6425.256822	2	71	5	10	3	2	0	.
...	...	...	...	...	...	...	...	...	...	...	.
31235	6261	62611	4461.752118	1	29	1	9	3	1	1	.
31236	6261	62612	4482.258683	1	29	1	9	3	1	1	.
31237	6261	62613	4469.827487	1	29	1	9	3	1	1	.
31238	6261	62614	4459.960505	1	29	1	9	3	1	1	.
31239	6261	62615	4479.387365	1	29	1	9	3	1	1	.

31240 rows × 351 columns

Part (1)

Compute the following summary statistics. Net worth in `NETWORTH`. Keep only data with positive networth.

In [10]:
```
# (1) Summary table
networth = scf['NETWORTH']
print(" How many survey participants                  : ", len(scf))
print(" Mean net worth in the sample                  : ", int(np.mean(networth) ))
print(" Median net worth in the sample                : ", int(np.median(networth) ))
print(" Net worth on the top 1% of households         : ", int(networth.quantile(q=.99)))
print(" Net worth on the top .1% of households        : ", int(networth.quantile(q=.999)))
print(" Fraction of households with negative wealth   : ",round((scf['NETWORTH']>0).sum()/len(scf),4) )

# Keep household with positive networth
scf = scf[scf['NETWORTH']>0]
```

```
 How many survey participants                  :  31240
 Mean net worth in the sample                  :  12682385
 Median net worth in the sample                :  200536
 Net worth on the top 1% of households         :  277306447
 Net worth on the top .1% of households        :  1134510792
 Fraction of households with negative wealth   :  0.9
```

Part (2)

Divide the data in buckets by 'AGE'. Compute a boxplot (as in the figure below) with information of networth by age category.

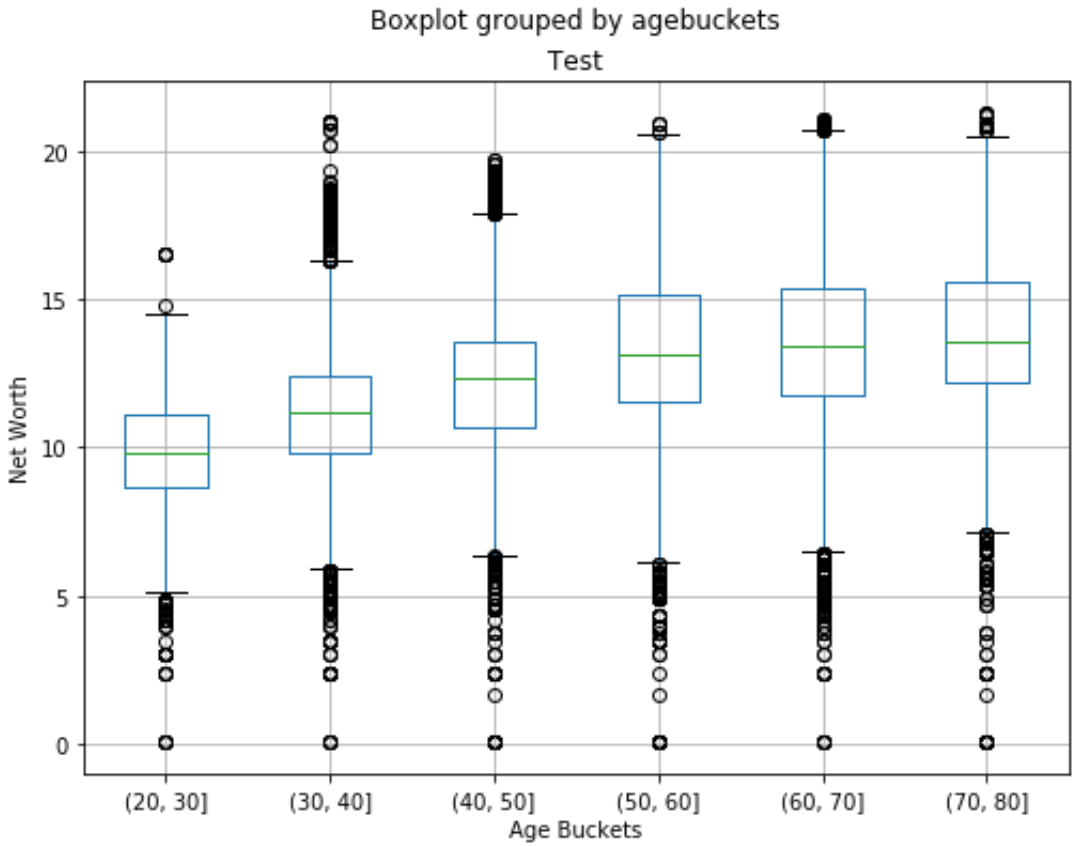
In [14]:
```
boxage = scf[['NETWORTH','AGE']].copy()
age_cuts = [ 20, 30, 40, 50, 60,70,80]
boxage['agebuckets'] = pd.cut(scf.AGE, age_cuts)
boxage['log_netw'] = np.log(scf['NETWORTH'])

boxage.head(3)
```

