upGrad

Operations Research - Lecture notes

In this module, you solved three optimisation problems using Pyomo. Let's summarise the problem statement, the mathematical model and the Pyomo model for each of these problem statements.

Case Study I – Airlines Optimisation

Problem Statement

FlyIndia, a fictional airline company, has to cater to two types of customer segments – the 'early birds' and the 'late buyers'. 'Early birds' are the customers who would be buying tickets much in advance and, hence, are eligible for a discount. 'Late buyers' are the customers who would be buying the tickets at the regular price. The airline company wants to maximise its revenue for a particular Delhi to Bangalore flight by allocating the regular and discounted seats judiciously. One hundred and sixty-six seats are available in the aircraft. Based on the past observations, the demand for regular tickets does not exceed 100 while the demand for discounted tickets does not exceed 150. The prices of the tickets are as follows:

Discounted ticket: ₹1,190

Regular (non-discounted) ticket: ₹3,085

Mathematical Formulation

Suppose 'x' is the decision variable representing the number of seats allocated to the regular/late buyers segment and 'y' is the decision variable representing the number of seats allocated to the early birds segment. In this case, the objective function becomes **Max(z) = 3085x + 1190y**.

The **objective function** is subject to the following constraints:

- The total number of seats to be allocated: **$x+y \leq 166$**
- Seats allocated at the regular price: **$x \leq 100$**
- Seats allocated at the discounted price: **$y \leq 150$**

Non-negativity constraints, i.e., the number of seats allocated at the regular or the discounted price cannot be negative.

- Seats allocated at regular price: **$x \geq 0$**

- Seats allocated at discounted price: **y≥ 0**

Pyomo Modelling

```
# Ask Python to load the Pyomo modelling environment

from pyomo.environ import *
```

```
# Indexes - Defining the sets/lists containing the indexes for this problem. The type of ticket becomes the index here

# Early bird- 'eb', Regular - 'reg'

tkt_types=['eb','reg']
```

```
# parameters - The price of each ticket type is defined as a Python dictionary

tkt_price={'eb':1190,'reg':3085}
```

```
# Creating an instance of a Concrete model since we have all the required data beforehand

model = ConcreteModel()
```

```
#variables - Decision variable X has the index as tkt_types

model.X=Var(tkt_types,within=PositiveIntegers)
```

```
#Objective - Maximise the total revenue
```

upGrad

```
model.value = Objective(expr = sum(tkt_price[t]*model.X[t] for t in tkt_types),
sense=maximize)
```

```
#constraints

#expr method

#Regular demand does not exceed 100

model.reg_demand=Constraint(expr=model.X['reg']<=100)

#Discount demand doesn't exceed 150
model.dis_demand=Constraint(expr=model.X['eb']<=150)


#rule method

#Maximum seats available is 166

def supply_constraint(model):
    return(sum(model.X[i] for i in tkt_types)==166)

model.supply=Constraint(rule=supply_constraint)
```

```
# Invoking the solver

result = SolverFactory('glpk').solve(model)
result.write()
```

The results from the solver are shown in the following image:

```
result = SolverFactory('glpk').solve(model)
result.write()
```

```
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: 387040.0
  Upper bound: 387040.0
  Number of objectives: 1
  Number of constraints: 4
  Number of variables: 3
  Number of nonzeros: 5
  Sense: maximize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Termination condition: optimal
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 1
      Number of created subproblems: 1
  Error rc: 0
  Time: 2.2100868225097656
# ----------------------------------------------------------
```

The optimised value of the objective function is ₹3,87,040. The upper bound and the lower bound matches because the solution has converged. There are cases where the objective function can take an infinite set of values without violating the given constraints. In such cases, the values of the upper and the lower bounds will be different. You can read more about it in this link. The value of the objective function can also be obtained using model.value().

Although only three constraints (two demand and one supply constraints) were defined initially, restricting the decision variable to integer values has also been counted as another constraint. Hence, there are a total of four constraints.

Only two decision variables (types of tickets) were defined. However, the solver shows three decision variables because it considers the value of the objective function as another variable.

model.pprint() gives more granular information about the model. It displays all the modelling components, including indexes, parameters, constraints and objective functions, in a single frame, as shown in the following image, which says that 100 seats have to be allocated to the early birds segment and 66 seats have to be allocated to the regular segment.

```
1 Set Declarations
    X_index : Size=1, Index=None, Ordered=Insertion
        Key  : Dimen : Domain : Size : Members
        None :     1 :    Any :    2 : {'eb', 'reg'}

1 Var Declarations
    X : Size=2, Index=X_index
        Key : Lower : Value : Upper : Fixed : Stale : Domain
         eb :     1 :  66.0 :  None : False : False : PositiveIntegers
        reg :     1 : 100.0 :  None : False : False : PositiveIntegers

1 Objective Declarations
    value : Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
        None :   True : maximize : 1190*X[eb] + 3085*X[reg]

3 Constraint Declarations
    dis_demand : Size=1, Index=None, Active=True
        Key  : Lower : Body  : Upper : Active
        None :  -Inf : X[eb] : 150.0 :   True
    reg_demand : Size=1, Index=None, Active=True
        Key  : Lower : Body   : Upper : Active
        None :  -Inf : X[reg] : 100.0 :   True
    supply : Size=1, Index=None, Active=True
        Key  : Lower : Body          : Upper : Active
        None : 166.0 : X[eb] + X[reg] : 166.0 :   True

6 Declarations: X_index X value reg_demand dis_demand supply
```

Case Study II – Warehouse Problem

Problem Statement

BBasket is an Indian corporation that operates a chain of membership-only grocery clubs. Currently, it has thousands of customers in the following locations – **Bengaluru, Chennai, Hyderabad** and **Pune**. The company wants to open **two** warehouses from the given list of locations made available – **Anantapur, Mysore** and **Vellore**. As an analyst, your task is to provide the company with a solution to have the minimum total cost of serving all the customers with a given set of constraints.

Only one active warehouse can serve a customer location. The transportation distance between a particular customer from the warehouse location is given in the following table.

'W' indicates warehouse locations and 'C' refers to customer locations.

W/C	Hyd	Chennai	Bangalore	Pune
Anantapur	422	482	215	882
Mysore	797	482	144	934
Vellore	779	157	201	1138

Note: Only one active warehouse can serve a customer location. Now, your objective is to use the concepts of operations research to determine the set of warehouses from the set 'W' of candidates to minimise the transportation cost of serving all the customers in the set 'C'.

Mathematical Formulation

The model formulation for the given problem statement is as follows:

Indexes

The variables and parameters for the given problem vary with warehouse and customer locations. Hence, the indexes are defined as follows:

- Warehouse location: $w \in W$
- Customer location: $c \in C$

The symbol '$\in$' in $w \in W$ means that w is an element of the set W.

Parameters

$d_{w,c}$: This is the transportation distance between customer ‘c’ from the warehouse location ‘w’.

Decision Variables

$x_{w,c}$: This binary variable indicates whether the warehouse ‘w’ serves customer ‘c’ or not (0,1).

y_w: This binary variable indicates whether the warehouse ‘w’ has been selected or not (0,1).

Objective Function

$$\textbf{Minimise} \sum_{w \in W, c \in C} d_{w,c} * x_{w,c}$$

Here, the product of $d_{w,c}$ and $x_{w,c}$ or $d_{w,c}x_{w,c}$ indicates the distance between a customer location ‘c’ and a warehouse ‘w’. When this is summed across warehouses ($\forall w \in W$) and for all 'c' in the customer locations set 'C' ($\forall c \in C$), the final result would indicate the total transportation distance for serving all the customer demands. This is the expression that you must intend to minimise in this optimisation problem. The cost of transportation per unit distance is assumed to be the same for all warehouse-customer locations. Therefore, distance minimisation would lead to cost minimisation.

Here is a more straightforward way to understand the equation. $\sum d_{w,c}$ allows you to add all possible warehouse-customer distances. Multiplying it with a boolean factor $x_{w,c}$ ensures that you only add the distances for which the warehouse 'w' serves the customer 'c'. Then, you try to find the minimum of all possible summations of distances.

Constraints

Constraint 1: Only one warehouse serves a single customer site and all the customer demands are met. Let's understand this with the help of an example.

As you know, $x_{w,c}$ represents whether a warehouse 'w' serves a customer 'c' or not. Suppose the customer site Hyderabad (= c) is served by Anantpur (= w), then $x_{Anantapur,Hyd}=1$. In this case, the entire demand of Hyderabad should be served only by the Anantapur warehouse. Hence, $x_{w,Hyd}$ for all the other warehouses except Anantapur will be 0. Therefore,

$x_{Anantapur,Hyd}+x_{Vellore,Hyd}+x_{Mysore,Hyd} = 1+0+0=1$ and the generalised equation can be written as follows:

$$\sum_{w \in W} x_{w,c}=1 \quad \forall c \in C$$

Constraint 2: Customer 'c' can only be served from the warehouse 'w' if the warehouse 'w' is selected. Let's understand this with the help of an example. Consider all the given possible scenarios.

Case 1:

If the warehouse in Mysore is not selected ($y_{Mysore}=0$), then the demand of no customer(c) can be met by the warehouse in Mysore. Hence, $x_{Mysore,c}=0=y_{Mysore}$ for all customer sites c in C.

Case 2:

If the warehouse in Mysore is open ($y_{Mysore}=1$), then $x_{Mysore,c}$ $\forall c \in C$ can take either 0 or 1. Hence, in this case, x_(Mysore,c) <= y_Mysore for all customer sites c in C.

Therefore, the equation that considers both the mentioned possibilities is as follows:

$$x_{w,c} \leq y_w \quad \forall c \in C, w \in W$$

Constraint 3: The budget allows us to build only a fixed number of warehouses (P). For this problem, P = 2.

$$\sum_{w \in W} y_w \leq P$$

Let's take an example to understand this equation. If the warehouses at Anantapur and Vellore are open ($y_{Anantapur}=1$, $y_{Vellore}=1$), then the warehouse at Mysore should not be open. Hence, $y_{Mysore}=0$.

Pyomo Modelling

```
#Importing Pyomo environment and other required libraries

from pyomo.environ import *

import pandas as pd
```

```
# Indexes - Defining sets/lists containing the indexes for this problem. This is a two index problem; the warehouse location and customer location will be the indexes

W = ['Anantpur','Mysore','Vellore']
C = ['Hyd','Chennai','Bangalore','Pune']
```

upGrad

```
# parameters - Distance between each warehouse-customer location

d = {
    ('Anantpur','Hyd') : 422,
    ('Anantpur','Chennai') : 482,
    ('Anantpur','Bangalore') : 215,
    ('Anantpur','Pune') : 882,
    ('Mysore','Hyd') : 797,
    ('Mysore','Chennai') : 482,
    ('Mysore','Bangalore') : 144,
    ('Mysore','Pune') : 934,
    ('Vellore','Hyd') : 779,
    ('Vellore','Chennai') : 157,
    ('Vellore','Bangalore') : 201,
    ('Vellore','Pune') : 1138
}

# Maximum number of warehouses that are allowed to be open
P = 2
```

```
# Creating an instance of a Concrete model since we have all the required data beforehand

model = ConcreteModel()
```

```
#Defining decision variables

model.x = Var(W,C,bounds = (0,1))
model.y = Var(W,within = Binary)
```

```
# Objective - To minimise the distance between warehouse and customer location

def obj_rule(m):
    return sum(d[w,c]*model.x[w,c] for w in W for c in C)
```

```
model.obj = Objective(rule = obj_rule)
```

```
# Constraint 1 - Single warehouse per customer

def warehouse_per_cust(model,c):
    return sum(model.x[w,c] for w in W) == 1

model.cust_warehouse_one = Constraint(C,rule = warehouse_per_cust)
```

```
#Constraint 2 - Only an active warehouse can serve a customer location

def active_warehouse_rule(model,w,c):
    return model.x[w,c] <= model.y[w]

model.is_warehouse_active = Constraint(W,C,rule = active_warehouse_rule)
```

```
#Constraint 3 - Number of warehouses cannot be more than what is required

def num_warehouses_rule(m):
    return sum(model.y[w] for w in W) <= P

model.num_warehouses = Constraint(rule = num_warehouses_rule)
```

```
#Invoking the solver

SolverFactory('glpk').solve(model)
model.pprint()
```

The solver results are shown in the following image:

```
2 Var Declarations
    x : Size=12, Index=x_index
        Key                       : Lower : Value : Upper : Fixed : Stale : Domain
        ('Anantpur', 'Bangalore') :     0 :   0.0 :     1 : False : False :  Reals
          ('Anantpur', 'Chennai') :     0 :   0.0 :     1 : False : False :  Reals
              ('Anantpur', 'Hyd') :     0 :   1.0 :     1 : False : False :  Reals
             ('Anantpur', 'Pune') :     0 :   1.0 :     1 : False : False :  Reals
          ('Mysore', 'Bangalore') :     0 :   0.0 :     1 : False : False :  Reals
            ('Mysore', 'Chennai') :     0 :   0.0 :     1 : False : False :  Reals
                ('Mysore', 'Hyd') :     0 :   0.0 :     1 : False : False :  Reals
               ('Mysore', 'Pune') :     0 :   0.0 :     1 : False : False :  Reals
         ('Vellore', 'Bangalore') :     0 :   1.0 :     1 : False : False :  Reals
           ('Vellore', 'Chennai') :     0 :   1.0 :     1 : False : False :  Reals
               ('Vellore', 'Hyd') :     0 :   0.0 :     1 : False : False :  Reals
              ('Vellore', 'Pune') :     0 :   0.0 :     1 : False : False :  Reals
    y : Size=3, Index=y_index
        Key      : Lower : Value : Upper : Fixed : Stale : Domain
        Anantpur :     0 :   1.0 :     1 : False : False : Binary
          Mysore :     0 :   0.0 :     1 : False : False : Binary
         Vellore :     0 :   1.0 :     1 : False : False : Binary
```

The results show that **Anantapur** serves **Hyderabad** and **Pune's** customers, and **Vellore** serves **Bengaluru** and **Chennai's** customers.

Now, to determine the value of the objective function, model.value() will return **1662 km**.

upGrad

Case Study III – Bank Marketing Case Study

Problem Statement

- In 2017, the 'Bank of Corporate' conducted a telemarketing campaign for one of its products 'Term Deposits'.
- The aim was to identify the target customer segments for term deposits from the pool of existing customers.
- The bank has allocated a budget of ₹150,000 and has decided to segment customers based on their marital status and educational background.
- The bank incurs a cost of ₹10 for a 1 minute call.
- The objective is to optimise the number of calls to be made to each customer segment such that the total number of customers opening term deposit is maximised.
- We had the following customer segments as shown in the following.

Customer Segment

Marital Status	Educational Background
Single	Bachelors
Married	Masters
Divorced	Doctorate

'Single - Bachelors' is considered as one segment. 'Single - Masters' is considered as one segment. Similarly, 'Married - Masters' is considered as one segment, 'Married - Doctorates' is considered as one segment and so on.

The bank is concerned about the overall customer diversification. It wants to ensure that it reaches out to all the customer segments. For this, it has provided you with the following information to include in your analysis:

- At least 50 customers need to be contacted from each customer segment.
- The total number of calls made to each customer category should meet the minimum number of calls as mentioned in the following table:

Bachelors	400
Masters	500
Doctorate	600
Married	600
Single	300
Divorced	350

- The total number of conversions of the following customer categories should match a minimum number as mentioned in the following table:

Bachelors	120
Masters	120
Doctorate	120
Married	150
Single	150
Divorced	100

The indexes and parameters of the problem can be defined as follows.

Mathematical Formulation

Sets:

Since the decision variables and the parameters change with respect to the marital status and educational degree, the indexes for the given problem are as follows:

- Marital status: $m \in$ marital_status
- Degree status: $d \in$ degree

Parameters:

- duration_nconverted$_{m,d}$: Average call duration for non-converted
- duration_converted$_{m,d}$: Average call duration for converted
- min_tcalls?|?: Minimum number of calls either by marital status or degree
- min_ccalls?|?: Minimum number of converted calls either by marital status or degree

- The objective of this problem is to **maximise the total number of converted calls** intuitively. It will be a function of the number of calls made and the conversion rate of each segment.

- Since the conversion rate for each segment is assumed to be constant as per the historical data, the only thing that you can control is the total number of calls (converted + non-converted) to be made for each customer segment.
- Hence, the **decision variable** becomes:

$$\textbf{total\_calls}_{m,d} \textbf{ where, } m \in \textbf{marital\_status}, d \in \textbf{degree}$$

- The **objective function** becomes:

$$\textbf{Maximum} \sum_{m,d} \textbf{total\_calls}_{m,d} * \textbf{conv\_rate}_{m,d}$$

where, $\mathbf{m \in marital\_status, d \in degree}$

upGrad

Constraints:

- At-least 50 customers need to be contacted from each segment
  - $\text{total\_calls}_{m,d} \geq 50 \quad \forall d \in degree, m \in marital\_status$
- Minimum number of total calls for each marital status
  - $\sum_{d} \text{total\_calls}_{m,d} \geq \text{min\_tcalls}_{m}$
- Minimum number of total calls for each degree
  - $\sum_{m} \text{total\_calls}_{m,d} \geq \text{min\_tcalls}_{d}$
- Minimum number of converted calls for each marital status
  - $\sum_{d} (\text{total\_calls}_{m,d} * \text{conv\_rate}_{m,d}) \geq \text{min\_tcalls}_{m}$
- Minimum number of converted calls for each degree
  - $\sum_{m} (\text{total\_calls}_{m,d} * \text{conv\_rate}_{m,d}) \geq \text{min\_tcalls}_{d}$

- Budget constraint

$\sum m,d$ **[total_calls**m,d $*$ **conv_rate**m,d $*$ **duration_converted**m,d**/60** $*$ **cost_per_min**

[total_callsm,d $*$ **(1- conv_rate**m,d**)** $*$ **Duration_nconverted**m,d**/60** $*$ **cost_per_min]**

≤total_budget

The pseudocode for the constraints can be written as shown in the following images.

```
model.total_calls = Var(M_status, Degree,
within=PositiveIntegers)
```

$converted\_calls_{m,d}$

Defining variable

For each marital_status:

For each degree :

```
marital_status,degree = minimum 50 total calls
```

Min. Calls per Segment

$$\boxed{total\_calls_{i,j} \geq 50 \quad \forall\, i \in m, j \in d}$$

For each marital_status:

```
sum(total_calls) for each degree > min. total calls for each
marital_status
```

Min. calls by marital status

$$\boxed{\sum_{d} total\_calls_{m,d} \geq \min\_tc_{m}}$$

For each degree:

```
sum(converted_calls) for each marital_status >=
min. total calls for each degree
```

Min. calls by Degree

$$\boxed{\sum_{m} total\_calls_{m,d} \geq \min\_tc_{d}}$$

For each marital_status:

```
sum(total_calls * conv_rate) for each degree >=
min. converted calls for each marital_status
```

By marital segment min converted calls

$$\boxed{\sum_{d} conv\_rate_{m,d} * total\_calls_{m,d} \geq \min\_cc_{m}}$$

By degree segment min. converted calls

For each degree:

```
sum(conv_rate * total_calls) for each marital_status) >=
min. converted calls
```

$$\sum_{m} conv\_rate_{m,d} * total\_calls_{m,d} \geq \min\_cc_{d}$$

Constraint of Maximum Budget

For each marital_status:
For each degree:

```
sum[cost_per_min * total_calls * (dur_converted * conv_rate + dur_nconverted *
{1-conv_rate})]
```

$$\sum_{m,d} \left(duration\_c_{m,d} * conv_{rate_{m,d}} + duration_{nc_{m,d}} * (1 - conv_{rate_{m,d}})\right) * cost\_per\_min * total\_calls_{m,d}$$

Objective function to max. number of calls

For each marital_status:
For each degree:

```
maximum[sum(conv_rate * total_calls)]
```

$$\max \sum_{m,d} conv\_rate_{m,d} * total\_calls_{m,d}$$

Run the solver and display the results

```
result = SolverFactory('glpk').solve(model)
result.write()
```

Pyomo Modelling:

```
# Importing the environment
from pyomo.environ import *
import pandas as pd
```

```
# Reading the data from the Excel workbook - Bank_marketing.xlsx
```

```
InputData = "Bank_marketing_input.xlsx"

#Read the data from Campaign_Data sheet
data = pd.read_excel(InputData,sheet_name='Campaign_Data')

#Read the data from Call criteria sheet
criteria = pd.read_excel (InputData,sheet_name='Call criterias')

#Total budget for marketing
total_budget=150000

#cost per 10 mins of a call
cost_per_min=10
```

```
# Extracting the unique values of marital status and the educational degree which will
act as indexes for the decision variables and the parameters

M_status=data['Marital Status'].unique()
Degree=data['Degree'].unique()
```

```
# Creating the required data structures for the parameters with marital status and
degree as indexes

# Duration of calls for the customers not converted

duration_nconverted=data.set_index(['Marital Status', 'Degree'])['Avg. Call Duration
(Not Converted)'].to_dict()

# Duration of calls for the customers converted
duration_converted=data.set_index(['Marital Status', 'Degree'])['Avg. Call Duration
(Converted)'].to_dict()

# Conversion rate
conv_rate=data.set_index(['Marital Status', 'Degree'])['Conversion Rate'].to_dict()

#Minimum number of total calls
min_tcalls=criteria.set_index('Customer Segment')['Minimum number of calls'].to_dict()

#Minimum number of converted calls
```

```
min_ccalls=criteria.set_index('Customer Segment')['Minimum conversion'].to_dict()
```

```
#Instantiating a model

model = ConcreteModel()
```

```
# Decision variable - Total number of calls

model.total_calls = Var(M_status, Degree, within=PositiveIntegers)
```

```
# Defining the objective rule

def obj_rule(model):
    return sum(conv_rate[m,d]*model.total_calls[m,d] for m in M_status for d in Degree)

# Maximise the reach, i.e., converted calls
model.value = Objective(rule=obj_rule, sense= maximize)
```

```
#At least 50 customers need to be contacted from each customer segment

def min_calls_per_segment(model,m,d):
    return (model.total_calls[m,d])>=50

model.min_calls = Constraint(M_status,Degree,rule=min_calls_per_segment)
```

```
#The total number of calls made to each customer segment should meet the minimum
number (based on marital status)

def min_tcalls_married_status(model,m):
    return(sum(model.total_calls[m,d] for d in Degree) >= min_tcalls[m])
```

```
model.min_tcalls_mstatus = Constraint(M_status,rule=min_tcalls_married_status)


#The total number of calls made to each customer segment should meet the minimum number (based on the degree)

def min_tcalls_degree(model,d):
    return(sum(model.total_calls[m,d] for m in M_status) >= min_tcalls[d])

model.tcalls_deg = Constraint(Degree,rule=min_tcalls_degree)
```

```
#The total number of conversions/converted calls of the customer segments should match a minimum number (based on marital status)

def min_ccalls_married_status(model,m):
    return(sum(conv_rate[m,d]*model.total_calls[m,d] for d in Degree) >= min_ccalls[m])

model.min_ccalls_mstatus = Constraint(M_status,rule=min_ccalls_married_status)
```

```
#The total number of conversions/converted calls of the customer segments should match a minimum number (based on degree)

def min_ccalls_degree(model,d):
    return(sum(conv_rate[m,d]*model.total_calls[m,d] for m in M_status) >=
min_ccalls[d])

model.ccalls_deg = Constraint(Degree,rule=min_ccalls_degree)
```

```
#Budget constraint

def maximum_budget(model):
```

```
    return
sum((duration_converted[m,d]/60)*cost_per_min*model.total_calls[m,d]*conv_rate[m,d
] +
(duration_nconverted[m,d]/60)*cost_per_min*model.total_calls[m,d]*(1-conv_rate[m,d]
) for m in M_status for d in Degree) <= total_budget

model.max_budget=Constraint(rule= maximum_budget)
```

```
#Invoking the solver

result = SolverFactory('glpk').solve(model)
result.write()
```

The solver results show that **458** is the maximum number of converted calls we can get with the given constraints with an optimal number of calls to be made for each segment as shown in the following image.

```
        None :     1 :    Any :    3 : {'Bachelors', 'Doctorates', 'Masters'}

1 Var Declarations
    total_calls : Size=9, Index=total_calls_index
        Key                           : Lower : Value  : Upper : Fixed : Stale : Domain
         ('Divorced', 'Bachelors') :     1 :  812.0 :  None : False : False : PositiveIntegers
        ('Divorced', 'Doctorates') :     1 :   50.0 :  None : False : False : PositiveIntegers
           ('Divorced', 'Masters') :     1 :   50.0 :  None : False : False : PositiveIntegers
          ('Married', 'Bachelors') :     1 :   50.0 :  None : False : False : PositiveIntegers
         ('Married', 'Doctorates') :     1 : 1106.0 :  None : False : False : PositiveIntegers
            ('Married', 'Masters') :     1 :   53.0 :  None : False : False : PositiveIntegers
           ('Single', 'Bachelors') :     1 :   50.0 :  None : False : False : PositiveIntegers
          ('Single', 'Doctorates') :     1 :  386.0 :  None : False : False : PositiveIntegers
             ('Single', 'Masters') :     1 :  865.0 :  None : False : False : PositiveIntegers

1 Objective Declarations
```

The final results can be written into a data frame as follows.

RESULTS

Marital Status	Degree	Total Calls	Converted Calls	Estimated Cost (Rs.)
Married	Bachelors	50	3	2,094
Married	Masters	53	4	2,209
Married	Doctorates	1,106	141	47,264
Single	Bachelors	50	5	2,246
Single	Masters	865	109	37,195