Group Members:
Vivian Wang (vw3852)
Boran Sheu (bs42973)
Parthiv Borgohain (pb25347)
Rukh Agha (msa3453)

Problem Description

Marketing budgets now comprise 11 percent of total company budgets, based on a CMO survey sponsored by the Fuqua School of Business at Duke University, Deloitte LLP, and the American Marketing Association. However, the effectiveness of marketing varies significantly: on the one hand, P&G cut more than $100 million in digital marketing spending because their digital ads were largely ineffective; on the other hand, Netflix plans a 54% boost in ad spending because they got very positive feedback in international markets.

One potential reason for such variation is the way of making marketing budget allocations. Namely, how much to invest in each advertisement platform. In this project, we use linear programming to build a simple marketing budget allocation strategy.

Section 1: Our goal is to recommend how to spread a budget of $10M among several marketing mediums to our company. The ROI for each of the marketing mediums are given as follows:

Platform	Print	TV	SEO	AdWords	Facebook	LinkedIn	Instagram	Snapchat	Twitter	Email
ROI	3.1%	4.9%	2.4%	3.9%	1.6%	2.4%	4.6%	2.6%	3.3%	4.4%

Section 2: In addition, our budget has to meet three constraints imposed by our boss:

- The amount invested in print and TV should be no more than the amount spent on Facebook and Email. Surprisingly, email seems to be a great channel for reaching real people.
- The total amount used in social media (Facebook, LinkedIn, Instagram, Snapchat, and Twitter) should be at least twice of SEO and AdWords.
- For each platform, the amount invested should be no more than $3M.

Section 3: Linear Programming - Marketing Budget:

```python
obj1 = np.array(roi.iloc[0]) # objective vector

A1 = np.zeros((3,len(roi.columns))) # initialize constraint matrix

# budget constraint
A1[0,:] = [1]*len(roi.columns)

# constraint a
A1[1,roi.columns.get_loc('Print')] = 1; A1[1, roi.columns.get_loc('TV')] = 1
A1[1,roi.columns.get_loc('Facebook')] = -1; A1[1, roi.columns.get_loc('Email')] = -1

# contraint b
A1[2,roi.columns.get_loc('Facebook')] = 1; A1[2, roi.columns.get_loc('LinkedIn')] = 1
A1[2,roi.columns.get_loc('Instagram')] = 1; A1[2, roi.columns.get_loc('Snapchat')] = 1
A1[2,roi.columns.get_loc('Twitter')] = 1
A1[2,roi.columns.get_loc('SEO')] = -2; A1[2, roi.columns.get_loc('AdWords')] = -2

b1 = np.array([10,0,0])
sense1 = np.array(['<','<','>'])
```

```python
ojModel1 = gp.Model() # initialize an empty model

ojModX1 = ojModel1.addMVar(len(roi.columns), lb=np.array([0]*len(roi.columns)), ub = np.array([3]*len(roi.columns)))
# lower bound of zero and upper bound of 3 million for every platform
# must define the variables before adding constraints because variables go into the constraints

ojModCon1 = ojModel1.addMConstrs(A1, ojModX1, sense1, b1) # add the constraints to the model
ojModel1.setMObjective(None,obj1,0,sense=gp.GRB.MAXIMIZE) # add the objective to the model
# quadratic = None, constant = 0(does not affect output)

ojModel1.Params.OutputFlag = 0 # tell gurobi to shut up!!
ojModel1.Params.TimeLimit = 3600
```

