Marketing-Analytics-Projeto

January 1, 2022

1 Business Analytics

2 Marketing Analytics

3 Projeto

3.1 Otimização de Preços e Mix de Produtos

```
In [1]: # Versão da Linguagem Python
        from platform import python_version
        print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
```

```
Versão da Linguagem Python Usada Neste Jupyter Notebook: 3.7.6
```

3.2 Marketing Analytics

Marketing Analytics compreende os processos e tecnologias que permitem aos profissionais de Marketing avaliar o sucesso de suas iniciativas.

Isso é feito medindo o desempenho das campanhas de Marketing, coletando os dados e analisando os resultados. Marketing Analytics utiliza métricas importantes de negócios, como ROI (Retorno Sobre o Investimento), Atribuição de Marketing e Eficácia Geral do Marketing. Em outras palavras, o Marketing Analytics mostra se os programas de Marketing estão sendo efetivos ou não.

Marketing Analytics reúne dados de todos os canais de marketing e os consolida em uma visão de marketing comum. A partir dessa visão comum, você pode extrair resultados analíticos que podem fornecer assistência inestimável para impulsionar os esforços de marketing.

3.3 Definindo o Problema

3.4 Lua Tech

A empresa Lua Smart Tech monta e testa dois modelos de smartphones, Lua1 e Lua2.

Para o próximo mês, a empresa quer decidir quantas unidades de cada modelo vai montar e depois testar.

Nenhum smartphone está em estoque desde o mês anterior e, como esses modelos serão trocados depois deste mês, a empresa não quer manter nenhum estoque para o mês seguinte.

title

Ela acredita que o máximo que pode vender neste mês são 600 unidades do modelo Lua1 e 1200 unidades do modelo Lua2. Cada modelo Lua1 é vendido por R$300 e cada modelo Lua2 por R$450.

O custo dos componentes de um Lua1 é de R$150 e para um Lua2 é R$225. A mão de obra é necessária para a montagem e teste. Existem no máximo 10.000 horas de montagem e 3.000 horas de teste disponíveis.

Cada hora de trabalho para montagem custa R$11 e cada hora de trabalho para teste custa R$15. Cada Lua1 requer cinco horas para montagem e uma hora para teste.

Cada Lua2 requer seis horas para montagem e duas horas para teste.

A Lua Smart Tech deseja saber quantas unidades de cada modelo deve produzir (montar e testar) para maximizar seu lucro líquido, mas não pode usar mais horas de trabalho do que as disponíveis e não deseja produzir mais do que pode vender.

O trabalho é realizar a otimização de preços e mix dos produtos da Lua Smart Tech.

3.5 Instalando e Carregando os Pacotes

```
In [2]: # Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de comando:
        # pip install -U nome_pacote

        # Para instalar a versão exata de um pacote, execute o comando abaixo no terminal ou p
        # !pip install nome_pacote==versão_desejada

        # Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.

        # Instala o pacote watermark.
        # Esse pacote é usado para gravar as versões de outros pacotes usados neste jupyter no
        #!pip install -q -U watermark
```

```
In [1]: # Instala o PuLP
        # https://pypi.org/project/PuLP/
```

```
        # https://coin-or.github.io/pulp/
        !pip install -q pulp

In [2]: # Imports
        from pulp import *

In [3]: # Versões dos pacotes usados neste jupyter notebook
        %reload_ext watermark
        %watermark -a "João Souza" --iversions

Author: João Souza

sys    : 3.7.3 (default, Mar 27 2019, 22:11:17)
[GCC 7.3.0]
re     : 2.2.1
pulp   : 2.6.0
ctypes : 1.1.0
json   : 2.0.9
logging: 0.5.1.2
```

