Lorenzo Rega

Lr3103

HOMEWORK 4

Exercise 1

First, we have to read the data:

```
import pandas as pd
df_school=pd.read_excel("homework 4.xlsx")
df_school=df_school.set_index("ID")

df_school.head()
```

ID	2010 ST	2011 ST	Treated	2012 ST
1	13	11	1	15
2	16	21	0	20
3	11	13	0	21
4	14	17	0	14
5	8	7	1	22

a)

Let's compute the Did:

Our test group is defined by all the students who did not receive SIS whereas our control group is defined by all the students who did not receive SIS.

```
test_group=df_school[df_school['Treated']==1]
```

```
control_group=df_school[df_school['Treated']==0]
```

We then compute the difference in the test group and in the control group

```
import numpy as np
#Let's compute the difference in the test group

before_test_average=np.mean(test_group['2011 ST'])
after_test_average=np.mean(test_group['2012 ST'])
diff_test=after_test_average-before_test_average
perc_diff_test=(diff_test/before_test_average)*100
print(f"{round(perc_diff_test,2)} %")
```

```
45.07 %
```

```
#Let's compute the difference in the control group
before_control_average=np.mean(control_group['2011 ST'])
after_control_average=np.mean(control_group['2012 ST'])
diff_control=after_control_average-before_control_average
perc_diff_control=(diff_control/before_control_average)*100
print(f"{round(perc_diff_control,2)} %")
```

```
-3.35 %
```

Finally, we can compute our DID:

```
did=perc_diff_test - perc_diff_control
print(f"did= {round(did,2)} %")
```

```
did= 48.42 %
```

This means that the increase in test scores from SIS is 48.42%.

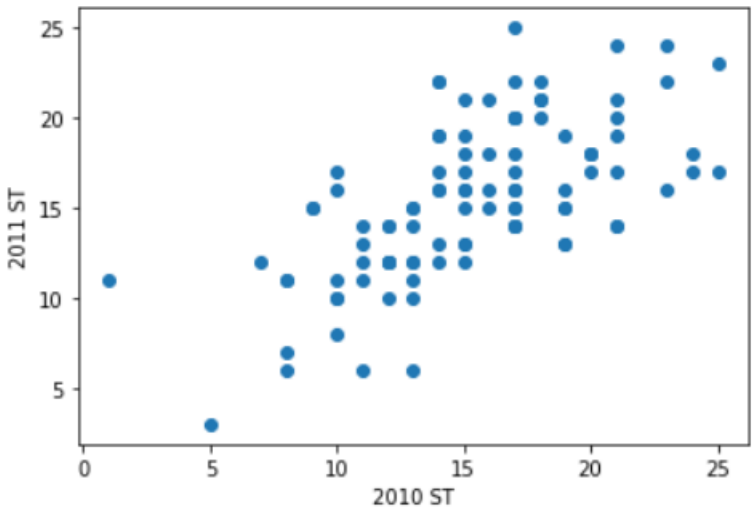
b)

We first Construct a scatter plot of 2011 vs 2010 test scores

```
import matplotlib.pyplot as plt
```

```
x=df_school['2010 ST']
y=df_school['2011 ST']


plt.xlabel("2010 ST")
plt.ylabel("2011 ST")
plt.scatter(x,y)
plt.show()
```



We then perform a linear regression on this data: our dependent variable is going to be the 2011 ST results, whereas the 2010 ST results is the independent variable

```
import statsmodels.formula.api as smf
```

```
linear_reg= smf.ols('Q("2011 ST") ~ Q("2010 ST")', data = df_school).fit()
linear_reg.summary()
```

OLS Regression Results

Dep. Variable:	Q("2011 ST")	**R-squared:**	0.412
Model:	OLS	**Adj. R-squared:**	0.406
Method:	Least Squares	**F-statistic:**	68.58
Date:	Thu, 01 Dec 2022	**Prob (F-statistic):**	6.34e-13
Time:	16:15:46	**Log-Likelihood:**	-261.54
No. Observations:	100	**AIC:**	527.1
Df Residuals:	98	**BIC:**	532.3
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	6.1336	1.172	5.232	0.000	3.807	8.460
Q("2010 ST")	0.6107	0.074	8.281	0.000	0.464	0.757

Omnibus:	0.895	**Durbin-Watson:**	1.809
Prob(Omnibus):	0.639	**Jarque-Bera (JB):**	1.003
Skew:	0.171	**Prob(JB):**	0.606
Kurtosis:	2.647	**Cond. No.**	56.0

The coefficient of this line is:

```
#shrinkage estimator c = linear regression coefficient
c=linear_reg.params[1]
c
```

```
0.6106589298231948
```

Since it is <1, we can assume that there is regression towards the mean. This means the test is not purely skill but has a luck component as well.

c)

Our linear regression coefficient can be used as our shrinkage estimator.

Let’s construct a shrinkage estimate of 2012 scores based on the 2011 test results:

```
#forumla for 2012 test score
prediction_2012=c*df_school['2011 ST']+(1-c)*np.mean(df_school['2011 ST'])

#we add this as a new column of the dataframe
df_school['2012 prediction']=prediction_2012
df_school
```

ID	2010 ST	2011 ST	Treated	2012 ST	2012 prediction
1	13	11	1	15	12.728674
2	16	21	0	20	18.835264
3	11	13	0	21	13.949992
4	14	17	0	14	16.392628
5	8	7	1	22	10.286039
...	...	...	...	...	...
96	16	16	0	13	15.781969
97	17	20	0	20	18.224605
98	12	12	0	13	13.339333
99	14	13	0	18	13.949992
100	15	18	0	19	17.003287

```
#let's compute the RMSE
import math

se= ((df_school['2012 ST'] - df_school['2012 prediction'])**2)
mse=np.sum(se)/ len(df_school)
rmse=math.sqrt(mse)
rmse
```

```
4.221955437290859
```

d)

New Did analysis

```
test_group=df_school[df_school['Treated']==1]
control_group=df_school[df_school['Treated']==0]

test_average_2012=np.mean(test_group['2012 ST'])
test_average_2012_pred=np.mean(test_group['2012 prediction'])

control_average_2012=np.mean(control_group['2012 ST'])
control_average_2012_pred=np.mean(control_group['2012 prediction'])
```

before scores=estimated scores , after scores= actual scores

```
diff_test=test_average_2012 - test_average_2012_pred
perc_diff_test=(diff_test*100)/test_average_2012_pred
print(f"diff test: {round(perc_diff_test,2)} %")
```

```
diff test: 12.63 %
```

```
diff_control=control_average_2012 - control_average_2012_pred
perc_diff_control=(diff_control*100)/control_average_2012_pred
print(f"diff control: {round(perc_diff_control,2)} %")
```

```
diff control: -0.45 %
```

```
did=perc_diff_test-perc_diff_control
print(f"Did: {round(did,2)} %")
```

```
Did: 13.08 %
```

e)

In the first DiD analysis we are not considering that this test is not purely a “skill game”.

This made the impact of SIS look even bigger than it actually was. The second DID analysis removes this bias and tries to show the real impact of SIS on students.

Exercise 2

a and b)

Simulation model

```
import numpy as np
```

```
def sim_one_night(p, n_rooms):
    #each room can be either 0 or 1, depending if people didn't or did show up
    rooms=np.random.binomial(n=1, p=p, size=n_rooms)
    n_costumers=len(rooms)-sum(rooms)
    return n_costumers
```

```
def sim_n_nights(n):
    np.random.seed(123)
    costumers=[]
    #let's see each night how many costumers actually show up
    for n in range(1000):
        costumers.append(sim_one_night(p=0.16, n_rooms=20))

    #we take the mean number of costumers who did show up
    mean=np.mean(costumers)
    return mean
```

```
sim_n_nights(1)
```

```
16.852
```

```
sim_n_nights(1000)
```

```
16.852
```

Probability the hotel is full:

```
x=[]
np.random.seed(123)
for n in range(1000):
    x.append(1 if sim_one_night(p=0.16, n_rooms=20)==20 else 0)
```

```
np.mean(x)
```

```
0.035
```

c)

Accepting 21 reservations, we have 0.03 walked costumers on average and the average occupancy is 17.685 customers each night.

Let's now accept 21 reservations (we still have only 20 rooms)

```
np.random.seed(123)
walked=[]
occupancy=[]
for n in range(1000):
    occupied_rooms=sum(np.random.binomial(n=1, p=0.84, size=21))
    #print(occupied_rooms)
    if occupied_rooms>20:
        walked.append(occupied_rooms-20)
        occupancy.append(occupied_rooms)
    else:
        walked.append(0)
        occupancy.append(occupied_rooms)
(np.mean(walked), np.mean(occupancy))
```

```
(0.03, 17.685)
```

d)

Let's now overbook by 2,3 and 4

```
np.random.seed(123)
for more in range(1,5):
    walked=[]
    occupancy=[]
    for n in range(1000):
        occupied_rooms=sum(np.random.binomial(n=1, p=0.84, size=(20+more)))
        if occupied_rooms>20:
            walked.append(occupied_rooms-20)
            occupancy.append(occupied_rooms)
        else:
            walked.append(0)
            occupancy.append(occupied_rooms)
    print((np.mean(walked), np.mean(occupancy)))
```

```
(0.03, 17.685)
(0.139, 18.494)
(0.375, 19.293)
(0.78, 20.173)
```

The number on the left represents the average walked customers whereas the one on the right represents the average occupancy.

e)

I would keep an overbook by 2. I don't want my hotel to become famous for having to walk his costumers. An overbook by 3 could still be acceptable. The thing we have to pay attention to is that the average walked costumers grow exponentially with each overbook increase. We have to be careful handling overbooking if we don't want to make our costumers upset.

Exercise 3

Problem formulation:

Let: xi be the amount of money invested on bond i

i=1,2,3,4,5

max z=0.044x1+0.028x2+0.026x3+0.021x4+0.047 x5

s.t.

x1+x2+x3+x4+x5<=12000000 We can't invest more than our budget of 12M

x3+x4+x5>=4000000 Government and agency bonds must total at least $4 million

xi<=8000000 for every i No more than $8 million can be invested in any one bond

2x1+2x2+x3+x4+5x5<=1.4(x1+x2+x3+x4+x5) The average quality of the portfolio cannot exceed 1.4 on the bank's rating scale

10x1+15x2+5x3+4x4+3x5<=5(x1+x2+x3+x4+x5) The average number of years to maturity of the portfolio cannot exceed 5 years.

xi>=0 for every i

In python:

We first read the data

```
df_investment = pd.read_excel('homework 4.xlsx', sheet_name = 'pb3_data')
df_investment
```

	Name	Type	Rating	Bank Rating	Maturity (Yrs.)	Yield to Maturity	After-Tax Yield
0	A	Municipal	AA	2	10	0.044	0.044
1	B	Agency	AA	2	15	0.055	0.028
2	C	Government	AAA	1	5	0.051	0.026
3	D	Government	AAA	1	4	0.043	0.021
4	E	Municipal	BB	5	3	0.047	0.047

```
import pulp as pl
```

```
#create the pulp lp
m = pl.LpProblem('Portfolio', pl.LpMaximize)
```

```
#create the variables:
x=[]

for i in df_investment.index:
    x.append( pl.LpVariable(f'x_{i}', cat='Continuous') )

x = np.array(x)
```

```
#create the constraints


#We can't invest more than our budget of 12M
m += (pl.lpSum(x) <= 12, 'maximum allocation constraint')

# Government and agency bonds must total at least $4 million
m += (x[1]+x[2]+x[3] >= 4, 'min government and agency constraint')

#  No more than $8 million can be invested in any one bond
for i in df_bonds.index:
    m += (x[i] <= 8, f'bond {i} max constraint')
    m += (x[i] >= 0, f'Positive variable constraint {i}')

# The average quality of the portfolio cannot exceed 1.4 on the bank's rating scale
m += (pl.lpSum(x*np.array(df_bonds['Bank Rating'])) <= 1.4*pl.lpSum(x), 'quality constraint')

# The average number of years to maturity of the portfolio cannot exceed 5 years.
m += (pl.lpSum(x*np.array(df_bonds['Maturity (Yrs.)'])) <= 5*pl.lpSum(x), 'maturity constraint')
```

```
#objective function
m += pl.lpSum(np.array(df_bonds['After-Tax Yield'])*x)
```

```
pl.list_solvers(onlyAvailable=True)
```

```
['PULP_CBC_CMD']
```

```
pl.PULP_CBC_CMD().solve(m)
```

```
1
```

```
# maximum objective function:
optimal_soln = m.objective.value()
optimal_soln
```

```
0.336928
```

```
#optimal x_i values
[x[i].value() for i in range(len(x))]
```

```
[0.832, 0.0, 8.0, 2.176, 0.992]
```

Exercise 4

a)

```
np.random.seed(123)
demands=[]
for n in range(10000):
    #demand in the morning
    d_morning=np.random.normal(loc=50, scale=10)

    s=np.random.binomial(n=1, p=0.4)

    if(s==1): #if it is a sunny day
        d_afternoon=np.random.uniform(60,80)
    else: #if it is a rainy day
        d_afternoon=np.random.uniform(20,50)

    d_total=d_morning+d_afternoon

    demands.append(d_total)
```

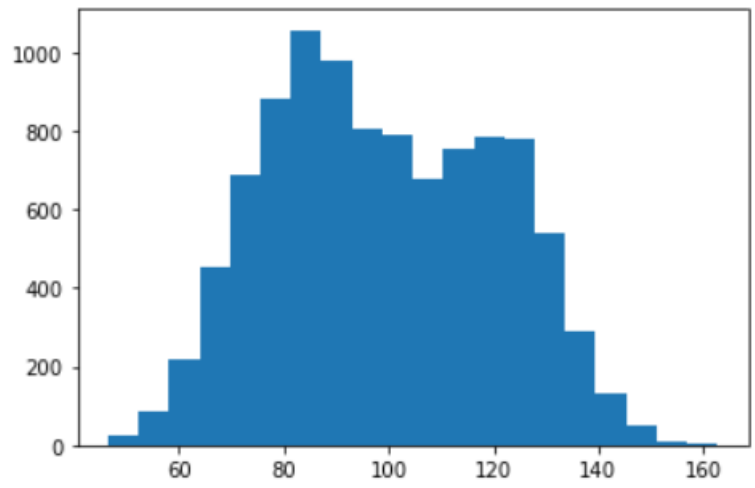
```
# 10th percentile
np.percentile(demands,10)
```

```
71.75845113376431
```

```
#90th percentile
np.percentile(demands,90)
```

```
128.01986291248477
```

```
#Creating histogram
import matplotlib.pyplot as plt
plt.hist(demands, bins = 20);
```



b)

Let's compute the revenue with 120 croissants; unit cost=1 dollar unit revenue=4 dollars

```
cost=1
revenue=4
croissants_made=120

np.random.seed(123)
profits=[]
for n in range(10000):
    #demand in the morning
    d_morning=np.random.normal(loc=50, scale=10)

    s=np.random.binomial(n=1, p=0.4)

    if(s==1): #if it is a sunny day
        d_afternoon=np.random.uniform(60,80)
    else: #if it is a rainy day
        d_afternoon=np.random.uniform(20,50)

    d_total=d_morning+d_afternoon
    if d_total>=croissants_made:
        profit=(revenue-cost)*croissants_made
    else:
        profit=(revenue*d_total)-(cost*croissants_made)

    profits.append(profit)
```

```
#expected profit:
np.mean(profits)
```

```
268.4605265845361
```

c)

To find the optimal number of croissants, we repeat the simulation for different numbers of croissants made, and just see when we have the maximum profit.

```
cost=1
revenue=4
croissants_made=120

np.random.seed(123)


for croissants_made in range(60,180):

    profits=[]

    for n in range(10000):
        #demand in the morning
        d_morning=np.random.normal(loc=50, scale=10)

        s=np.random.binomial(n=1, p=0.4)

        if(s==1): #if it is a sunny day
            d_afternoon=np.random.uniform(60,80)
        else: #if it is a rainy day
            d_afternoon=np.random.uniform(20,50)

        d_total=d_morning+d_afternoon
        if d_total>=croissants_made:
            profit=(revenue-cost)*croissants_made
        else:
            profit=(revenue*d_total)-(cost*croissants_made)

        profits.append(profit)

    expected_profits.append(np.mean(profits))
```

We put the results into a dataframe:

Each row is an expected profit. The index of the row represents the number of croissants made that day.

```
import pandas as pd
data=pd.DataFrame(expected_profits, index=range(60,180), columns=['expected profits'])
```

```
data
```

	expected profits
60	179.724142
61	182.669314
62	185.551010
63	188.379253
64	191.351042
...	...
175	221.778490
176	219.159246
177	217.240763
178	217.032664
179	218.281673

120 rows × 1 columns

We find the index of the maximum expected value to get the number of croissants that should be made every day to maximize the profit.

```
data.idxmax()
```

```
expected profits    118
dtype: int64
```

The number is 118 croissants in a day.

Exercise 5

a)

Let:

Xi number of product i produced

i=0,1,2,3

0->Standard laptop, 1 -> Costumized Laptop, 2-> Standard Desktop, 3-> Customized desktop

Max z= 100x_0 + 200x_1 + 150x_2 + 400x_3

s.t.

x1+x3<=600 only 600 machines can be customized in a month

x0+x1<=1500 The manufacturer makes 1500 laptops per month

x2+x3<=1000 The manufacturer makes 1000 desktops per month

We can't produce more than our monthly demand:

x0<=1200

x1<=1000

x2<=700

x3<=400

xi>=0 for every i

The problem is a linear problem

b)

In python:

```
n = pl.LpProblem('Laptops', pl.LpMaximize)
```

```
# Create variables
x = []
for i in df.index:
    x.append( pl.LpVariable(f'x_{i}', cat='Continuous') )

x = np.array(x)
```

```
# Create constraints

# The manufacturer makes 1500 laptops per month
n += (x[0] + x[1] <= 1500, 'Laptop production constraint')

# The manufacturer makes 1000 desktops per month
n += (x[2] + x[3] <= 1000, 'Desktop production constraint')

# only 600 machines can be customized in a month
n += (x[1] + x[3] <= 600, 'Customization constraint')

#We can't produce more than our monthly demand:
for i in df.index:
    n += (x[i] <= df.Demand[i], f'Comp {i} demand constraint')
```

```
#objective function
n += pl.lpSum(np.array(df['Net Profit'])*x)
```

```
#solve the problem
pl.list_solvers(onlyAvailable=True)
pl.PULP_CBC_CMD().solve(n)
```

```
1
```

```
# Obtain optimal solution
optimal_soln = n.objective.value()
optimal_soln
```

```
410000.0
```

```
#x_i values
[x[i].value() for i in range(len(x))]
```

```
[1200.0, 200.0, 600.0, 400.0]
```

c)

We can change 1 line of code, increasing by 200 the customized machines:

```
n += (x[1] + x[3] <= 800, 'Customization constraint')
```

And get as the maximum profit:

```
# Obtain optimal solution
optimal_soln = n.objective.value()
optimal_soln
```

```
440000.0
```

Which is 30000 $ more than the one computed before.

d)

We change this line of code:

```
n += (x[0] + x[1] <= 1400, 'Laptop production constraint')
```

```
# Obtain optimal solution
optimal_soln = n.objective.value()
optimal_soln
```

```
410000.0
```

We notice that our optimal value of the objective function does not change.