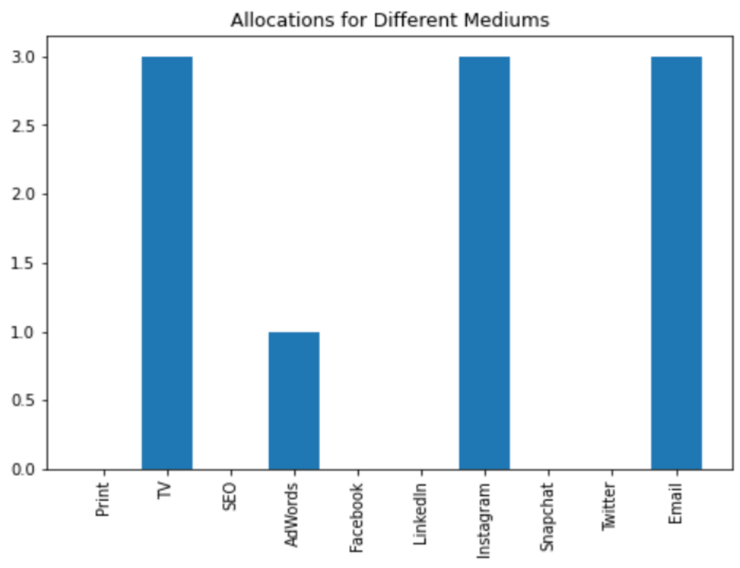
```
Restricted license - for non-production use only - expires 2023-10-25
```

```python
ojModel1.optimize()
mod1_objval = ojModel1.objVal
mod1_x = ojModX1.x
print("Optimal return is ",mod1_objval,"Million USD")

df1 = pd.DataFrame()
df1['First_ROI'] = ['ROI','Allocation ($M)']
columns = list(roi.columns.values)
x = 0

for col in columns:
    df1[col] = [obj1[x],mod1_x[x]]
    x += 1

df1 = df1.set_index('First_ROI')
df1
```

```
Optimal return is  0.45600000000000007 Million USD
```

	Print	TV	SEO	AdWords	Facebook	LinkedIn	Instagram	Snapchat	Twitter	Email
First_ROI										
ROI	0.031	0.049	0.024	0.039	0.016	0.024	0.046	0.026	0.033	0.044
Allocation ($M)	0.000	3.000	0.000	1.000	0.000	0.000	3.000	0.000	0.000	3.000

The optimal allocation under these ROIs is to invest **$3 million in TV**, **$1 million in AdWords**, **$3 million in Instagram**, and **$3 million in Email**.



Section 4: Alternative ROI Programming:

A second set of ROI estimates read as follows:

Platform	Print	TV	SEO	AdWords	Facebook	LinkedIn	Instagram	Snapchat	Twitter	Email
ROI	4.9%	2.3%	2.4%	3.9%	4.4%	4.6%	2.6%	1.9%	3.7%	2.6%

```
obj2 = np.array(roi.iloc[1]) # objective vector

A2 = np.zeros((3,len(roi.columns))) # initialize constraint matrix

# budget constraint
A2[0,:] = [1]*len(roi.columns)

# cosntraint a
A2[1,roi.columns.get_loc('Print')] = 1; A1[1, roi.columns.get_loc('TV')] = 1
A2[1,roi.columns.get_loc('Facebook')] = -1; A1[1, roi.columns.get_loc('Email')] = -1

# constraint b
A2[2,roi.columns.get_loc('Facebook')] = 1; A2[2, roi.columns.get_loc('LinkedIn')] = 1
A2[2,roi.columns.get_loc('Instagram')] = 1; A2[2, roi.columns.get_loc('Snapchat')] = 1
A2[2,roi.columns.get_loc('Twitter')] = 1
A2[2,roi.columns.get_loc('SEO')] = -2; A2[2, roi.columns.get_loc('AdWords')] = -2

b2 = np.array([10,0,0])
sense2 = np.array(['<','<','>'])
```

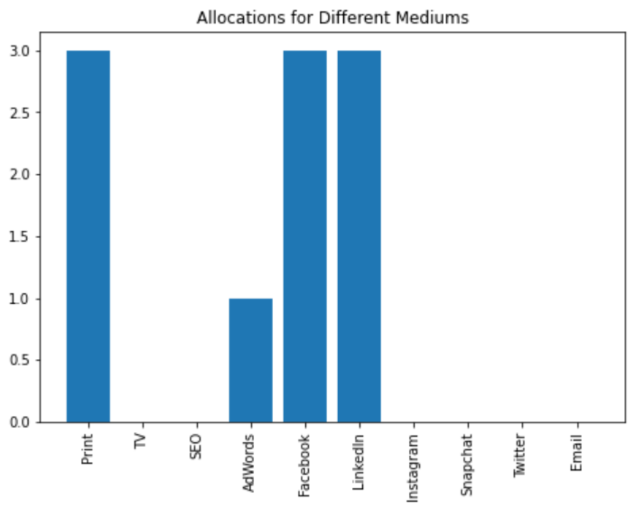
```
ojModel2 = gp.Model() # initialize an empty model

ojModX2 = ojModel2.addMVar(len(roi.columns), lb=np.array([0]*len(roi.columns)), ub = np.array([3]*len(roi.columns)))
# lower bound of zero and upper bound of 3 million for every platform
# must define the variables before adding constraints because variables go into the constraints

ojModCon2 = ojModel2.addMConstrs(A2, ojModX2, sense2, b2) # add the constraints to the model
ojModel2.setMObjective(None,obj2,0,sense=gp.GRB.MAXIMIZE) # add the objective to the model
# quadratic = None, constant = 0(does not affect output)

ojModel2.Params.OutputFlag = 0 # tell gurobi to shut up!!
ojModel2.Params.TimeLimit = 3600
```

```
ojModel2.optimize()
mod2_objval = ojModel2.objVal
mod2_x = ojModX2.x
print("Optimal return is ",mod2_objval,"Million USD")

df2 = pd.DataFrame()
df2['Second_ROI'] = ['ROI','Allocation ($M)']
columns = list(roi.columns.values)
x = 0

for col in columns:
    df2[col] = [obj2[x],mod2_x[x]]
    x += 1

df2.set_index('Second_ROI')
df2
```

```
Optimal return is  0.45600000000000007 Million USD
```

Second_ROI	Print	TV	SEO	AdWords	Facebook	LinkedIn	Instagram	Snapchat	Twitter	Email
ROI	0.049	0.023	0.024	0.039	0.044	0.046	0.026	0.019	0.037	0.026
Allocation ($M)	3.00	0.00	0.00	1.00	3.00	3.00	0.00	0.00	0.00	0.00

Using this set of estimates, the optimal allocation we obtained is to invest **$3 million in Print**, **$1 million in AdWords**, **$3 million in Facebook**, and **$3 million in LinkedIn**.



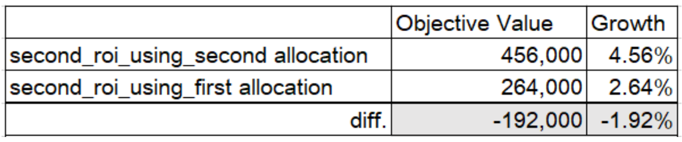
Section 5: Comparing ROI Outcomes

The two allocations are not the same.

Assuming the first ROI data is correct, if we use the second allocation, the objective would be $204,000 lower compared to the optimal objective. In other words, growth would drop from 4.56% to 2.52%.

	Objective Value	Growth
first_roi_using_first allocation	456,000	4.56%
first_roi_using_second allocation	252,000	2.52%
diff.	-204,000	-2.04%

Assuming the second ROI data is correct, if we use the first allocation, the objective would be $192,000 lower compared to the optimal objective . In other words, growth would drop from 4.56% to 2.64%.



	Objective Value	Growth
second_roi_using_second allocation	456,000	4.56%
second_roi_using_first allocation	264,000	2.64%
diff.	-192,000	-1.92%

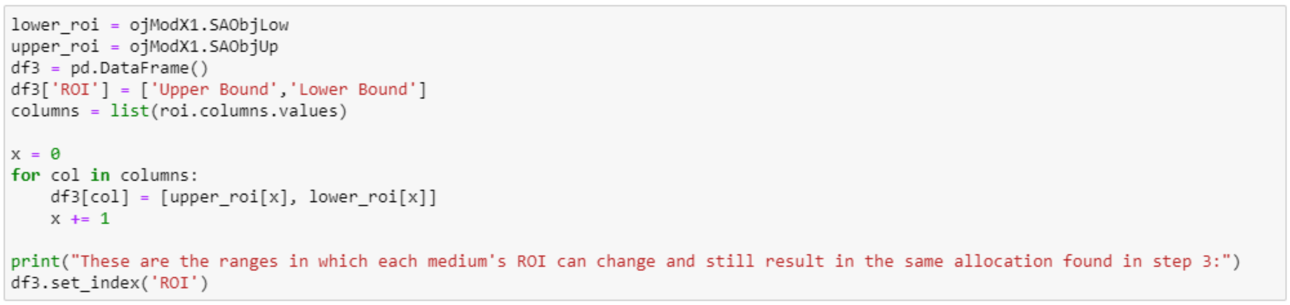
Analysis of 3rd Constraint - Hard-Capping Spend on Individual Channels:

Types	ROI	Misallocation
With 3rd constraint	4.56%	2.52% (initial) 2.64% (second)
Without 3rd constraint	4.65%	2.35% (initial) 2.45% (second)

Based on our analysis, the boss’ s experienced recommendation to hedge when ROI predictions are wrong by capping spend on individual categories is certainly a useful strategy, it avoids putting eggs in the same basket. Removing the cap would mean that if ROI predictions are right, then growth would be bolstered about 1.02x (see ROI column), but will shrink to about 0.93x when wrong (see Misallocation column) - meaning the downside risk of not capping spend on each category is too high compared to the benefits. Even if both cases were symmetrical, getting kicked while you’ re already down is more costly than a boost when you’ re ahead.

Section 6: ROI Upper and Lower Bounds for Objective

Using the first ROI data, these are the ranges in which each medium’ s ROI can change and still result in the same allocation stated in step 3:



```
lower_roi = ojModX1.SAObjLow
upper_roi = ojModX1.SAObjUp
df3 = pd.DataFrame()
df3['ROI'] = ['Upper Bound','Lower Bound']
columns = list(roi.columns.values)

x = 0
for col in columns:
    df3[col] = [upper_roi[x], lower_roi[x]]
    x += 1

print("These are the ranges in which each medium's ROI can change and still result in the same allocation found in step 3:")
df3.set_index('ROI')
```

	Print	TV	SEO	AdWords	Facebook	LinkedIn	Instagram	Snapchat	Twitter	Email
ROI										
Upper Bound	0.049	0.062	0.039	0.046	0.029	0.039	inf	0.039	0.039	inf
Lower Bound	-inf	0.039	-inf	0.033	-inf	-inf	0.039	-inf	-inf	0.029

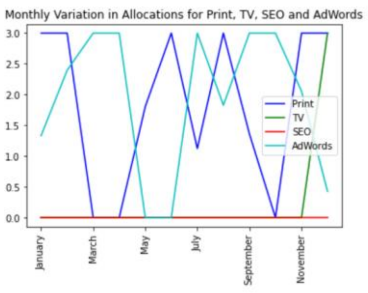
Section 7: Annual Budgeting with Reinvestment

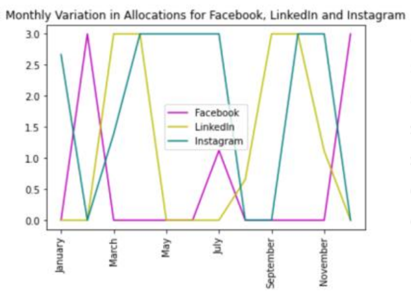
Now we are allowed to reinvest half of the return. The three constraints given by our boss are still in place for each month. The optimal allocation for each month is as follows (numbers are in units of $M):

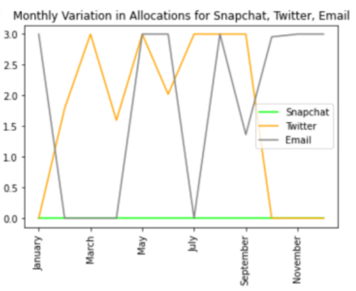
```
df4 = pd.DataFrame(columns = list(roi_mat.columns.values))

initial_amount = 10
return_percentage = 0
profit = []
total_amount = []

A = np.zeros((3,len(roi_mat.columns))) # initialize constraint matrix
A[0,:] = [1]*len(roi_mat.columns) # budget constraint
# A[1,:] = [1,1,0,0,-1,0,0,0,0,-1] # cosntraint a
A[1,roi_mat.columns.get_loc('Print')] = 1; A[1, roi_mat.columns.get_loc('TV')] = 1
A[1,roi_mat.columns.get_loc('Facebook')] = -1; A[1, roi_mat.columns.get_loc('Email')] = -1

# A[2,:] = [0,0,-2,-2,1,1,1,1,1,0] # constraint b
A[2,roi_mat.columns.get_loc('Facebook')] = 1; A[2, roi_mat.columns.get_loc('LinkedIn')] = 1
A[2,roi_mat.columns.get_loc('Instagram')] = 1; A[2, roi_mat.columns.get_loc('Snapchat')] = 1
A[2,roi_mat.columns.get_loc('Twitter')] = 1
A[2,roi_mat.columns.get_loc('SEO')] = -2; A[2, roi_mat.columns.get_loc('AdWords')] = -2


for i in range(len(roi_mat)):
    obj = np.array(roi_mat.iloc[i])/100

    b = np.array([initial_amount,0,0])
    sense = np.array(['<','<','>'])

    ojModel = gp.Model() # initialize an empty model

    ojModX = ojModel.addMVar(len(roi_mat.columns), lb=np.array([0]*len(roi_mat.columns)), ub = np.array([3]*len(roi_mat.columns))
    # lower bound of zero and upper bound of 3 million for every platform
    # must define the variables before adding constraints because variables go into the constraints

    ojModCon = ojModel.addMConstrs(A, ojModX, sense, b) # add the constraints to the model
    ojModel.setMObjective(None,obj,0,sense=gp.GRB.MAXIMIZE) # add the objective to the model
    # quadratic = None, constant = 0(does not affect output)

    ojModel.Params.OutputFlag = 0 # tell gurobi to shut up!!
    ojModel.Params.TimeLimit = 3600

    #print(obj)
    ojModel.optimize()
    mod_objval = ojModel.objVal
```

```
    mod_x = ojModX.x

    # adding the allocations to df4
    df4.loc[len(df4)] = mod_x


    #allocation.append(list(mod_x))
    profit.append(mod_objval)

    if mod_objval > 0:
        return_percentage = mod_objval / initial_amount
        initial_amount += initial_amount * return_percentage * 0.5
        total_amount.append(initial_amount)

    #print(return_percentage)
    #print(initial_amount)
    #print()
```

	Print	TV	SEO	AdWords	Facebook	LinkedIn	Instagram	Snapchat	Twitter	Email	Profit ($M)	Amount_Available_to_Invest ($M)
January	3.000000	0.0	0.0	1.333333	0.000000	0.000000	2.666667	0.0	0.000000	3.000000	0.373000	10.186500
February	3.000000	0.0	0.0	2.395500	3.000000	0.000000	0.000000	0.0	1.791000	0.000000	0.406296	10.389648
March	0.000000	0.0	0.0	3.000000	0.000000	3.000000	1.389648	0.0	3.000000	0.000000	0.414417	10.596856
April	0.000000	0.0	0.0	3.000000	0.000000	3.000000	3.000000	0.0	1.596856	0.000000	0.414487	10.804100
May	1.804100	0.0	0.0	0.000000	0.000000	0.000000	3.000000	0.0	3.000000	3.000000	0.432143	11.020172
June	3.000000	0.0	0.0	0.000000	0.000000	0.000000	3.000000	0.0	2.020172	3.000000	0.454767	11.247555
July	1.123777	0.0	0.0	3.000000	1.123777	0.000000	3.000000	0.0	3.000000	0.000000	0.468655	11.481882
August	3.000000	0.0	0.0	1.827294	0.000000	0.654588	0.000000	0.0	3.000000	3.000000	0.487966	11.725865
September	1.362933	0.0	0.0	3.000000	0.000000	3.000000	0.000000	0.0	3.000000	1.362933	0.459220	11.955475
October	0.000000	0.0	0.0	3.000000	0.000000	3.000000	3.000000	0.0	0.000000	2.955475	0.427575	12.169263
November	3.000000	0.0	0.0	2.056421	0.000000	1.112842	3.000000	0.0	0.000000	3.000000	0.517376	12.427951
December	3.000000	3.0	0.0	0.427951	3.000000	0.000000	0.000000	0.0	0.000000	3.000000	0.516834	12.686368







Section 8: Pro Forma Budget Stability

The allocation we found is not stable. By summing up the values in the dataframe shown below , we see that spending changes by more than $1M in 42 cases.

	Print	TV	SEO	AdWords	Facebook	LinkedIn	Instagram	Snapchat	Twitter	Email
February	0.000000	0.000000	0.000000	1.062167	3.000000	0.000000	-2.666667	0.000000	1.791000	-3.000000
March	-3.000000	0.000000	0.000000	0.604500	-3.000000	3.000000	1.389648	0.000000	1.209000	0.000000
April	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.610352	0.000000	-1.403144	0.000000
May	1.804100	0.000000	0.000000	-3.000000	0.000000	-3.000000	0.000000	0.000000	1.403144	3.000000
June	1.195900	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.979828	0.000000
July	-1.876223	0.000000	0.000000	3.000000	1.123777	0.000000	0.000000	0.000000	0.979828	-3.000000
August	1.876223	0.000000	0.000000	-1.172706	-1.123777	0.654588	-3.000000	0.000000	0.000000	3.000000
September	-1.637067	0.000000	0.000000	1.172706	0.000000	2.345412	0.000000	0.000000	0.000000	-1.637067
October	-1.362933	0.000000	0.000000	0.000000	0.000000	0.000000	3.000000	0.000000	-3.000000	1.592543
November	3.000000	0.000000	0.000000	-0.943579	0.000000	-1.887158	0.000000	0.000000	0.000000	0.044525
December	0.000000	3.000000	0.000000	-1.628470	3.000000	-1.112842	-3.000000	0.000000	0.000000	0.000000

Programmatically: To add this as a constraint, a set of 20 (2 * channels) additional constraints (20x10) would be added to the "A" matrix as two stacked identity matrices of size 10x10 (channels). And for each iteration of monthly programming, the prior month's spend on each channel would be added/appended to the "B" RHS matrix twice: one time with spend matrix minus one and one time with spend matrix plus one. The sense matrix would append/add ">" 1

0 times and append/add “<” 10 times after that. The constraint, sense, and RHS matrices would all be expanded by 20 rows. This would create constraints that effectively create a lower bound of $1M less than last month’s spend and an upper bound of $1M more than last month’s spend in addition to the default $0M lower bound and $3M upper bound to enforce stability.

Brute-Force: To add this constraint in a one-go brute-force optimization problem, a set of 120 (months * channels) variables with each channel’s spend in each month would need to be created. And 220 ((months - 1) * channels * 2) constraints would need to be added; 22 for each type of channel. The additional constraints would subtract the spend on that medium from the prior month’s spend on that medium (using a -1, 1, and filler zeros; January excluded), and set the sense and RHS to < 1 in 11 constraints for each month and > -1 in the other 11 for each month. This would create custom individual constraints for Feb-Dec for all channels for both an increase or decrease in ad spend to enforce stability.