3.6 Criando o Modelo Matemático Para a Otimização

3.6.1 Parâmetros

- Ai = Número máximo de smartphones modelo tipo i para vender este mês, onde i pertence ao conjunto {Lua1, Lua2}
- Bi = Preço de venda de smartphones modelo tipo i, onde i pertence ao conjunto {Lua1, Lua2}
- Ci = Preço de custo das peças componentes para smartphones modelo tipo i, onde i pertence ao conjunto {Lua1, Lua2}
- Di = Custo de mão de obra de montagem por hora de smartphones modelo tipo i, onde i pertence ao conjunto {Lua1, Lua2}
- Ei = Custo de mão de obra de teste por hora de smartphones modelo tipo i, onde i pertence ao conjunto {Lua1, Lua2}
- F = Número máximo de horas de trabalho de montagem
- G = Número máximo de horas de trabalho de teste
- Hf,i = Horas de montagem necessárias para construir cada modelo de smartphone tipo i, onde i pertence ao conjunto {Lua1, Lua2}
- Hg,i = Horas de teste necessárias para testar cada modelo de smartphone tipo i, onde i pertence ao conjunto {Lua1, Lua2}

3.6.2 Variável de Decisão

- Xi = Número de smartphones modelo tipo i a produzir este mês, onde i pertence ao conjunto {Lua1, Lua2}

$$\text{LucroTotal} = \sum (X_i * B_i) - \sum (X_i * H_{f,i} * D_i) - \sum (X_i * H_{g,i} * E_i) - \sum (X_i * C_i)$$

title

3.6.3 Função Objetivo

3.6.4 Restrições

- O número de smartphones modelo tipo i a serem produzidos não pode ser negativo, ou seja, Xi >= 0, onde i pertence ao conjunto {Lua1, Lua2}.
- O número total de horas de montagem não pode ser maior que o número máximo de horas de mão de obra de montagem disponíveis.
- O número total de horas de teste não pode ser maior do que o máximo de horas de mão de obra de teste disponíveis.
- O número de smartphones modelo tipo i a serem produzidos não pode ser maior do que o número máximo de smartphones modelo tipo i a serem vendidos neste mês, onde i pertence ao conjunto {Lua1, Lua2}.

3.7 Implementando o Modelo Matemático

3.7.1 Organizando os Parâmetros

```
In [4]: # Número máximo de smartphones para vender este mês
        A_Lua1 = 600
        A_Lua2 = 1200

In [5]: # Preço de venda de smartphones
        B_Lua1 = 300
        B_Lua2 = 450

In [6]: # Preço de custo das peças componentes para smartphones
        C_Lua1 = 150
        C_Lua2 = 225

In [7]: # Custo de mão de obra de montagem por hora de smartphones
        D_Lua1 = 11
        D_Lua2 = 11

In [8]: # Custo de mão de obra de teste por hora de smartphones
        E_Lua1 = 15
        E_Lua2 = 15

In [9]: # Número máximo de horas de trabalho de montagem
        F = 10000
```

```
In [10]: # Número máximo de horas de trabalho de teste
         G = 3000

In [11]: # Horas de montagem necessárias para construir cada modelo de smartphone
         Hfi_Lua1 = 5
         Hfi_Lua2 = 6

In [12]: # Horas de teste necessárias para testar cada modelo de smartphone
         Hgi_Lua1 = 1
         Hgi_Lua2 = 2
```

3.7.2 Criação da Variável Para o Problema de Otimização

```
In [15]: help(LpProblem)

Help on class LpProblem in module pulp.pulp:

class LpProblem(builtins.object)
 |  LpProblem(name='NoName', sense=1)
 |
 |  An LP Problem
 |
 |  Methods defined here:
 |
 |  __getstate__(self)
 |
 |  __iadd__(self, other)
 |
 |  __init__(self, name='NoName', sense=1)
 |      Creates an LP Problem
 |
 |      This function creates a new LP Problem  with the specified associated parameters
 |
 |      :param name: name of the problem used in the output .lp file
 |      :param sense: of the LP problem objective.                  Either :data:`~pulp.const.
 |      :return: An LP Problem
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  add(self, constraint, name=None)
 |
 |  addConstraint(self, constraint, name=None)
 |
 |  addVariable(self, variable)
 |      Adds a variable to the problem before a constraint is added
 |
```

```
|      @param variable: the variable to be added
|
|  addVariables(self, variables)
|      Adds variables to the problem before a constraint is added
|
|      @param variables: the variables to be added
|
|  assignConsPi(self, values)
|
|  assignConsSlack(self, values, activity=False)
|
|  assignStatus(self, status, sol_status=None)
|      Sets the status of the model after solving.
|      :param status: code for the status of the model
|      :param sol_status: code for the status of the solution
|      :return:
|
|  assignVarsDj(self, values)
|
|  assignVarsVals(self, values)
|
|  checkDuplicateVars(self)
|      Checks if there are at least two variables with the same name
|      :return: 1
|      :raises `const.PulpError`: if there ar duplicates
|
|  checkLengthVars(self, max_length)
|      Checks if variables have names smaller than `max_length`
|      :param int max_length: max size for variable name
|      :return:
|      :raises const.PulpError: if there is at least one variable that has a long name
|
|  coefficients(self, translation=None)
|
|  copy(self)
|      Make a copy of self. Expressions are copied by reference
|
|  deepcopy(self)
|      Make a copy of self. Expressions are copied by value
|
|  extend(self, other, use_objective=True)
|      extends an LpProblem by adding constraints either from a dictionary
|      a tuple or another LpProblem object.
|
|      @param use_objective: determines whether the objective is imported from
|      the other problem
|
|      For dictionaries the constraints will be named with the keys
```

```
|      For tuples an unique name will be generated
|      For LpProblems the name of the problem will be added to the constraints
|      name
|
|  fixObjective(self)
|
|  getSense(self)
|
|  get_dummyVar(self)
|
|  infeasibilityGap(self, mip=1)
|
|  isMIP(self)
|
|  normalisedNames(self)
|
|  numConstraints(self)
|      :return: number of constraints in model
|
|  numVariables(self)
|      :return: number of variables in model
|
|  resolve(self, solver=None, **kwargs)
|      resolves an Problem using the same solver as previously
|
|  restoreObjective(self, wasNone, dummyVar)
|
|  roundSolution(self, epsInt=1e-05, eps=1e-07)
|      Rounds the lp variables
|
|      Inputs:
|          - none
|
|      Side Effects:
|          - The lp variables are rounded
|
|  sequentialSolve(self, objectives, absoluteTols=None, relativeTols=None, solver=None, debug=
|      Solve the given Lp problem with several objective functions.
|
|      This function sequentially changes the objective of the problem
|      and then adds the objective function as a constraint
|
|      :param objectives: the list of objectives to be used to solve the problem
|      :param absoluteTols: the list of absolute tolerances to be applied to
|         the constraints should be +ve for a minimise objective
|      :param relativeTols: the list of relative tolerances applied to the constraints
|      :param solver: the specific solver to be used, defaults to the default solver.
|
```

```
|  setObjective(self, obj)
|      Sets the input variable as the objective function. Used in Columnwise Modelling
|
|      :param obj: the objective function of type :class:`LpConstraintVar`
|
|      Side Effects:
|          - The objective function is set
|
|  setSolver(self, solver=<pulp.apis.coin_api.PULP_CBC_CMD object at 0x7fa716576450>)
|      Sets the Solver for this problem useful if you are using
|      resolve
|
|  solve(self, solver=None, **kwargs)
|      Solve the given Lp problem.
|
|      This function changes the problem to make it suitable for solving
|      then calls the solver.actualSolve() method to find the solution
|
|      :param solver:  Optional: the specific solver to be used, defaults to the
|            default solver.
|
|      Side Effects:
|          - The attributes of the problem object are changed in
|            :meth:`~pulp.solver.LpSolver.actualSolve()` to reflect the Lp solution
|
|  toDict(self)
|      creates a dictionary from the model with as much data as possible.
|      It replaces variables by variable names.
|      So it requires to have unique names for variables.
|
|      :return: dictionary with model data
|      :rtype: dict
|
|  toJson(self, filename, *args, **kwargs)
|      Creates a json file from the LpProblem information
|
|      :param str filename: filename to write json
|      :param args: additional arguments for json function
|      :param kwargs: additional keyword arguments for json function
|      :return: None
|
|  to_dict = toDict(self)
|
|  to_json = toJson(self, filename, *args, **kwargs)
|
|  unusedConstraintName(self)
|
|  valid(self, eps=0)
```

```
|
|  variables(self)
|      Returns a list of the problem variables
|
|      Inputs:
|          - none
|
|      Returns:
|          - A list of the problem variables
|
|  variablesDict(self)
|
|  writeLP(self, filename, writeSOS=1, mip=1, max_length=100)
|      Write the given Lp problem to a .lp file.
|
|      This function writes the specifications (objective function,
|      constraints, variables) of the defined Lp problem to a file.
|
|      :param str filename: the name of the file to be created.
|      :return: variables
|      Side Effects:
|          - The file is created
|
|  writeMPS(self, filename, mpsSense=0, rename=0, mip=1)
|      Writes an mps files from the problem information
|
|      :param str filename: name of the file to write
|      :param int mpsSense:
|      :param bool rename: if True, normalized names are used for variables and constraints
|      :param mip: variables and variable renames
|      :return:
|      Side Effects:
|          - The file is created
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  fromDict(_dict) from builtins.type
|      Takes a dictionary with all necessary information to build a model.
|      And returns a dictionary of variables and a problem object
|
|      :param _dict: dictionary with the model stored
|      :return: a tuple with a dictionary of variables and a :py:class:`LpProblem`
|
|  fromJson(filename) from builtins.type
|      Creates a new Lp Problem from a json file with information
|
|      :param str filename: json file name
```

```
 |      :return: a tuple with a dictionary of variables and an LpProblem
 |      :rtype: (dict, :py:class:`LpProblem`)
 |
 |  from_dict = fromDict(_dict) from builtins.type
 |      Takes a dictionary with all necessary information to build a model.
 |      And returns a dictionary of variables and a problem object
 |
 |      :param _dict: dictionary with the model stored
 |      :return: a tuple with a dictionary of variables and a :py:class:`LpProblem`
 |
 |  from_json = fromJson(filename) from builtins.type
 |      Creates a new Lp Problem from a json file with information
 |
 |      :param str filename: json file name
 |      :return: a tuple with a dictionary of variables and an LpProblem
 |      :rtype: (dict, :py:class:`LpProblem`)
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  MPS_bound_lines(name, variable, mip)
 |
 |  MPS_column_lines(cv, variable, mip, name, cobj, objName)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

```
In [14]: # Variável para o problema
         problema = LpProblem("MixProdutos", LpMaximize)

In [15]: # Visualiza
         problema

Out[15]: MixProdutos:
         MAXIMIZE
         None
         VARIABLES
```

3.7.3 Definindo a Variável de Decisão de Cada Modelo de Smartphone

```
In [16]: #help(LpVariable)
```

$$\text{LucroTotal} = \sum (Xi * Bi) - \sum (Xi * Hf,i * Di) - \sum (Xi * Hg,i * Ei) - \sum (Xi * Ci)$$

title

```
In [17]: # Define as variáveis
         x_Lua1 = LpVariable("Unidades Lua1", 0, None, LpInteger)
         x_Lua2 = LpVariable("Unidades Lua2", 0, None, LpInteger)

In [18]: # Print
         print(x_Lua1)
         print(x_Lua2)

Unidades_Lua1
Unidades_Lua2
```

3.7.4 Implementando a Função Objetivo

```
In [19]: # Faturamento -> unidades vendidas x valor de venda
         faturamento = (x_Lua1 * B_Lua1) + (x_Lua2 * B_Lua2)
         faturamento

Out[19]: 300*Unidades_Lua1 + 450*Unidades_Lua2 + 0

In [20]: # Custo de Montagem
         custo_montagem = (x_Lua1 * Hfi_Lua1 * D_Lua1) + (x_Lua2 * Hfi_Lua2 * D_Lua2)
         custo_montagem

Out[20]: 55*Unidades_Lua1 + 66*Unidades_Lua2 + 0

In [21]: # Custo de Teste
         custo_teste = (x_Lua1 * Hgi_Lua1 * E_Lua1) + (x_Lua2 * Hgi_Lua2 * E_Lua2)
         custo_teste

Out[21]: 15*Unidades_Lua1 + 30*Unidades_Lua2 + 0

In [22]: # Custo de Componentes
         custo_componentes = (x_Lua1 * C_Lua1) + (x_Lua2 * C_Lua2)
         custo_componentes

Out[22]: 150*Unidades_Lua1 + 225*Unidades_Lua2 + 0

In [23]: # Lucro = Faturamento - Custo de Montagem - Custo de Teste - Custo de Componentes
         problema += faturamento - custo_montagem - custo_teste - custo_componentes
         problema
```

```
Out[23]: MixProdutos:
         MAXIMIZE
         80*Unidades_Lua1 + 129*Unidades_Lua2 + 0
         VARIABLES
         0 <= Unidades_Lua1 Integer
         0 <= Unidades_Lua2 Integer
```

3.7.5 Restrições

```
In [24]: # Número máximo de horas de montagem
         problema += (x_Lua1 * Hfi_Lua1) + (x_Lua2 * Hfi_Lua2) <= F,"Número Máximo de Horas de

In [25]: # Número máximo de horas de teste
         problema += (x_Lua1 * Hgi_Lua1) + (x_Lua2 * Hgi_Lua2) <= G,"Número Máximo de Horas de

In [26]: # Produção menor ou igual a demanda pelo modelo Lua1
         problema += x_Lua1 <= A_Lua1,"Produção menor ou igual a demanda pelo modelo Lua1"

In [27]: # Produção menor ou igual a demanda pelo modelo Lua2
         problema += x_Lua2 <= A_Lua2,"Produção menor ou igual a demanda pelo modelo Lua2"

In [28]: # Problema final
         problema

Out[28]: MixProdutos:
         MAXIMIZE
         80*Unidades_Lua1 + 129*Unidades_Lua2 + 0
         SUBJECT TO
         Número_Máximo_de_Horas_de_Montagem: 5 Unidades_Lua1 + 6 Unidades_Lua2 <= 10000

         Número_Máximo_de_Horas_de_Teste: Unidades_Lua1 + 2 Unidades_Lua2 <= 3000

         Produção_menor_ou_igual_a_demanda_pelo_modelo_Lua1: Unidades_Lua1 <= 600

         Produção_menor_ou_igual_a_demanda_pelo_modelo_Lua2: Unidades_Lua2 <= 1200

         VARIABLES
         0 <= Unidades_Lua1 Integer
         0 <= Unidades_Lua2 Integer
```

3.7.6 Resolvendo o Problema de Otimização

```
In [29]: # Otimização
         problema.solve()

Out[29]: 1

In [30]: # Lucro Maximizado
         print("Lucro Maximizado:", value(problema.objective))
```

```
Lucro Maximizado: 199600.0
```

```
In [33]: # Número de Unidades do Modelo Lua 1 a Produzir
         print("Número de Unidades do Modelo Lua 1 a Produzir:", problema.variables()[0].varVa
```

```
Número de Unidades do Modelo Lua 1 a Produzir: 560.0
```

```
In [31]: # Número de Unidades do Modelo Lua 1 a Produzir
         print("Número de Unidades do Modelo Lua 2 a Produzir:", problema.variables()[1].varVa
```

```
Número de Unidades do Modelo Lua 2 a Produzir: 1200.0
```

3.8 Conclusão

A empresa Lua Smart Tech deve produzir 560 unidades do tipo de smartphone Lua1 e 1200 do tipo Lua2 para atingir o lucro máximo de R$199.600.

4 Fim