Out[14]:

	NETWORTH	AGE	agebuckets	log_netw
0	187954.52039	71	(70, 80]	12.143955
1	188071.51336	71	(70, 80]	12.144578
2	187965.15612	71	(70, 80]	12.144012

In [15]:
```
boxage.boxplot(column='log_netw', by='agebuckets',figsize=(8,6))
plt.title('Test')
plt.xlabel('Age Buckets')
plt.ylabel('Net Worth')
```

Out[15]:
```
Text(0, 0.5, 'Net Worth')
```



(3) Risky Assets

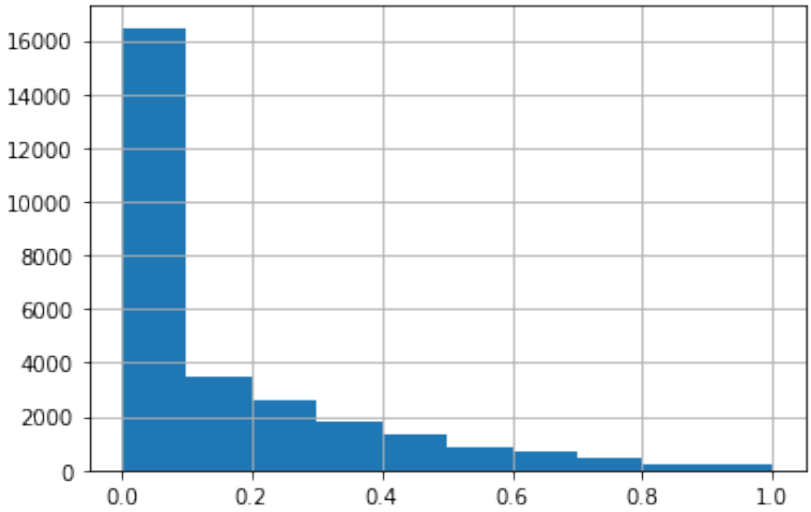
Compute the fraction of risky assets by individual as

$$FracRisky = \frac{risky - assets}{assets}$$

where $risky - assets$ is the sum of `BOND`, `EQUITY` and `OTHFIN`. Total assets is `ASSET`. Compute the mean proportion of risky asset held by income percentile group. To this end, divide the income into 5 income percentile. Then, compute the mean proportion by category.

In [16]:
```
ra = scf[['ASSET','INCOME','BOND','EQUITY','OTHFIN']]/1000
risky_asset = (ra['BOND']+ra['EQUITY']+ra['OTHFIN'])/ra['ASSET']

risky_asset.hist()
```

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc060b4a3d0>
```



In [17]:
```
ra.INCOME.describe()
```

Out[17]:
```
count     28117.000000
mean        940.393098
std        6118.306537
min           0.000000
25%          37.705152
50%          81.874044
75%         196.066789
max      325298.503020
Name: INCOME, dtype: float64
```

In [18]:
```
bin_labels_5 = ['0-20','20-40','40-60','60-80','80-100']
income_qcut = pd.qcut(ra.INCOME, q=[0, .2, .4, .6, .8, 1],labels=bi
n_labels_5)
```

In [19]:
```
risky_asset.groupby(income_qcut).apply(np.mean)
```

Out[19]:
```
INCOME
0-20       0.059317
20-40      0.100274
40-60      0.145071
60-80      0.186379
80-100     0.276649
dtype: float64
```

(4) Predicting the fraction of risky assets

Suppose you want to predict the fraction of risky assets that each invidual/household will hold. The data scientist propose the following regression model

$$FracRisky_i = \beta_0 + \beta_1 age_i + \beta_2 gen_i + \beta_3 hou_i + \beta_4 mar_i + \beta_5 emp_i + \beta_6 inc_i + \beta_7 netw_i + \epsilon_i$$

where the variable age (`AGE`) is the age of the invidual, gen (`HHSEX`) is the gender of the household, hou (`HOUSECL`) is whether the indivual has a house, mar (`MARRIED`) is the marital status, emp (`LF`) is whether the indivual is employed, inc is the income percentile group, netw is the net worth percentile group and ϵ_i is the error term.

In [145]:
```
inc_dummies = pd.get_dummies(income_qcut)
inc_dummies.head()
```

Out[145]:

	0-20	20-40	40-60	60-80	80-100
0	1	0	0	0	0
1	1	0	0	0	0
2	1	0	0	0	0
3	1	0	0	0	0
4	1	0	0	0	0

In [146]:
```
X = scf[['AGE','HDEBT','HOUSECL','MARRIED','LF']]
X = pd.concat([X,inc_dummies],axis=1)
y = risky_asset
X
```

Out[146]:

	AGE	HDEBT	HOUSECL	MARRIED	LF	0-20	20-40	40-60	60-80	80-100
0	71	1	1	2	0	1	0	0	0	0
1	71	1	1	2	0	1	0	0	0	0
2	71	1	1	2	0	1	0	0	0	0
3	71	1	1	2	0	1	0	0	0	0
4	71	1	1	2	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
31235	29	1	2	1	1	1	0	0	0	0
31236	29	1	2	1	1	1	0	0	0	0
31237	29	1	2	1	1	1	0	0	0	0
31238	29	1	2	1	1	1	0	0	0	0
31239	29	1	2	1	1	1	0	0	0	0

28117 rows × 10 columns

In [26]:
```
#regression
y = risky_asset
X = scf[['AGE','HDEBT','HOUSECL','MARRIED','LF']]
X = pd.concat([X,inc_dummies],axis=1)

