Homework 4

Problem 1:

a) To apply the difference-in-difference analysis to the given Hillside students' data, we begin by finding the average test scores. To do so, we check the 'Treated' column in the data to differentiate between the groups to which treatment (here, SIS) was given or not, and find the averages before and after for each group.

```
df_sis_data=pd.read_excel('homework 4.xlsx', sheet_name = 'pb1_data')
df_sis_data.head(10)
```

	ID	2010 ST	2011 ST	Treated	2012 ST
0	1	13	11	1	15
1	2	16	21	0	20
2	3	11	13	0	21
3	4	14	17	0	14
4	5	8	7	1	22
5	6	15	12	0	16
6	7	15	16	0	17
7	8	17	15	0	12
8	9	10	8	1	8
9	10	8	6	1	7

In [12]:
```
control_before = df_sis_data[df_sis_data.Treated == 0]['2011 ST'].mean()
control_after = df_sis_data[df_sis_data.Treated == 0]['2012 ST'].mean()
treatment_before = df_sis_data[df_sis_data.Treated == 1]['2011 ST'].mean()
treatment_after = df_sis_data[df_sis_data.Treated == 1]['2012 ST'].mean()
conrol_perc_diff = (contol_after-control_before)*100/control_before
treatment_perc_diff = (treatment_after-treatment_before)*100/treatment_before
```

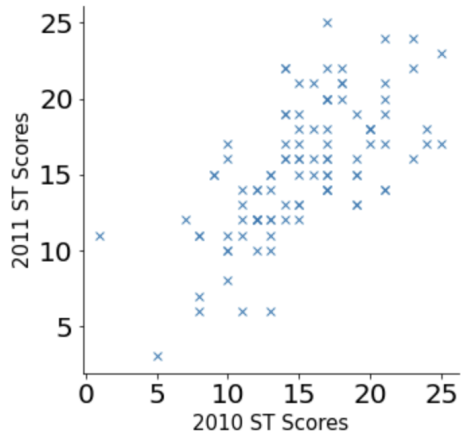
In [18]:
```
print('Control Group')
print(f'Pre-SIS:{round(control_before,2 )}')
print(f'Post-SIS:{round(contol_after,2 )}')
print(f'Difference:{round(control_after-control_before, 2)}')
print(f'% difference: {round(conrol_perc_diff,2)}%')
print('\nTreatment Group')
print(f'Pre-SIS:{round(treatment_before,2 )}')
print(f'Post-SIS:{round(treatment_after,2 )}')
print(f'Difference:{round(treatment_after-treatment_before, 2)}')
print(f'% difference:{round(treatment_perc_diff,2)}%')
print(f'\nDifference in differences:{round(treatment_perc_diff - conrol_perc_diff, 2)}%')
```

```
Control Group
Pre-SIS:16.69
Post-SIS:16.13
Difference:-0.56
% difference: -3.35%

Treatment Group
Pre-SIS:8.88
Post-SIS:12.88
Difference:4.0
% difference:45.07%

Difference in differences:48.42%
```

Based on the DiD estimate, we notice that the scores for control-group students went from 16.69 to 16.13, thus decreasing by 0.56 (3.35%), while scores for students subjected to the SIS program went from 8.88 to 12.88, showing an increase of 4 points(45.07%). The DiD is then 48.42%, which suggests that the SIS program may have been successful.

b) Since the DiD analysis did not account for regression to the mean, we can try to compute a shrinkage coefficient using the data from the years 2010 and 2011 and then recompute this DiD. First, we begin by plotting the 2011 test scores against the 2010 test scores.

In [36]:
```
plt.figure(figsize=(5,5))
plt.plot(df_sis_data['2010 ST'], df_sis_data['2011 ST'], marker='x', linewidth=0)
plt.xlabel('2010 ST Scores', fontsize=15)
plt.ylabel('2011 ST Scores', fontsize=15)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
sns.despine()
```



Only the students who performed poorly on the 2011 exam were enrolled in SIS, some increase in their 2012 scores could be expected simply due to the effect of regression on the mean. To test this hypothesis, we consider the performance of the students between 2010 and 2011. To find the shrinkage coefficient, we will use results from the linear regression and find the slope of the regression line.

In [33]:
```
linear_regression=smf.ols('ST_2011 ~ ST_2010',
                          data = df_sis_data.rename(columns={'2011 ST':'ST_2011',
                                                             '2010 ST':'ST_2010'})).fit()
linear_regression.summary()
```

Out[33]:

OLS Regression Results

Dep. Variable:	ST_2011	**R-squared:**	0.412
Model:	OLS	**Adj. R-squared:**	0.406
Method:	Least Squares	**F-statistic:**	68.58
Date:	Sat, 10 Dec 2022	**Prob (F-statistic):**	6.34e-13
Time:	00:05:06	**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
ST_2010	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

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

The summary gives us the slope of our regression line, which is 0.6107. As this slope is less than 1, we can conclude there is a regression to the mean in test scores.

c)

```
In [39]: c=linear_reg.params[1]
         average_2011=df_sis_data['2011 ST'].mean()

         df_sis_data['2012 Prediction']=c*df_sis_data['2011 ST']+(1-c)*average_2011

         df_sis_data['Predicetd Err']=df_sis_data['2012 Prediction']-df_sis_data['2012 ST']
         df_sis_data.head(10)
```

Out[39]:

	ID	2010 ST	2011 ST	Treated	2012 ST	2012 Prediction	Predicetd Err
0	1	13	11	1	15	12.728674	-2.271326
1	2	16	21	0	20	18.835264	-1.164736
2	3	11	13	0	21	13.949992	-7.050008
3	4	14	17	0	14	16.392628	2.392628
4	5	8	7	1	22	10.286039	-11.713961
5	6	15	12	0	16	13.339333	-2.660667
6	7	15	16	0	17	15.781969	-1.218031
7	8	17	15	0	12	15.171310	3.171310
8	9	10	8	1	8	10.896698	2.896698
9	10	8	6	1	7	9.675380	2.675380

```
In [42]: rmse = np.sqrt((df_sis_data['Predicted Err']**2).mean())
         rmse

Out[42]: 4.221955437290858
```

We also want to calculate the Root Mean Squared Error of this estimate, which is done by first calculating the error between our 2012 predicted score and the 2012 actual scores**.** The mean for the 'Predicted Error' column will give us an RMSE of 4.22. We can proceed with a new DiD analysis using the shrinkage coefficient and the predicted 2012 scores as the before scores and the actual 2012 scores our after scores.

d) We can now find the average of the estimated and actual 2012 scores for both the SIS students and non-SIS students, like we did before, to get the correct results for the Difference in Differneces analysis.

```
In [48]: print('Control Group New')
         print(f'Predicted:{round(control_before_c,2 )}')
         print(f'Actual:{round(control_after_c,2 )}')
         print(f'Difference:{round(control_after_c-control_before_c, 2)}')
         print(f'% difference:{round(control_perc_diff_new,2)}%')
         print('\nTreatment Group New')
         print(f'Predicted:{round(treatment_before_c,2 )}')
         print(f'Actual:{round(treatment_after_c,2 )}')
         print(f'Difference:{round(treatment_after_c-treatment_before_c, 2)}')
         print(f'% difference:{round(treatment_perc_diff_new,2)}%')

         print(f'\nDifference in differences:{round(treatment_perc_diff_new-control_perc_diff_new, 2)}%')

         Control Group New
         Predicted:16.2
         Actual:16.13
         Difference:-0.07
         % difference:-0.45%

         Treatment Group New
         Predicted:11.43
         Actual:12.88
         Difference:1.44
         % difference:12.63%

         Difference in differences:13.08%
```

We notice the difference in differences is 13.08%, far lower than our initial result of 48.42% .

e) Initially, our analysis determined the difference in scores between 2011 and 2012 between the treatment and control groups, but we noticed from the results of the regression that there may be a reversion towards the mean and may not tell us the true impact of the SIS program. Our second approach compares 2012 actual scores with 2012 predicted scores and therefore uses the shrinkage coefficient to incorporate the impact of regression to the mean. This method more accurately reflects the true effectiveness of the SIS program.

Problem 2:

a) As owners of the small boutique hotel in Montauk, we wanted to analyze the behavior of our customers to know the expected number of guests on a given night, as well as the probability of having all rooms occupied. To do this, we use a simulation model. We consider our random variable to be a Binomial Variable because we have two possible outcomes-Cancelled or not. (We assume that on a given night we receive 20 reservations, but some of those 20 reservations cancel randomly. The probability that a given reservation cancels is 0.16(therefore probability of success=1-0.16=0.84) . We need the number of guests that showed up and whether or not the hotel is full. But this would only reflect the results for one specific night.

```
In [50]: np.random.seed(123) # initialize random seed
         prob_occupied = np.random.binomial(n=1, p=0.84, size=20)
         no_of_guests = prob_occupied.sum()
         full = True if no_of_guests == 20 else False
         no_of_guests, full

Out[50]: (19, False)
```

b) We then create a function to calculate the average occupancy of the hotel instead of looking at results for just one night. The function accepts four arguments:

- bookings: the number of bookings the hotel acceps
- rooms: the number of rooms in the hotel
- n: the number of simulations to use
- seed: the seed to use for the simulation

It returns a tuple containing the average occupancy, the probability of the hotel being full, and the average number of walked customers we may have to compensate for the cancellations.

```
In [67]: def avg_occ(bookings, rooms=20, n=1000, seed=123):

             np.random.seed(seed)
             occupancies = []
             full = []
             avg_walked = []

             for i in range(n):
                 prob_occupied = np.random.binomial(n=1, p=0.84, size=bookings)

                 occupancies.append(prob_occupied.sum())

                 if prob_occupied.sum() >= rooms:
                     full.append(1)
                     avg_walked.append(prob_occupied.sum()-rooms)
                 else:
                     full.append(0)
                     avg_walked.append(0)

             return (np.mean(prob_occupied), np.mean(full), np.mean(avg_walked))
```

```
In [56]: avg_occ(20)
Out[56]: (16.852, 0.035, 0.0)
```

Based on our simulation, we see that the average occupancy is roughly 17 when we have 20 rooms, and the probability of the hotel being full is just 3.5%, with 0 probable walked customers.

c) If we consider overbooking the hotel by one room and accept 21 reservations, hoping that there is a chance that a customer might have to be "walked" if all 21 reservations arrive. We create a new simulation with 1,000 days of operation of the hotel with this new overbooking policy. We can add the data to a dataframe as well, to see the average results on different nights. We find the average occupancies, the average rate at which the hotel would get full, and the average number of customers that may have walked. We see the new results as:

```
In [75]: avg_occ(21)
Out[75]: (0.6190476190476191, 0.135, 0.03)
```

d) We now simulate the model for overbooking(by 2,3,4) that is: accept 22,23 and 24 reservations. We run the same model for different values of 'n' from 18-30 to account for underbooking and also checking which value of 'n' would be most suitable.

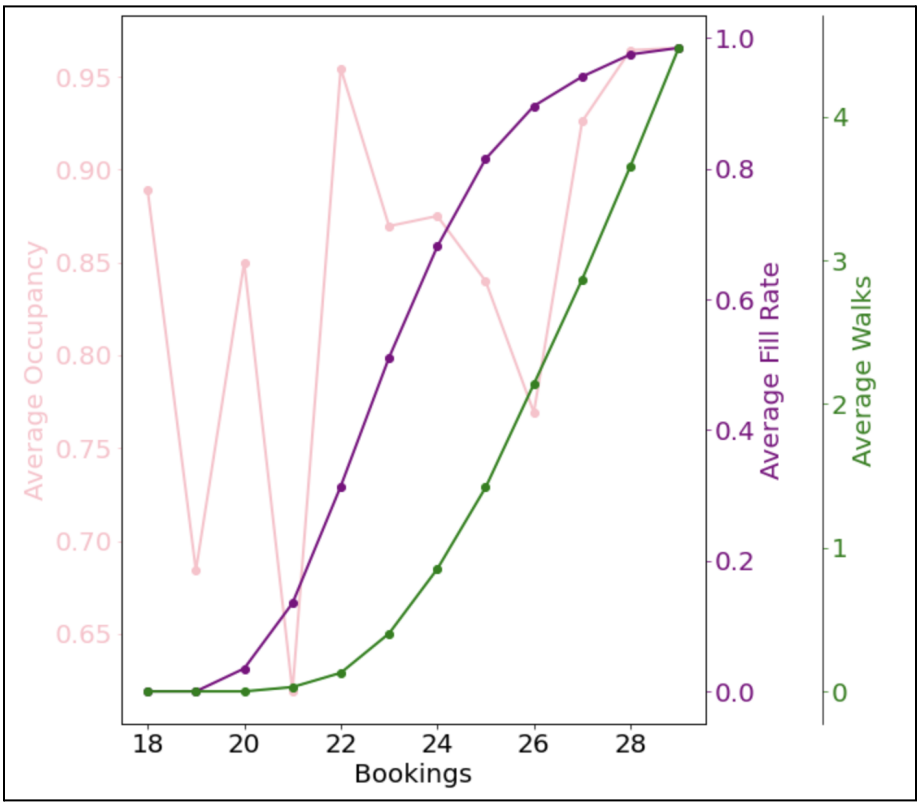
```
In [69]: bookings = [i for i in range(18, 30)]
avg_occupancy = []
avg_fill_rate = []
avg_walk_customers = []
for i in bookings:
    a, b, c =avg_occ(i)
    avg_occupancy.append(a)
    avg_fill_rate.append(b)
    avg_walk_customers.append(c)

df_hotel = pd.DataFrame({'Bookings': bookings, 'Avg Occupancy': avg_occupancy,
                         'Avg Fill Rate': avg_fill_rate, 'Avg Walks': avg_walk_customers})
df_hotel
```

Out[69]:

	Bookings	Avg Occupancy	Avg Fill Rate	Avg Walks
0	18	0.888889	0.000	0.000
1	19	0.684211	0.000	0.000
2	20	0.850000	0.035	0.000
3	21	0.619048	0.135	0.030
4	22	0.954545	0.314	0.129
5	23	0.869565	0.511	0.403
6	24	0.875000	0.681	0.852
7	25	0.840000	0.815	1.419
8	26	0.769231	0.896	2.139
9	27	0.925926	0.941	2.867
10	28	0.964286	0.975	3.655
11	29	0.965517	0.985	4.478

To analyse it better, we can also plot the various metrics.

```
In [74]: fig, ax = plt.subplots(figsize=(10,10))
         axes = [ax, ax.twinx(), ax.twinx()]
         fig.subplots_adjust(right=0.75)
         axes[-1].spines['right'].set_position(('axes', 1.2))
         axes[-1].set_frame_on(True)
         axes[-1].patch.set_visible(False)
         data = (avg_occupancy, avg_fill_rate, avg_walk_customers)
         labels = ('Average Occupancy', 'Average Fill Rate', 'Average Walks')
         colors = ('Pink', 'Purple', 'Green')
         for ax, data, label, color in zip(axes, data, labels, colors):
             ax.plot(bookings, data, marker='o', linewidth=2, color=color)
             ax.set_ylabel(label, color=color, size=20)
             ax.tick_params(axis='y', colors=color, labelsize=20)
             ax.tick_params(axis='x', labelsize=20)
             axes[0].set_xlabel('Bookings', size=20)
```

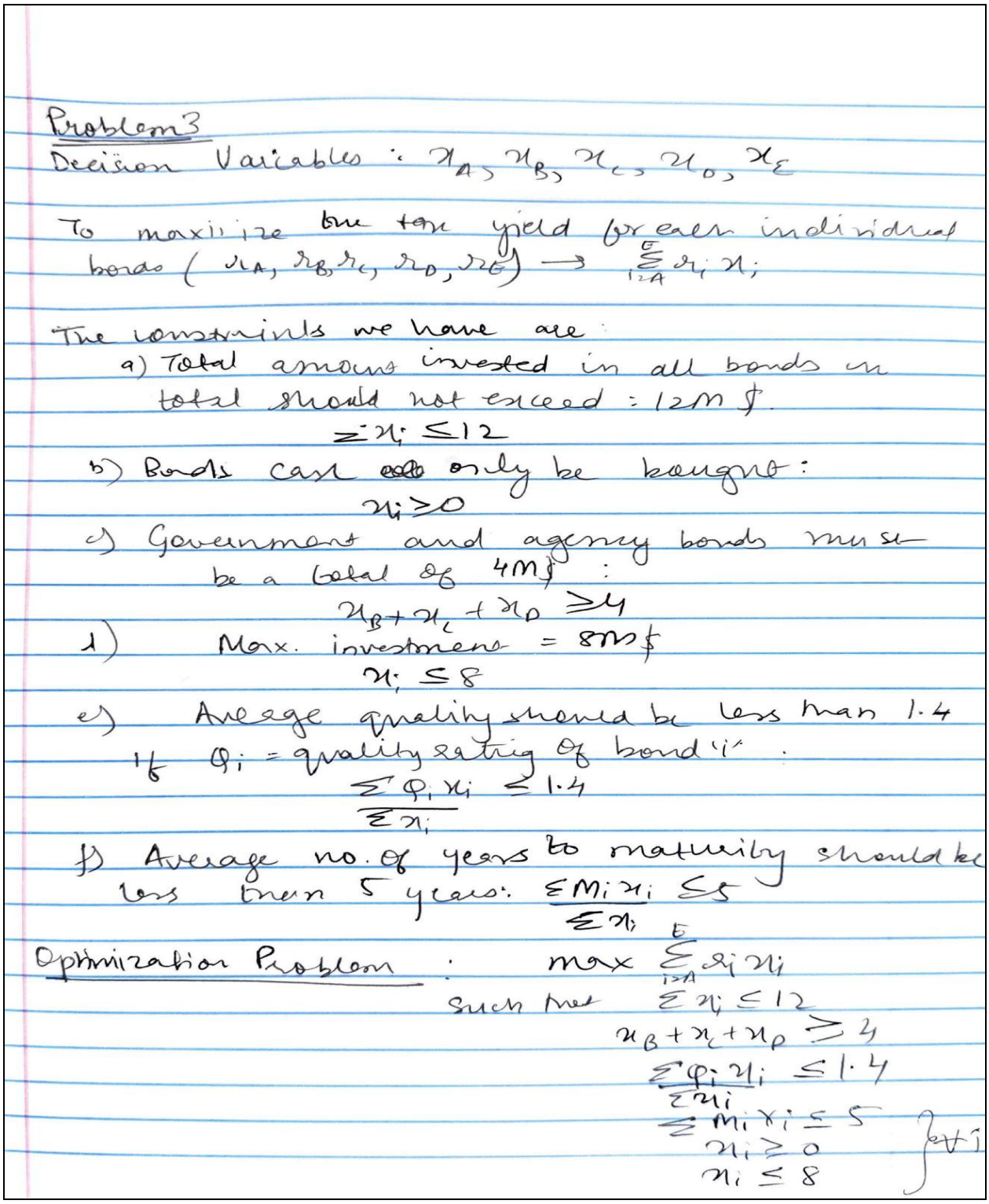


We observe that the average occupancy increases erratically with the number of bookings, which is not something we had expected to see since the probability of a customer showing for their booking was constant at 84% (being constant should have led to a linear increase). The average fill rate (probability of the hotel being fully occupied) increases slowly at first and then levels out around 27 or 28 bookings. Lastly, the average number of walks increases at first until the average occupancy begins to rise above 20, then the average walks starts to grow rapidly.

e) To decide the best number of overbooking reservations to make, we must consider the implicit trade-off between maintaing full occupancy and guests walking to another hotel. For example, if we set the level of bookings at 25, we expect the hotel to be full 80% of the time which is a great increase as compared to the 3.4% when accepting exactly 20 bookings, but we might have to also expect 2 guests walking each night. Having unoccupied room comes with a opportunity cost equal to the unseen revenue. If we overbook too much, it would also cost us money for the guests that walked as the hotel without open rooms usually compensates the guest to stay at another hotel. Thus we need to determine the optimal level of bookings is a problem of cost minimization. There may be some other latent factors that might be affecting the

customers' decisions. In my opinion, accepting around 22 bookings or so(overbooking slightly, but not too much) could help us make some profit without losing out on the opportunity cost or reimbursement fee due to walking.

Problem 3:



Problem 3

Decision Variables : x_A, x_B, x_C, x_D, x_E

To maximize the tax yield for each individual bonds (r_A, r_B, r_C, r_D, r_E) $\rightarrow \sum_{i=A}^{E} r_i x_i$

The constraints we have are :

a) Total amount invested in all bonds in total should not exceed : 12M $.

$$\sum x_i \leq 12$$

b) Bonds can only be bought :

$$x_i \geq 0$$

c) Government and agency bonds must be a total of 4M $:

$$x_B + x_C + x_D \geq 4$$

d) Max. investment = 8M $

$$x_i \leq 8$$

e) Average quality should be less than 1.4

If q_i = quality rating of bond 'i' :

$$\frac{\sum q_i x_i}{\sum x_i} \leq 1.4$$

f) Average no. of years to maturity should be less than 5 years: $\frac{\sum m_i x_i}{\sum x_i} \leq 5$

Optimization Problem :

$$\max \sum_{i=A}^{E} r_i x_i$$

such that

$$\sum x_i \leq 12$$

$$x_B + x_C + x_D \geq 4$$

$$\frac{\sum q_i x_i}{\sum x_i} \leq 1.4$$

$$\sum m_i x_i \leq 5$$

$$x_i \geq 0$$

$$x_i \leq 8$$

} $\forall i$

We begin the problem by framing our optimization problem keeping in mind all the constraints. We then go to Python. We can use the PuLP library to solve such problems.

```
In [82]: df_bond_data = pd.read_excel('homework 4.xlsx', sheet_name = 'pb3_data')
         df_bond_data
```

Out[82]:

	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

```
In [84]: import pulp as pl
         m = pl.LpProblem('Portfolio', pl.LpMaximize)

In [86]: x = []
         for bond in df_bond_data.index:
             x.append( pl.LpVariable(f'a_{bond}', cat='Continuous') )

         x = np.array(x)
```

We begin by importing the data from the excel workbook as a Pandas data frame, and use the PuLP library to solve the optimization problem. We add all the constraints we used to frame the problem.

```
In [97]: m += (pl.lpSum(x) <= 12, 'Total allocation constraint')
         m += (x[1]+x[2]+x[3] >= 4, 'Gov agency constraint')
         for bond in df_bond_data.index:
             m += (x[bond] <= 8, f'bond {bond} max constraint')
             m += (x[bond] >= 0, f'No leverage constraint {bond}')
         m += (pl.lpSum(x*np.array(df_bond_data['Bank Rating'])) <= 1.4*pl.lpSum(x), 'Bank rating constraint')
         m += (pl.lpSum(x*np.array(df_bond_data['Maturity (Yrs.)'])) <= 5*pl.lpSum(x), 'Maturity constraint')
```

We then create the objective function and maximize based on the constraints.

```
In [98]: # Create objective
         m += pl.lpSum(np.array(df_bond_data['After-Tax Yield'])*x)

In [99]: m

Out[99]: Portfolio:
         MAXIMIZE
         0.044*a_0 + 0.028*a_1 + 0.026*a_2 + 0.021*a_3 + 0.047*a_4 + 0.0
         SUBJECT TO
         Total_allocation_constraint: a_0 + a_1 + a_2 + a_3 + a_4 <= 12

         Gov_agency_constraint: a_1 + a_2 + a_3 >= 4

         bond_0_max_constraint: a_0 <= 8

         No_leverage_constraint_0: a_0 >= 0

         bond_1_max_constraint: a_1 <= 8

         No_leverage_constraint_1: a_1 >= 0

         bond_2_max_constraint: a_2 <= 8

         No_leverage_constraint_2: a_2 >= 0
```

We check for the locally available solvers that we can use to solve the problem and analyze the data.

```
In [100]: # View available solvers
          pl.list_solvers(onlyAvailable=True)

Out[100]: ['PULP_CBC_CMD']

In [101]: # Solve the linear optimization problem
          pl.PULP_CBC_CMD().solve(m)

          Welcome to the CBC MILP Solver
          Version: 2.10.3
          Build Date: Dec 15 2019

          command line - /Users/vriddhimisra/opt/anaconda3/lib/python3.9/site-packages/pulp/solverdir/cbc/osx/64/cbc /var/folde
          rs/rp/nw6h5p9140s6nymnfr9lmlbh0000gn/T/a46d03ba862342b5ab7c101a0aef05dd-pulp.mps max timeMode elapsed branch printing
          Options all solution /var/folders/rp/nw6h5p9140s6nymnfr9lmlbh0000gn/T/a46d03ba862342b5ab7c101a0aef05dd-pulp.sol (defa
          ult strategy 1)
          At line 2 NAME          MODEL
          At line 3 ROWS
          At line 19 COLUMNS
          At line 53 RHS
          At line 68 BOUNDS
          At line 74 ENDATA
          Problem MODEL has 14 rows, 5 columns and 27 elements
          Coin0008I MODEL read with 0 errors
          Option for timeMode changed from cpu to elapsed
          Presolve 4 (-10) rows, 5 (0) columns and 17 (-10) elements
          0  Obj -0 Primal inf 3.999999 (1) Dual inf 0.165995 (5)
          4  Obj 0.336928
          Optimal - objective value 0.336928
          After Postsolve, objective 0.336928, infeasibilities - dual 0 (0), primal 0 (0)
          Optimal objective 0.336928 - 4 iterations time 0.002, Presolve 0.00
          Option for printingOptions changed from normal to all
          Total time (CPU seconds):       0.00   (Wallclock seconds):       0.00

Out[101]: 1
```

Finally, we get the following result with the optimal solution and allocations.

```
In [105]: optimal_solution = m.objective.value()
          optimal_solution

Out[105]: 0.336928

In [106]: # Add optimal allocations to original DataFrame
          df_bond_data['Allocation'] = [x[i].value() for i in range(len(x))]
          df_bond_data
```

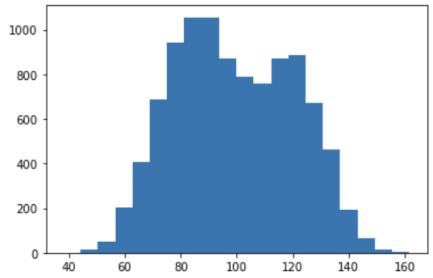
Out[106]:

	Name	Type	Rating	Bank Rating	Maturity (Yrs.)	Yield to Maturity	After-Tax Yield	Allocation
0	A	Municipal	AA	2	10	0.044	0.044	0.832
1	B	Agency	AA	2	15	0.055	0.028	0.000
2	C	Government	AAA	1	5	0.051	0.026	8.000
3	D	Government	AAA	1	4	0.043	0.021	2.176
4	E	Municipal	BB	5	3	0.047	0.047	0.992

Problem 4:

a) Based on historical data of the bakery, the estimated demand each morning is normally distributed with mean 50 and standard deviation 10. In the afternoon, demand is uniformly distributed between 60 and 80 on sunny days and uniformly distributed between 20 and 50 on rainy days. The probability of a sunny day is 0.4. We begin by generating the morning demand as a random normal variable, and then generating the afternoon demand as a uniform normal distribution. To determine the distribution for the afternoon demand, we also need to determine if the day is sunny or rainy. We also need to simulate 10,000 days of total demand and create a histogram of daily demand.

```
In [109]: demands = []
          for i in range(10000):
              demand_morn = np.random.normal(50, 10)
              rain = 1 if np.random.uniform() >= 0.4 else 0
              if rain:
                  demand_aft = 20 + 30*np.random.uniform()
              else:
                  demand_aft = 60 + 20*np.random.uniform()

              demand_tot = demand_morn + demand_aft

              demands.append(demand_tot)

In [110]: import matplotlib.pyplot as plt
          plt.hist(demands, bins = 20)

Out[110]: (array([1.000e+00, 1.200e+01, 5.200e+01, 2.050e+02, 4.060e+02, 6.890e+02,
                  9.430e+02, 1.055e+03, 1.052e+03, 8.700e+02, 7.910e+02, 7.600e+02,
                  8.710e+02, 8.850e+02, 6.740e+02, 4.610e+02, 1.930e+02, 6.500e+01,
                  1.200e+01, 3.000e+00]),
           array([ 37.98192797,  44.16816817,  50.35440837,  56.54064857,
                   62.72688877,  68.91312896,  75.09936916,  81.28560936,
                   87.47184956,  93.65808976,  99.84432996, 106.03057015,
                  112.21681035, 118.40305055, 124.58929075, 130.77553095,
                  136.96177114, 143.14801134, 149.33425154, 155.52049174,
                  161.70673194]),
           <BarContainer object of 20 artists>)
```



Based on the histogram and the demands calculated, we can estimate the 10th and 9th percentile as follows:

```
In [111]: demands.sort()
          print(f'10th percentile:      {round(demands[1000],2 )}')
          print(f'90th percentile:      {round(demands[9000],2 )}')

          10th percentile:      72.15
          90th percentile:      128.0
```

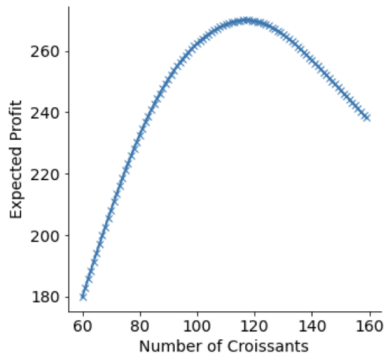
b) We can take the simulation code we wrote and create a function in order to check demand and price for the different quantities of croissants. The function accepts four arguments: croissants: the number of croissants the bakery bakes each morning; cost: the cost to produce each croissant; price: the revenue received from the sale of each croissant; and the seed: the seed to use for the simulation.
Using the formula: profit = price * min(total demand, croissants) – cost * croissants, we can calculate the profit for the bakery.

```
In [112]: def bakery_deman_profit_simulation(croissants, cost=1, price=4, seed=123):
              np.random.seed(seed)
              demands = []
              profits = []
              for i in range(10000):
                  demand_morn = np.random.normal(50, 10)
                  rain = 1 if np.random.uniform() >= 0.4 else 0
                  if rain:
                      demand_aft = 20 + 30*np.random.uniform()
                  else:
                      demand_aft = 60 + 20*np.random.uniform()
                  demand_tot = demand_morn + demand_aft
                  demands.append(demand_tot)
                  profits.append(price*min(demand_tot, croissants) - cost*croissants)
              return demands, profits

In [114]: demand, profits = bakery_deman_profit_simulation(120)
          np.mean(profits)

Out[114]: 269.6739413149552
```

The expected profit is $269.67 if 120 croissants are made every day.

c) To calculate the optimal number of Croissants and Optimal profit, we use the simulation results over different values of 'number of croissants' and calculate the demand and profit for each number and find the maximum profit. We can also plot the number of croissants against the profit to visualize the optimization problem.

```
In [130]: croissants = [60 + i for i in range(100)]
          profs = []
          for num in croissants:
              demand, profits = bakery_deman_profit_simulation(num)
              profs.append(np.mean(profits))

In [131]: import seaborn as sns

          plt.figure(figsize=(5,5))

          plt.plot(croissants, profs, marker='x', linewidth=2)

          plt.xlabel('Number of Croissants', fontsize=14)
          plt.ylabel('Expected Profit', fontsize=14)

          plt.xticks(fontsize=14)
          plt.yticks(fontsize=14)

          sns.despine()
```

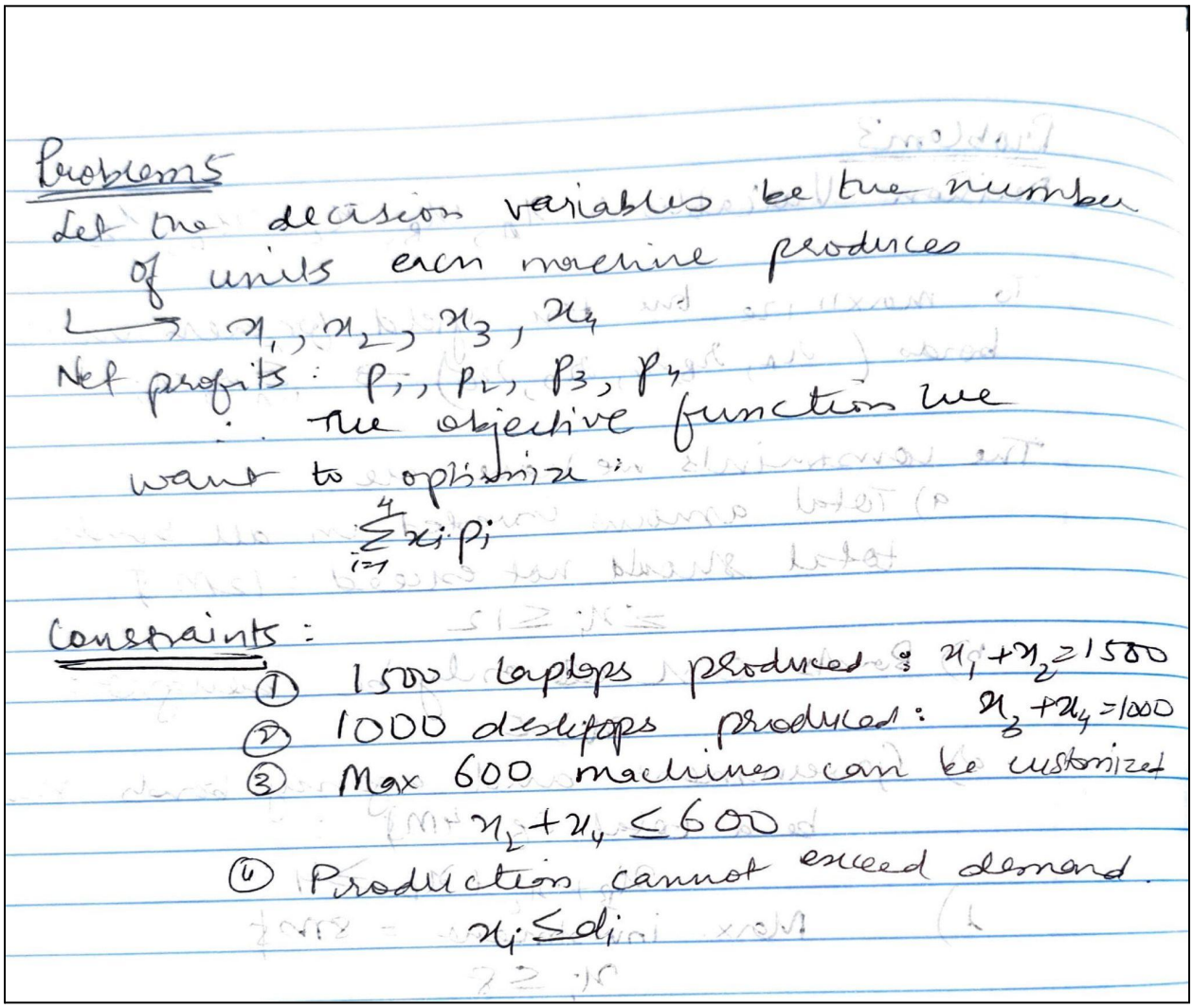


We can observe that the maximum expected profit is achieved at around 120 croissants. The maximum expected profit of $269.94, only 27 cents higher than at 120, is achieved at 117 croissants.

```
In [132]: print(f'Max profit:              {round(max(profs),2 )}')
          print(f'Optimal croissants:     {round(croissants[profs.index(max(profs))],2 )}')

          Max profit:              269.94
          Optimal croissants:      117
```

Problem 5:

a) We begin by framing our optimization problem by deciding our decision variables and set the objective function that we want to optimize. We also lay down the constraints that are given in the problem.



Based on the relationship between the variables, it is a linear problem.

b) We begin the optimization in Python using the PuLP library. We import the computers' data as a dataframe.

```
In [134]: df_computers = pd.DataFrame([['Standard Laptop', 1200, 100], ['Custom Laptop', 1000,200],
                                       ['Standard Desktop', 700, 150], ['Custom Desktop', 400, 400]],
                                      columns=['Type', 'Demand', 'Net Profit'])
          df_computers
```

Out[134]:

	Type	Demand	Net Profit
0	Standard Laptop	1200	100
1	Custom Laptop	1000	200
2	Standard Desktop	700	150
3	Custom Desktop	400	400

Since it is a maximization problem, we do Lp.Maximize. Then we go on to make the variables.

```
In [137]: n = pl.LpProblem('Laptops', pl.LpMaximize)

In [138]: y = []
          for comp in df_computers.index:
              y.append( pl.LpVariable(f'q_{comp}', cat='Continuous') )

          y = np.array(y)
```

Then we write the constraints as we framed them previously and calculate the maximized net profit based on the all the data.

```
In [139]: n += (y[0] + y[1] == 1500, 'Laptop production constraint')
          n += (y[2] + y[3] == 1000, 'Desktop production constraint')
          n += (y[1] + y[3] <= 600, 'Customization constraint')
          for comp in df_computers.index:
              n += (y[comp] <= df_computers.Demand[comp], f'Comp {comp} demand constraint')

In [141]: n += pl.lpSum(np.array(df_computers['Net Profit'])*y)

In [142]: n

Out[142]: Laptops:
          MAXIMIZE
          100*q_0 + 200*q_1 + 150*q_2 + 400*q_3 + 0
          SUBJECT TO
          Laptop_production_constraint: q_0 + q_1 = 1500

          Desktop_production_constraint: q_2 + q_3 = 1000

          Customization_constraint: q_1 + q_3 <= 600

          Comp_0_demand_constraint: q_0 <= 1200

          Comp_1_demand_constraint: q_1 <= 1000

          Comp_2_demand_constraint: q_2 <= 700

          Comp_3_demand_constraint: q_3 <= 400

          VARIABLES
          q_0 free Continuous
          q_1 free Continuous
          q_2 free Continuous
          q_3 free Continuous
```

After using the Solver avaiable on the local machine, we get the optimal solution from the value returned by the objective function as $405,000, which us the maximum net profit. We can make a dataframe with the allocated values for production of the laptops and desktops based on the demand and net profit.

```
In [147]: # Obtain optimal solution
          optimal_solution = n.objective.value()
          optimal_solution

Out[147]: 405000.0

In [148]: # Add optimal allocations to original DataFrame
          df_computers['Production'] = [y[i].value() for i in range(len(y))]
          df_computers
```

Out[148]:

	Type	Demand	Net Profit	Production
0	Standard Laptop	1200	100	1200.0
1	Custom Laptop	1000	200	300.0
2	Standard Desktop	700	150	700.0
3	Custom Desktop	400	400	300.0

c) If we are able to customize 200 more machines, let us assume that the production for custom laptops increases by 100(now 300) and that of custom desktops increases by 100 (now 400) , simultaneously reducing the standard production by 200. We get a new solution which produces a total profit of $440,000. eg)

	Type	Demand	Net Profit	Production
0	Standard Laptop	1200	100	1100.0
1	Custom Laptop	1000	200	400.0
2	Standard Desktop	700	150	600.0
3	Custom Desktop	400	400	400.0

d) Similarly**,** if we manufacture 100 fewer laptops(100 more desktops), we get another new solution that is also more profitable than the first, netting a total profit of $410,000.