reg1 = sm.OLS(y, sm.add_constant(X), missing = "drop")
results1 = reg1.fit();
print(results1.summary()) #Regression-results gets printed below
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:
0.155
Model:                            OLS   Adj. R-squared:
0.155
Method:                 Least Squares   F-statistic:
572.5
Date:                Tue, 03 Nov 2020   Prob (F-statistic):
0.00
Time:                        18:40:09   Log-Likelihood:
6455.2
No. Observations:               28117   AIC:
-1.289e+04
Df Residuals:                   28107   BIC:
-1.281e+04
Df Model:                           9
Covariance Type:            nonrobust
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0295      0.008      3.632      0.000       0.014       0.045
AGE            0.0009   9.54e-05      9.290      0.000       0.001       0.001
HDEBT         -0.0269      0.003     -9.571      0.000      -0.032      -0.021
HOUSECL        0.0491      0.003     15.653      0.000       0.043       0.055
MARRIED        0.0324      0.003     11.844      0.000       0.027       0.038
LF            -0.0229      0.003     -7.001      0.000      -0.029      -0.017
0-20          -0.1241      0.003    -37.655      0.000      -0.131      -0.118
20-40         -0.0563      0.003    -18.951      0.000      -0.062      -0.051
40-60          0.0076      0.003      2.729      0.006       0.002       0.013
60-80          0.0596      0.003     21.296      0.000       0.054       0.065
80-100         0.1428      0.003     48.137      0.000       0.137       0.149
==============================================================================
Omnibus:                     8030.695   Durbin-Watson:                   0.529
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            20537.216
Skew:                           1.568   Prob(JB):                         0.00
Kurtosis:                       5.775   Cond. No.                     2.89e+17
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.08e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [ ]:

Part (5)

1. Compare the parameters when you estimate the same regression model above using the machine learning procedure Lasso (Hint: Use package sklearn). Show in a table the estimates of the linear regression model (from 4) and the estimates from Lasso.

In [48]:
```
# import
from sklearn import linear_model
from sklearn.linear_model import Lasso

# construct the model instance
reg1 = linear_model.LinearRegression()

# fit the model
reg1.fit(X, y)

# Lasso Model: This may not be the best Lasso model (we need anythi
ng that works here)
reg2 = Lasso(alpha=0.01)
reg2.fit(X, y,)
```

Out[48]:
```
Lasso(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_st
ate=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

In [49]:
```
beta_s = reg1.coef_
beta_s
```

Out[49]:
```
array([ 0.0008865 , -0.02690537,  0.04905312,  0.03243763, -0.0229
174 ,
       -0.12996924, -0.06225412,  0.00167048,  0.05368975,  0.1368
6313])
```

In [47]:
```
beta_lasso = reg2.coef_
beta_lasso
```

Out[47]:
```
array([ 0.00129675, -0.        ,  0.        , -0.        ,  0.
,
       -0.03608794, -0.        ,  0.        ,  0.        ,  0.0731
5863])
```

In [54]:
```
# Simple table
data = {'regression': beta_s, 'lasso': beta_lasso}
df = pd.DataFrame(data=data)
df
```

Out[54]:

	regression	lasso
0	0.000887	0.001297
1	-0.026905	-0.000000
2	0.049053	0.000000
3	0.032438	-0.000000
4	-0.022917	0.000000
5	-0.129969	-0.036088
6	-0.062254	-0.000000
7	0.001670	0.000000
8	0.053690	0.000000
9	0.136863	0.073159

In [ ]: