Student: Melwyn D Souza

Student No: R00209495

Course: MSc in AI

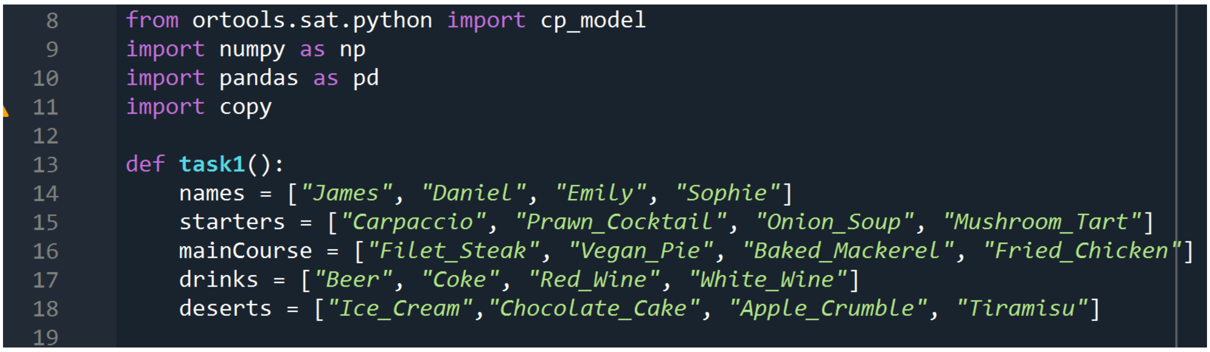
Module: Decision Analytics

Date: 10/Nov/2022

All the code snippets are captured with the line numbers of that block, these line numbers will match the python file submitted with this pdf

Task 1:

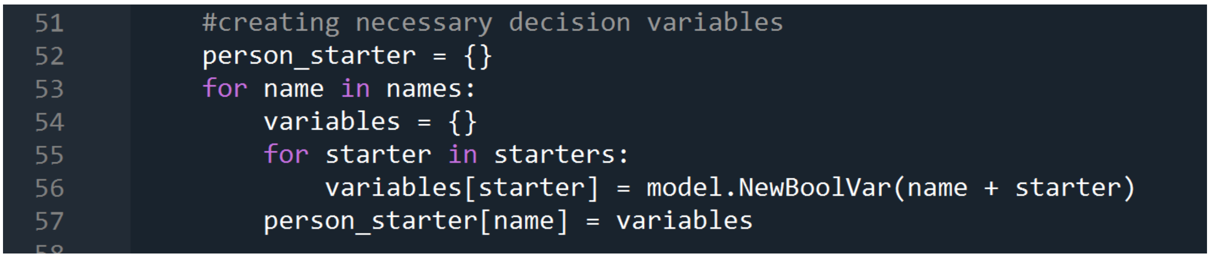
The task 1 is defined as a function with name task1() and is called at the end of the script A1.py submitted along with this report, the snippet below shows the initial few lines of task 1



```
 8  from ortools.sat.python import cp_model
 9  import numpy as np
10  import pandas as pd
11  import copy
12  
13  def task1():
14      names = ["James", "Daniel", "Emily", "Sophie"]
15      starters = ["Carpaccio", "Prawn_Cocktail", "Onion_Soup", "Mushroom_Tart"]
16      mainCourse = ["Filet_Steak", "Vegan_Pie", "Baked_Mackerel", "Fried_Chicken"]
17      drinks = ["Beer", "Coke", "Red_Wine", "White_Wine"]
18      deserts = ["Ice_Cream","Chocolate_Cake", "Apple_Crumble", "Tiramisu"]
19  
```

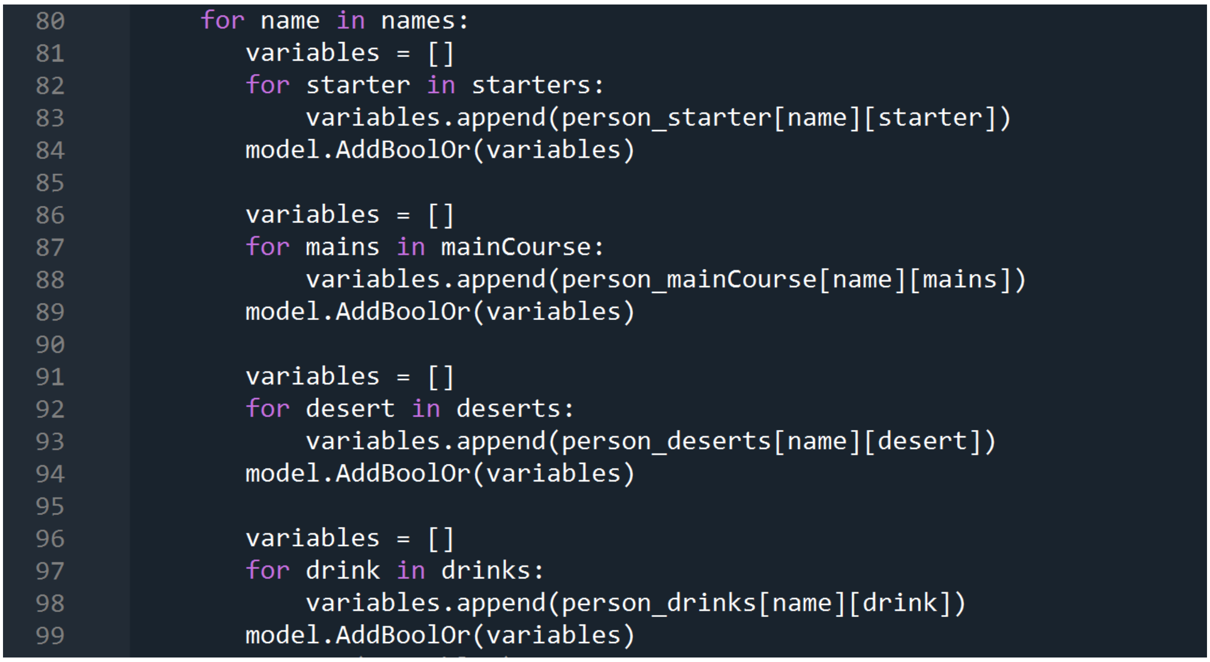
The task is to identify objects predicates and attributes and solve using CT-SAT solver using necessary decision variables

The snippet below is where decision variables are created for each possible pair



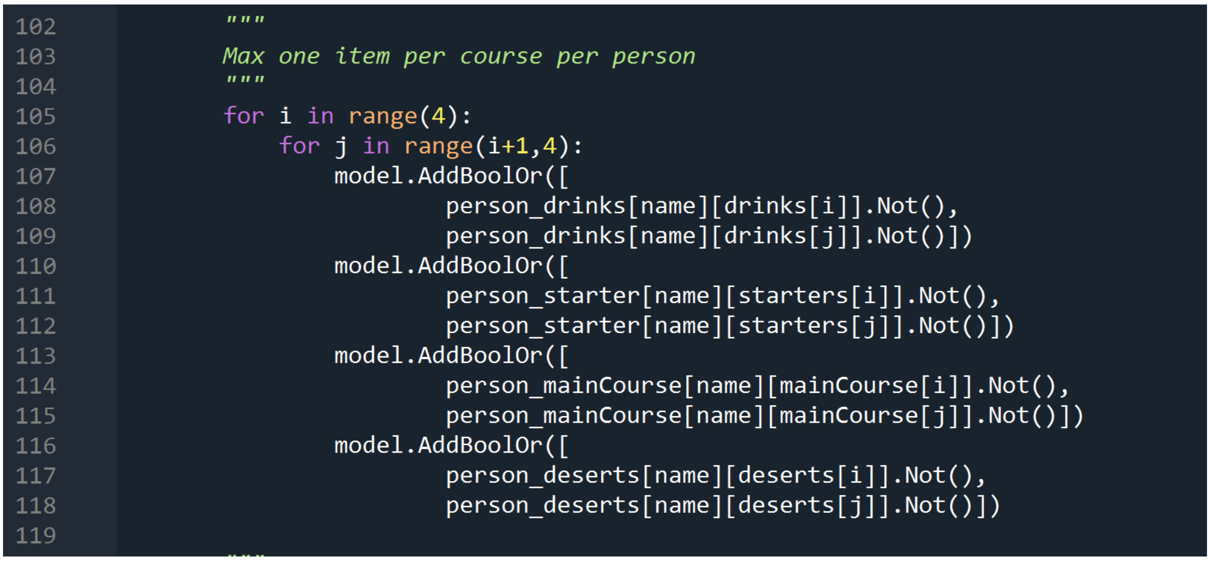
```
51      #creating necessary decision variables
52      person_starter = {}
53      for name in names:
54          variables = {}
55          for starter in starters:
56              variables[starter] = model.NewBoolVar(name + starter)
57          person_starter[name] = variables
```

The snippet below is to create at least one item from each course for each person



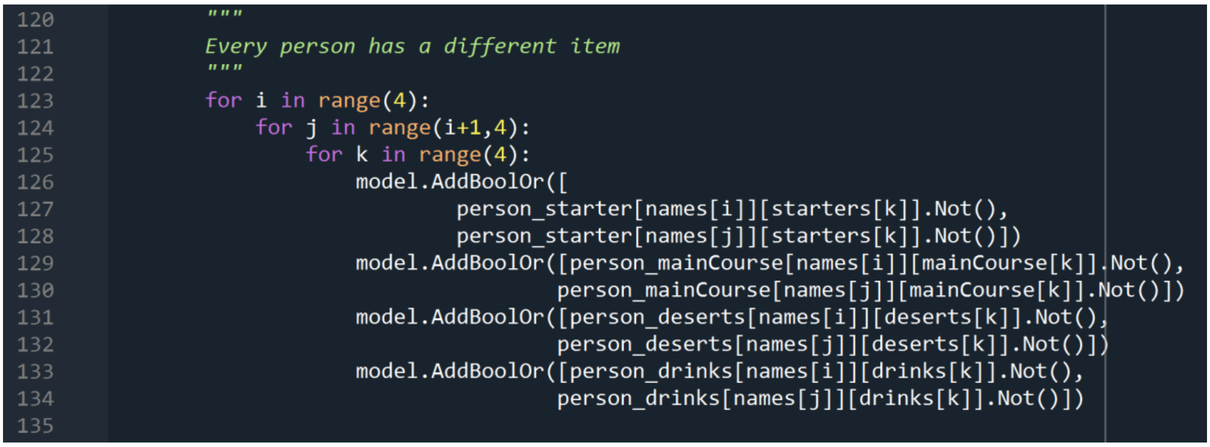
```
    for name in names:
       variables = []
       for starter in starters:
           variables.append(person_starter[name][starter])
       model.AddBoolOr(variables)

       variables = []
       for mains in mainCourse:
           variables.append(person_mainCourse[name][mains])
       model.AddBoolOr(variables)

       variables = []
       for desert in deserts:
           variables.append(person_deserts[name][desert])
       model.AddBoolOr(variables)

       variables = []
       for drink in drinks:
           variables.append(person_drinks[name][drink])
       model.AddBoolOr(variables)
```

A person can only have one item from each course, hence maximum is one item as shown in below snippet



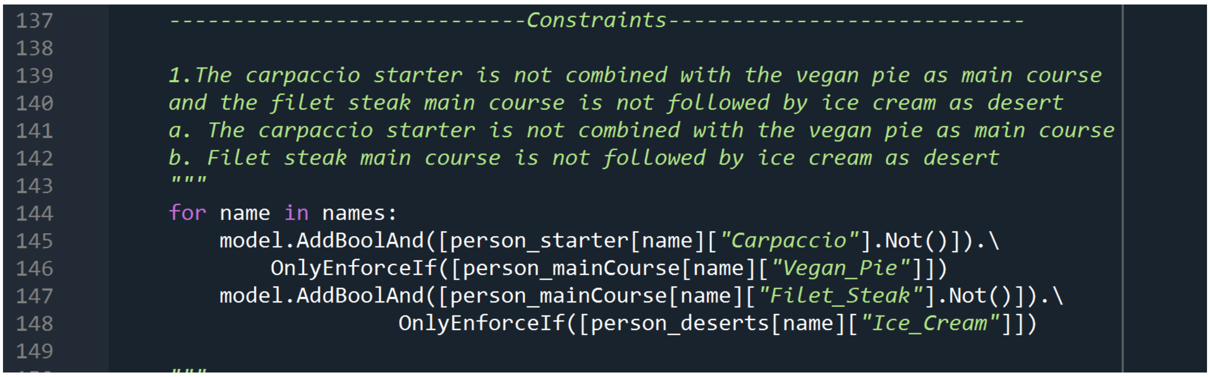
```
        """
        Max one item per course per person
        """
        for i in range(4):
            for j in range(i+1,4):
                model.AddBoolOr([
                        person_drinks[name][drinks[i]].Not(),
                        person_drinks[name][drinks[j]].Not()])
                model.AddBoolOr([
                        person_starter[name][starters[i]].Not(),
                        person_starter[name][starters[j]].Not()])
                model.AddBoolOr([
                        person_mainCourse[name][mainCourse[i]].Not(),
                        person_mainCourse[name][mainCourse[j]].Not()])
                model.AddBoolOr([
                        person_deserts[name][deserts[i]].Not(),
                        person_deserts[name][deserts[j]].Not()])

```

My assumption from task 1 description is that each person eats an item from each course, the variables for this are scripted as shown below



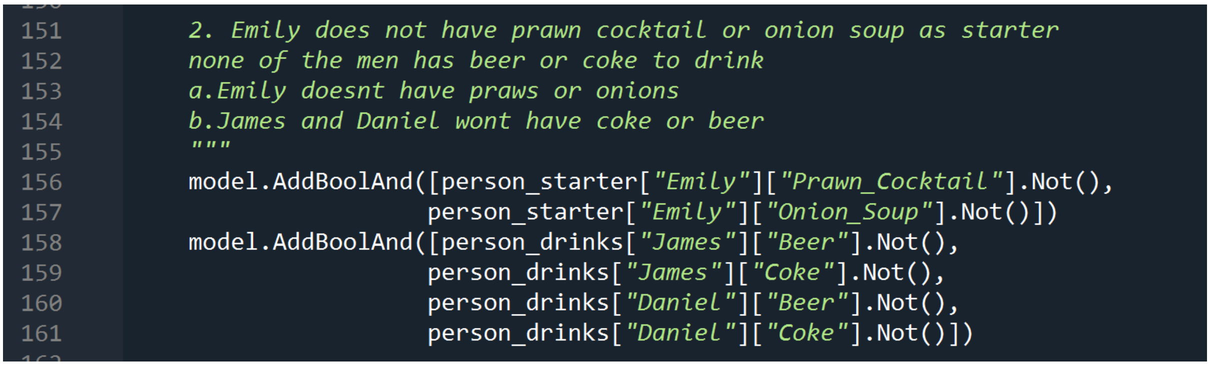
```
        """
        Every person has a different item
        """
        for i in range(4):
            for j in range(i+1,4):
                for k in range(4):
                    model.AddBoolOr([
                             person_starter[names[i]][starters[k]].Not(),
                             person_starter[names[j]][starters[k]].Not()])
                    model.AddBoolOr([person_mainCourse[names[i]][mainCourse[k]].Not(),
                                     person_mainCourse[names[j]][mainCourse[k]].Not()])
                    model.AddBoolOr([person_deserts[names[i]][deserts[k]].Not(),
                                     person_deserts[names[j]][deserts[k]].Not()])
                    model.AddBoolOr([person_drinks[names[i]][drinks[k]].Not(),
                                     person_drinks[names[j]][drinks[k]].Not()])
```

Sentence 1 constraints are as follows



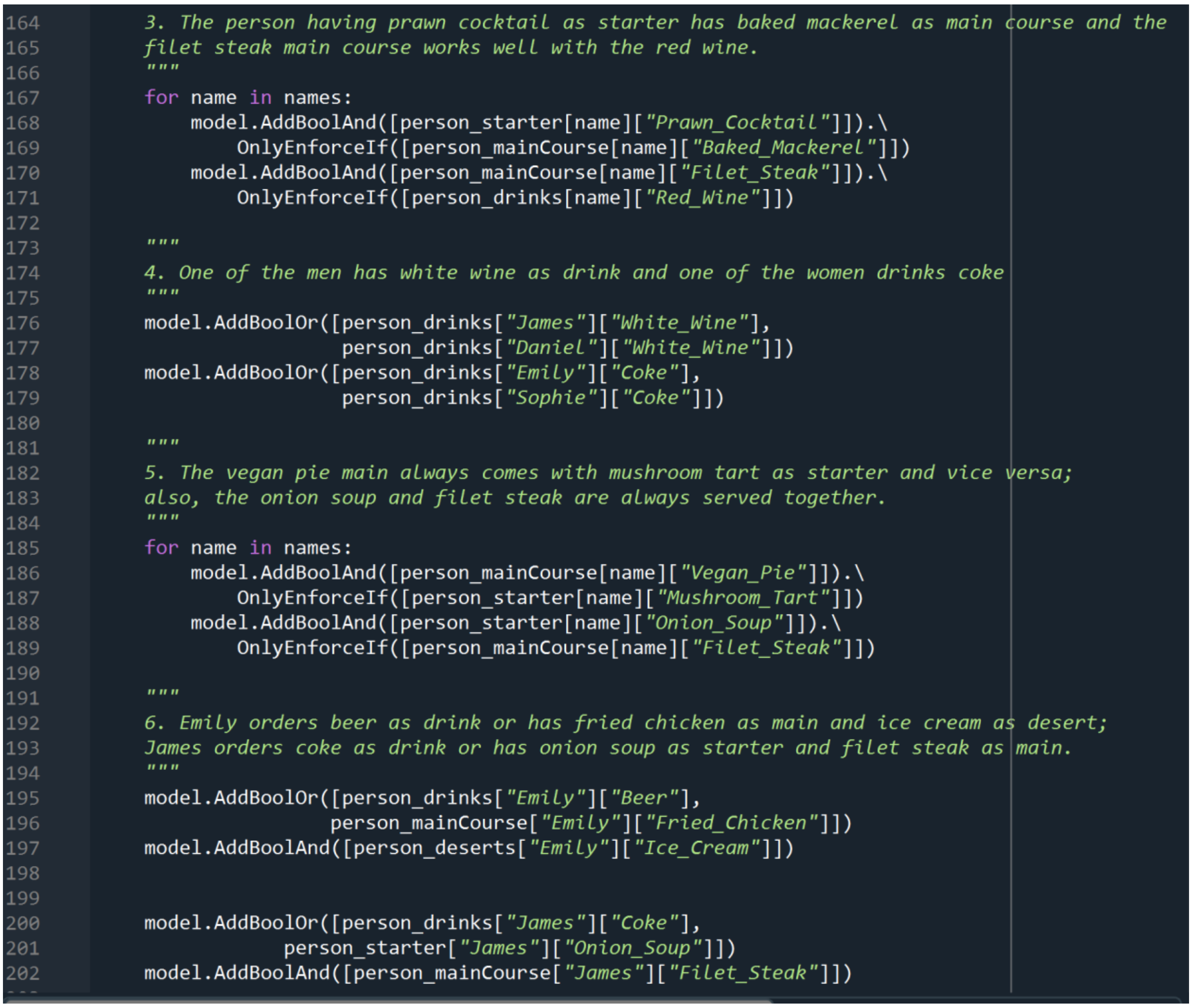
```
    ----------------------------Constraints----------------------------

    1.The carpaccio starter is not combined with the vegan pie as main course
    and the filet steak main course is not followed by ice cream as desert
    a. The carpaccio starter is not combined with the vegan pie as main course
    b. Filet steak main course is not followed by ice cream as desert
    """
    for name in names:
        model.AddBoolAnd([person_starter[name]["Carpaccio"].Not()]).\
            OnlyEnforceIf([person_mainCourse[name]["Vegan_Pie"]])
        model.AddBoolAnd([person_mainCourse[name]["Filet_Steak"].Not()]).\
                      OnlyEnforceIf([person_deserts[name]["Ice_Cream"]])

    """
```

Sentence 2 constraints are as follows:



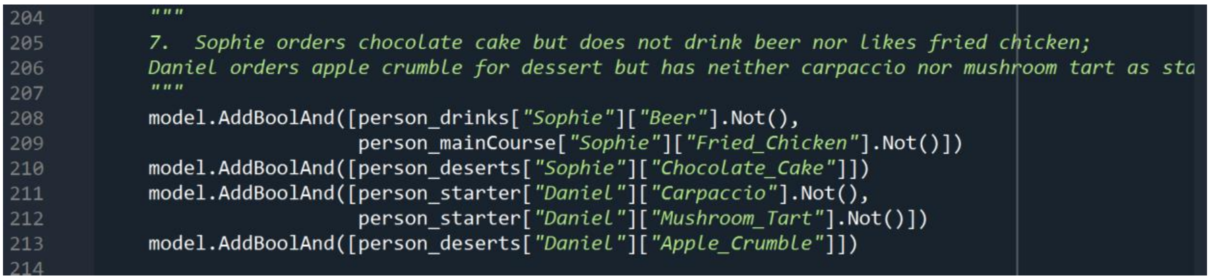
```
    2. Emily does not have prawn cocktail or onion soup as starter
    none of the men has beer or coke to drink
    a.Emily doesnt have praws or onions
    b.James and Daniel wont have coke or beer
    """
    model.AddBoolAnd([person_starter["Emily"]["Prawn_Cocktail"].Not(),
                     person_starter["Emily"]["Onion_Soup"].Not()])
    model.AddBoolAnd([person_drinks["James"]["Beer"].Not(),
                     person_drinks["James"]["Coke"].Not(),
                     person_drinks["Daniel"]["Beer"].Not(),
                     person_drinks["Daniel"]["Coke"].Not()])
```

Sentence 3, 4, 5 and 6 constraints are as shown in the snippet below



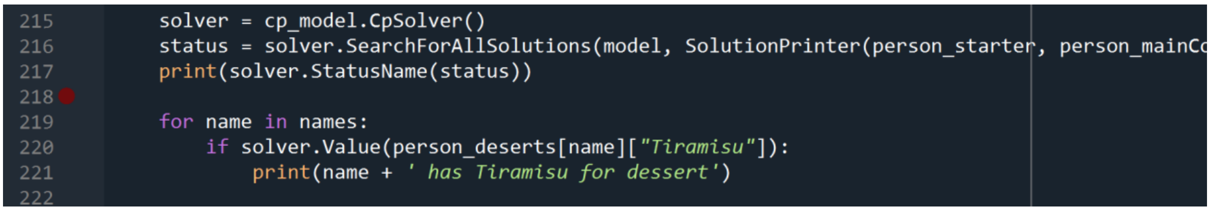
```
    3. The person having prawn cocktail as starter has baked mackerel as main course and the
    filet steak main course works well with the red wine.
    """
    for name in names:
        model.AddBoolAnd([person_starter[name]["Prawn_Cocktail"]]).\
            OnlyEnforceIf([person_mainCourse[name]["Baked_Mackerel"]])
        model.AddBoolAnd([person_mainCourse[name]["Filet_Steak"]]).\
            OnlyEnforceIf([person_drinks[name]["Red_Wine"]])

    """
    4. One of the men has white wine as drink and one of the women drinks coke
    """
    model.AddBoolOr([person_drinks["James"]["White_Wine"],
                     person_drinks["Daniel"]["White_Wine"]])
    model.AddBoolOr([person_drinks["Emily"]["Coke"],
                     person_drinks["Sophie"]["Coke"]])

    """
    5. The vegan pie main always comes with mushroom tart as starter and vice versa;
    also, the onion soup and filet steak are always served together.
    """
    for name in names:
        model.AddBoolAnd([person_mainCourse[name]["Vegan_Pie"]]).\
            OnlyEnforceIf([person_starter[name]["Mushroom_Tart"]])
        model.AddBoolAnd([person_starter[name]["Onion_Soup"]]).\
            OnlyEnforceIf([person_mainCourse[name]["Filet_Steak"]])

    """
    6. Emily orders beer as drink or has fried chicken as main and ice cream as desert;
    James orders coke as drink or has onion soup as starter and filet steak as main.
    """
    model.AddBoolOr([person_drinks["Emily"]["Beer"],
                    person_mainCourse["Emily"]["Fried_Chicken"]])
    model.AddBoolAnd([person_deserts["Emily"]["Ice_Cream"]])


    model.AddBoolOr([person_drinks["James"]["Coke"],
                person_starter["James"]["Onion_Soup"]])
    model.AddBoolAnd([person_mainCourse["James"]["Filet_Steak"]])
```

Likewise for sentence 7 constraints are as shown below



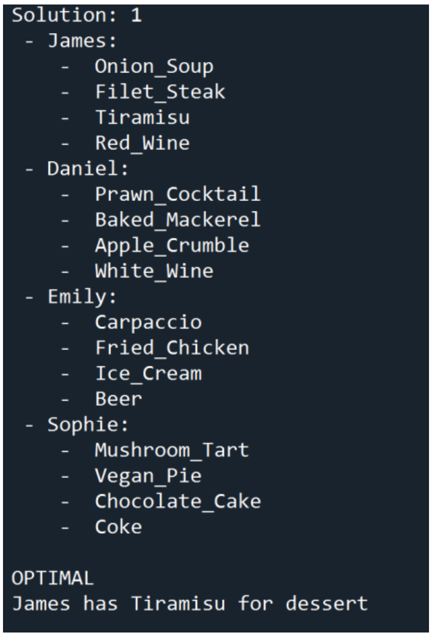
```
    """
    7.  Sophie orders chocolate cake but does not drink beer nor likes fried chicken;
    Daniel orders apple crumble for dessert but has neither carpaccio nor mushroom tart as sta
    """
    model.AddBoolAnd([person_drinks["Sophie"]["Beer"].Not(),
                      person_mainCourse["Sophie"]["Fried_Chicken"].Not()])
    model.AddBoolAnd([person_deserts["Sophie"]["Chocolate_Cake"]])
    model.AddBoolAnd([person_starter["Daniel"]["Carpaccio"].Not(),
                      person_starter["Daniel"]["Mushroom_Tart"].Not()])
    model.AddBoolAnd([person_deserts["Daniel"]["Apple_Crumble"]])
```

CT_SAT solver is used to solve for tiramisu question



```
    solver = cp_model.CpSolver()
    status = solver.SearchForAllSolutions(model, SolutionPrinter(person_starter, person_mainCo
    print(solver.StatusName(status))

    for name in names:
        if solver.Value(person_deserts[name]["Tiramisu"]):
            print(name + ' has Tiramisu for dessert')
```

The CP-SAT solver finds optimal solution and the tiramisu question is hence answered, James has tiramisu for dessert, the solution is printed in the console as shown below



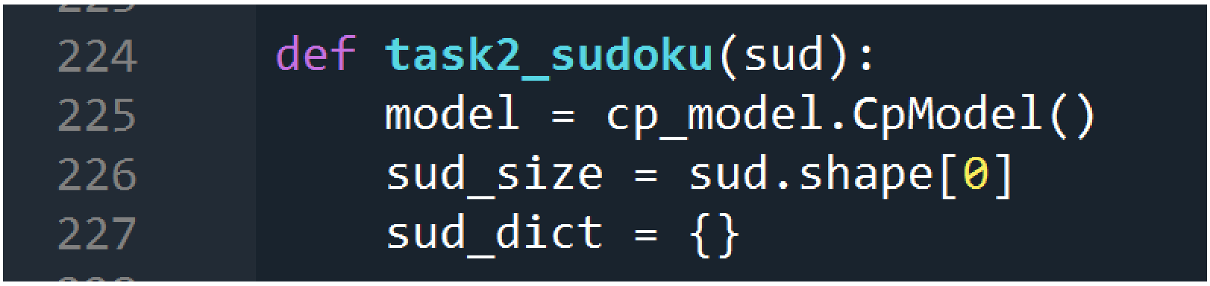
```
Solution: 1
 - James:
    -  Onion_Soup
    -  Filet_Steak
    -  Tiramisu
    -  Red_Wine
 - Daniel:
    -  Prawn_Cocktail
    -  Baked_Mackerel
    -  Apple_Crumble
    -  White_Wine
 - Emily:
    -  Carpaccio
    -  Fried_Chicken
    -  Ice_Cream
    -  Beer
 - Sophie:
    -  Mushroom_Tart
    -  Vegan_Pie
    -  Chocolate_Cake
    -  Coke

OPTIMAL
James has Tiramisu for dessert
```

Task 2:

Sudoku solver using CP-SAT

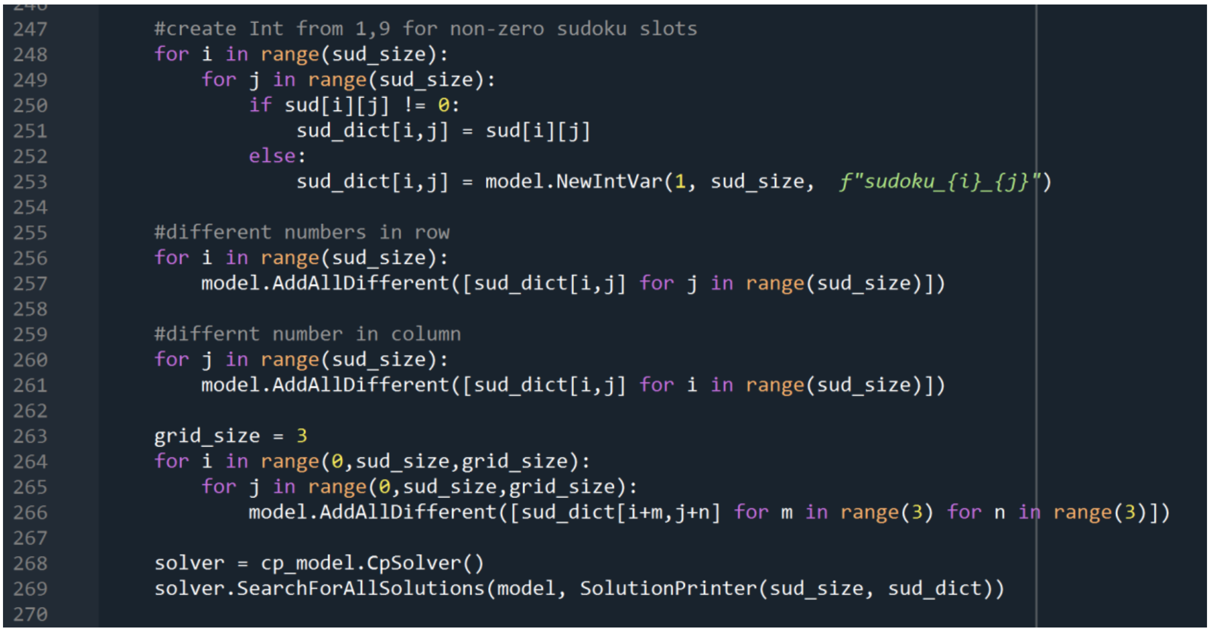
Task 2 is defined as a function with name task2_sudoku(sud):

It takes one argument which is the sudoku values matrix (numpy array)



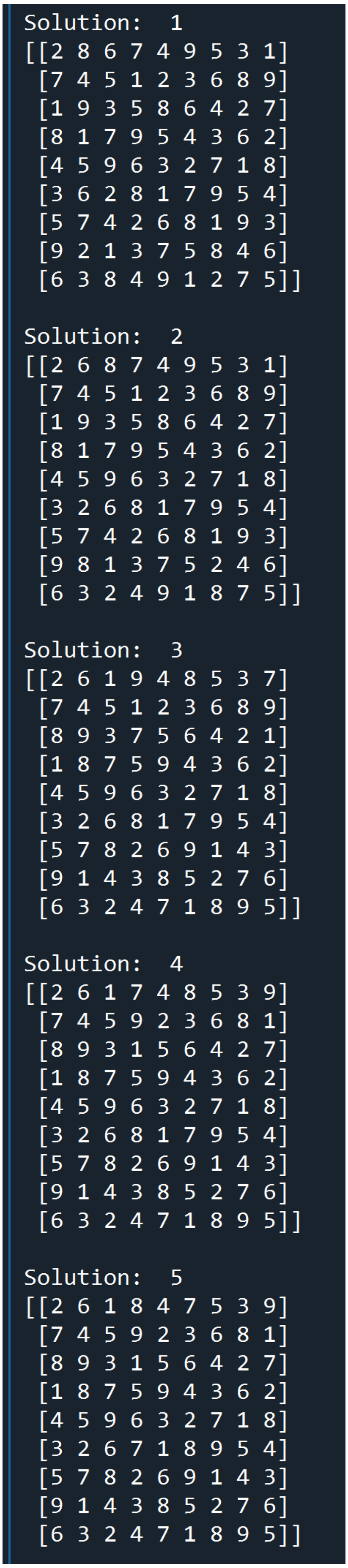
```python
def task2_sudoku(sud):
    model = cp_model.CpModel()
    sud_size = sud.shape[0]
    sud_dict = {}
```

The code snippet below and the comments explain the steps followed



```
    #create Int from 1,9 for non-zero sudoku slots
    for i in range(sud_size):
        for j in range(sud_size):
            if sud[i][j] != 0:
                sud_dict[i,j] = sud[i][j]
            else:
                sud_dict[i,j] = model.NewIntVar(1, sud_size,  f"sudoku_{i}_{j}")

    #different numbers in row
    for i in range(sud_size):
        model.AddAllDifferent([sud_dict[i,j] for j in range(sud_size)])

    #differnt number in column
    for j in range(sud_size):
        model.AddAllDifferent([sud_dict[i,j] for i in range(sud_size)])

    grid_size = 3
    for i in range(0,sud_size,grid_size):
        for j in range(0,sud_size,grid_size):
            model.AddAllDifferent([sud_dict[i+m,j+n] for m in range(3) for n in range(3)])

    solver = cp_model.CpSolver()
    solver.SearchForAllSolutions(model, SolutionPrinter(sud_size, sud_dict))

```

The solver finds 5 solutions for this problem, and they are printed in console as shown below



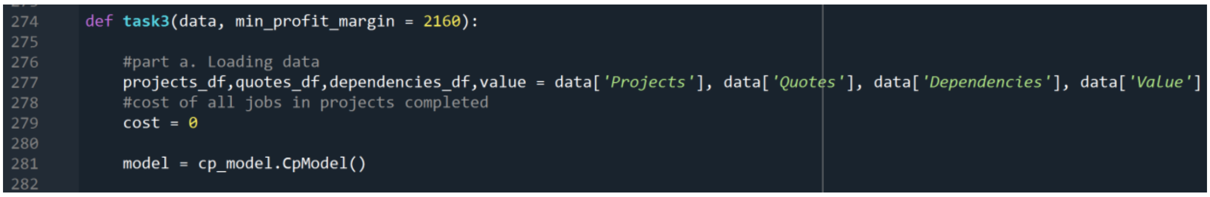
```
Solution:  1
[[2 8 6 7 4 9 5 3 1]
 [7 4 5 1 2 3 6 8 9]
 [1 9 3 5 8 6 4 2 7]
 [8 1 7 9 5 4 3 6 2]
 [4 5 9 6 3 2 7 1 8]
 [3 6 2 8 1 7 9 5 4]
 [5 7 4 2 6 8 1 9 3]
 [9 2 1 3 7 5 8 4 6]
 [6 3 8 4 9 1 2 7 5]]

Solution:  2
[[2 6 8 7 4 9 5 3 1]
 [7 4 5 1 2 3 6 8 9]
 [1 9 3 5 8 6 4 2 7]
 [8 1 7 9 5 4 3 6 2]
 [4 5 9 6 3 2 7 1 8]
 [3 2 6 8 1 7 9 5 4]
 [5 7 4 2 6 8 1 9 3]
 [9 8 1 3 7 5 2 4 6]
 [6 3 2 4 9 1 8 7 5]]

Solution:  3
[[2 6 1 9 4 8 5 3 7]
 [7 4 5 1 2 3 6 8 9]
 [8 9 3 7 5 6 4 2 1]
 [1 8 7 5 9 4 3 6 2]
 [4 5 9 6 3 2 7 1 8]
 [3 2 6 8 1 7 9 5 4]
 [5 7 8 2 6 9 1 4 3]
 [9 1 4 3 8 5 2 7 6]
 [6 3 2 4 7 1 8 9 5]]

Solution:  4
[[2 6 1 7 4 8 5 3 9]
 [7 4 5 9 2 3 6 8 1]
 [8 9 3 1 5 6 4 2 7]
 [1 8 7 5 9 4 3 6 2]
 [4 5 9 6 3 2 7 1 8]
 [3 2 6 8 1 7 9 5 4]
 [5 7 8 2 6 9 1 4 3]
 [9 1 4 3 8 5 2 7 6]
 [6 3 2 4 7 1 8 9 5]]

Solution:  5
[[2 6 1 8 4 7 5 3 9]
 [7 4 5 9 2 3 6 8 1]
 [8 9 3 1 5 6 4 2 7]
 [1 8 7 5 9 4 3 6 2]
 [4 5 9 6 3 2 7 1 8]
 [3 2 6 7 1 8 9 5 4]
 [5 7 8 2 6 9 1 4 3]
 [9 1 4 3 8 5 2 7 6]
 [6 3 2 4 7 1 8 9 5]]
```

Task 3:

The task was to find the optimal way of assigning the project jobs to contractors to finish different projects on time while keeping the profit margin 2160

The task 3 is defined as a function called task3(data, min_profit_margin = 2160)

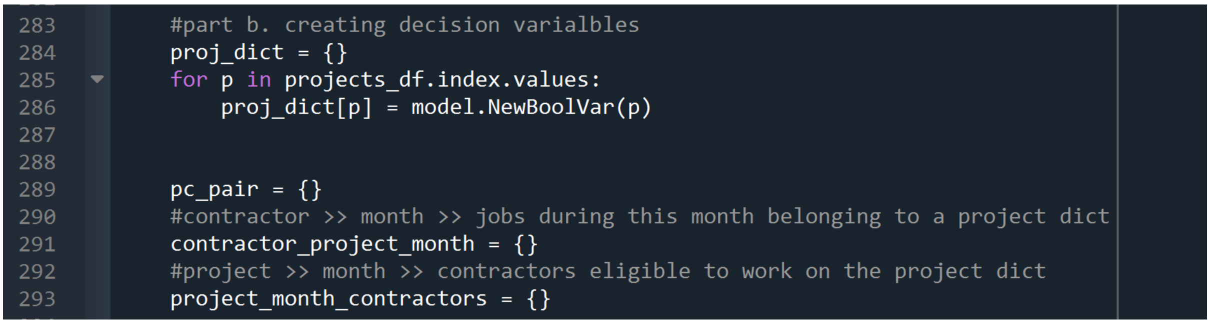
Argument data is read from the xlsx file as dataframes projects, quotes, dependencies and values



```python
def task3(data, min_profit_margin = 2160):

    #part a. Loading data
    projects_df,quotes_df,dependencies_df,value = data['Projects'], data['Quotes'], data['Dependencies'], data['Value']
    #cost of all jobs in projects completed
    cost = 0

    model = cp_model.CpModel()

```

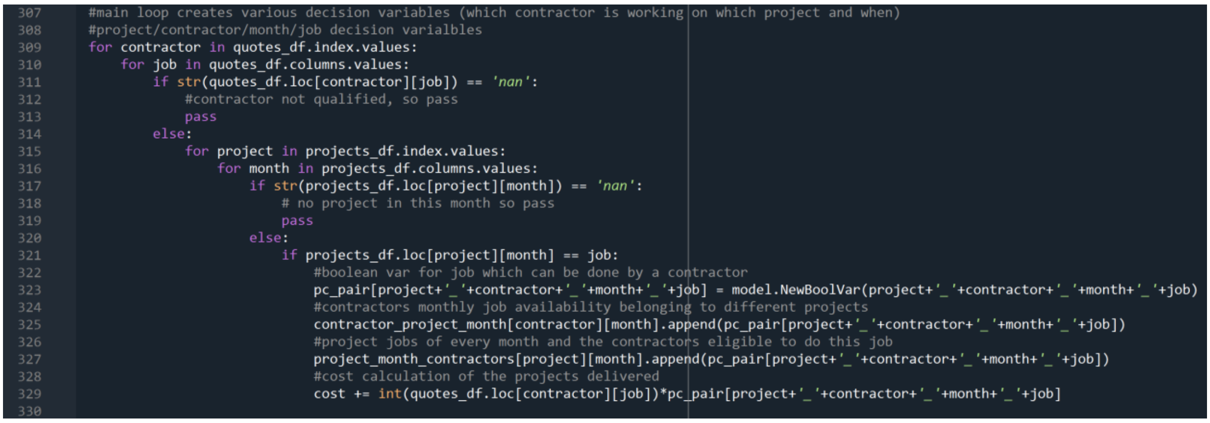
Main decision variables are created for projects

Project contractor pair, contractor project month pairs are created which are used for different blocks later in the script



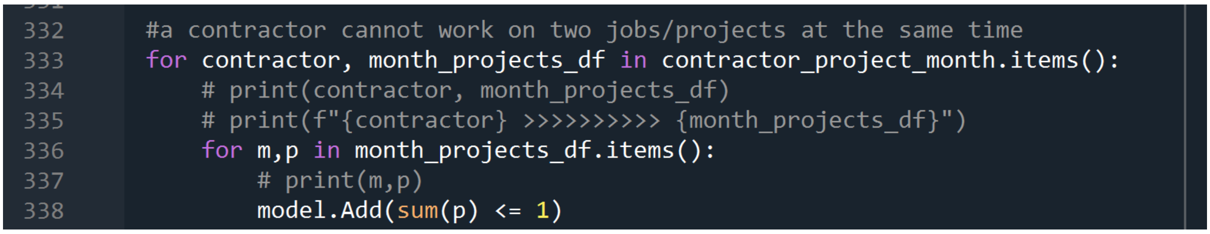
```python
    #part b. creating decision varialbles
    proj_dict = {}
    for p in projects_df.index.values:
        proj_dict[p] = model.NewBoolVar(p)


    pc_pair = {}
    #contractor >> month >> jobs during this month belonging to a project dict
    contractor_project_month = {}
    #project >> month >> contractors eligible to work on the project dict
    project_month_contractors = {}
```

This is the main loop in the snippet below, it loops on all the dataframes to add many Boolean constraints



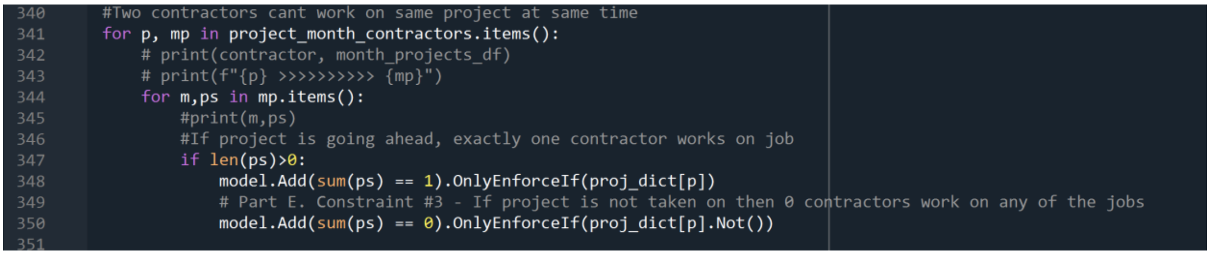
```python
    #main loop creates various decision variables (which contractor is working on which project and when)
    #project/contractor/month/job decision varialbles
    for contractor in quotes_df.index.values:
        for job in quotes_df.columns.values:
            if str(quotes_df.loc[contractor][job]) == 'nan':
                #contractor not qualified, so pass
                pass
            else:
                for project in projects_df.index.values:
                    for month in projects_df.columns.values:
                        if str(projects_df.loc[project][month]) == 'nan':
                            # no project in this month so pass
                            pass
                        else:
                            if projects_df.loc[project][month] == job:
                                #boolean var for job which can be done by a contractor
                                pc_pair[project+'_'+contractor+'_'+month+'_'+job] = model.NewBoolVar(project+'_'+contractor+'_'+month+'_'+job)
                                #contractors monthly job availability belonging to different projects
                                contractor_project_month[contractor][month].append(pc_pair[project+'_'+contractor+'_'+month+'_'+job])
                                #project jobs of every month and the contractors eligible to do this job
                                project_month_contractors[project][month].append(pc_pair[project+'_'+contractor+'_'+month+'_'+job])
                                #cost calculation of the projects delivered
                                cost += int(quotes_df.loc[contractor][job])*pc_pair[project+'_'+contractor+'_'+month+'_'+job]

```

A contractor can only work on one project job every month, this is constructed as shown below



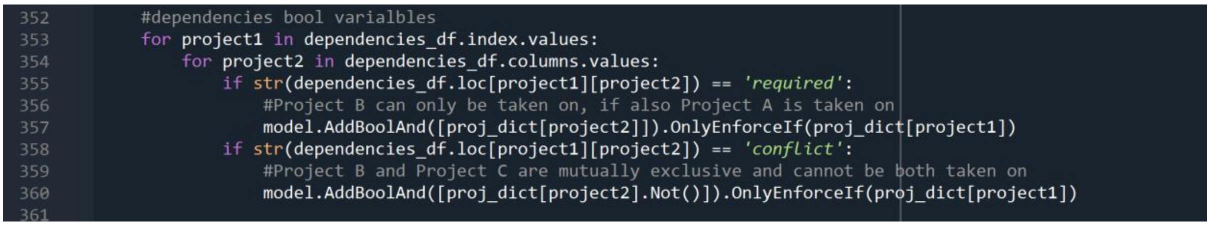
```
#a contractor cannot work on two jobs/projects at the same time
for contractor, month_projects_df in contractor_project_month.items():
    # print(contractor, month_projects_df)
    # print(f"{contractor} >>>>>>>>>> {month_projects_df}")
    for m,p in month_projects_df.items():
        # print(m,p)
        model.Add(sum(p) <= 1)
```

Two contractors cannot work on the same project job, this is shown in the snippet below



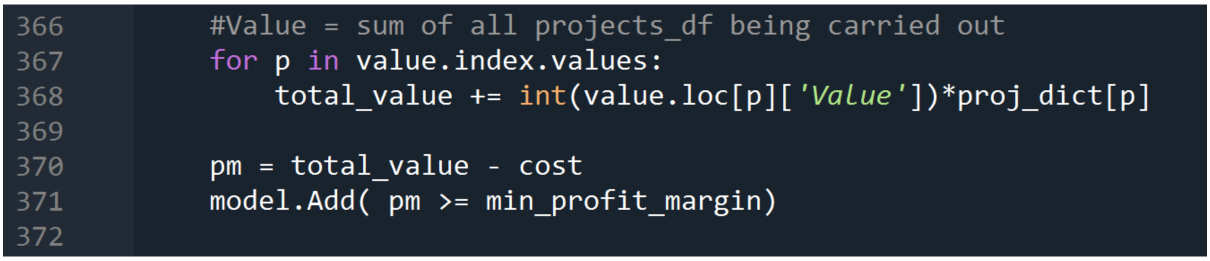
```
#Two contractors cant work on same project at same time
for p, mp in project_month_contractors.items():
    # print(contractor, month_projects_df)
    # print(f"{p} >>>>>>>>>> {mp}")
    for m,ps in mp.items():
        #print(m,ps)
        #If project is going ahead, exactly one contractor works on job
        if len(ps)>0:
            model.Add(sum(ps) == 1).OnlyEnforceIf(proj_dict[p])
            # Part E. Constraint #3 - If project is not taken on then 0 contractors work on any of the jobs
            model.Add(sum(ps) == 0).OnlyEnforceIf(proj_dict[p].Not())
```

The dependencies of each project is checked and constraints are added a shown below



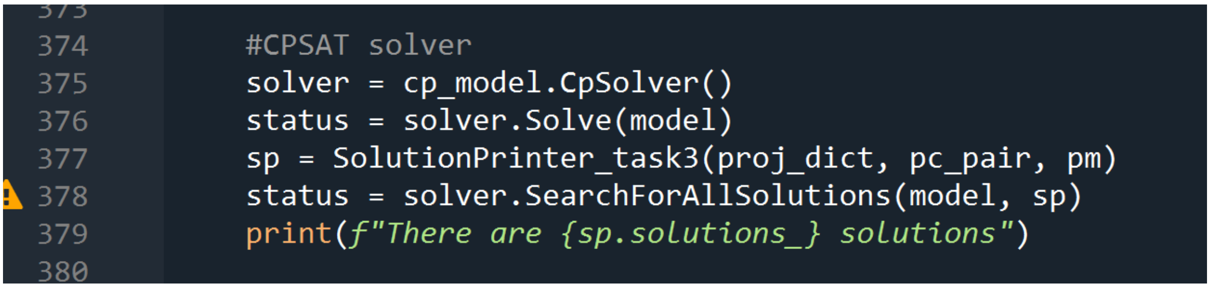
```
#dependencies bool varialbles
for project1 in dependencies_df.index.values:
    for project2 in dependencies_df.columns.values:
        if str(dependencies_df.loc[project1][project2]) == 'required':
            #Project B can only be taken on, if also Project A is taken on
            model.AddBoolAnd([proj_dict[project2]]).OnlyEnforceIf(proj_dict[project1])
        if str(dependencies_df.loc[project1][project2]) == 'conflict':
            #Project B and Project C are mutually exclusive and cannot be both taken on
            model.AddBoolAnd([proj_dict[project2].Not()]).OnlyEnforceIf(proj_dict[project1])
```

The value/cost of all the projects carried on by various contractors is calculated and the profit margin is found, the margin should be more than the minimum profit margin supplied to the script



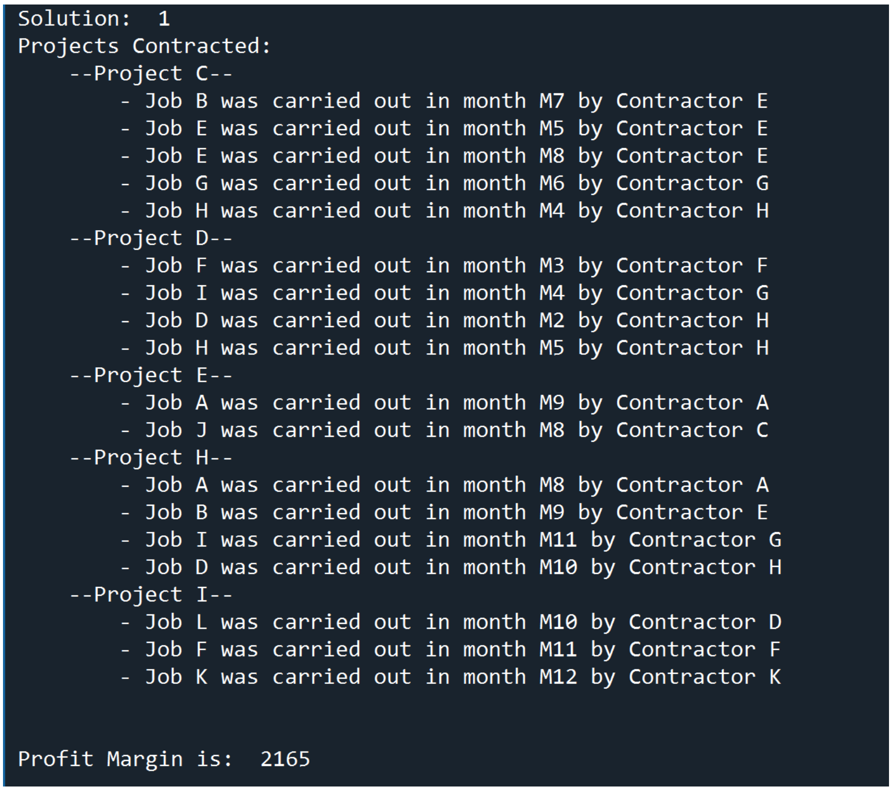
```
#Value = sum of all projects_df being carried out
for p in value.index.values:
    total_value += int(value.loc[p]['Value'])*proj_dict[p]

pm = total_value - cost
model.Add( pm >= min_profit_margin)
```

Finally, CP-SAT is called to solve the constraints and find the optimal solution

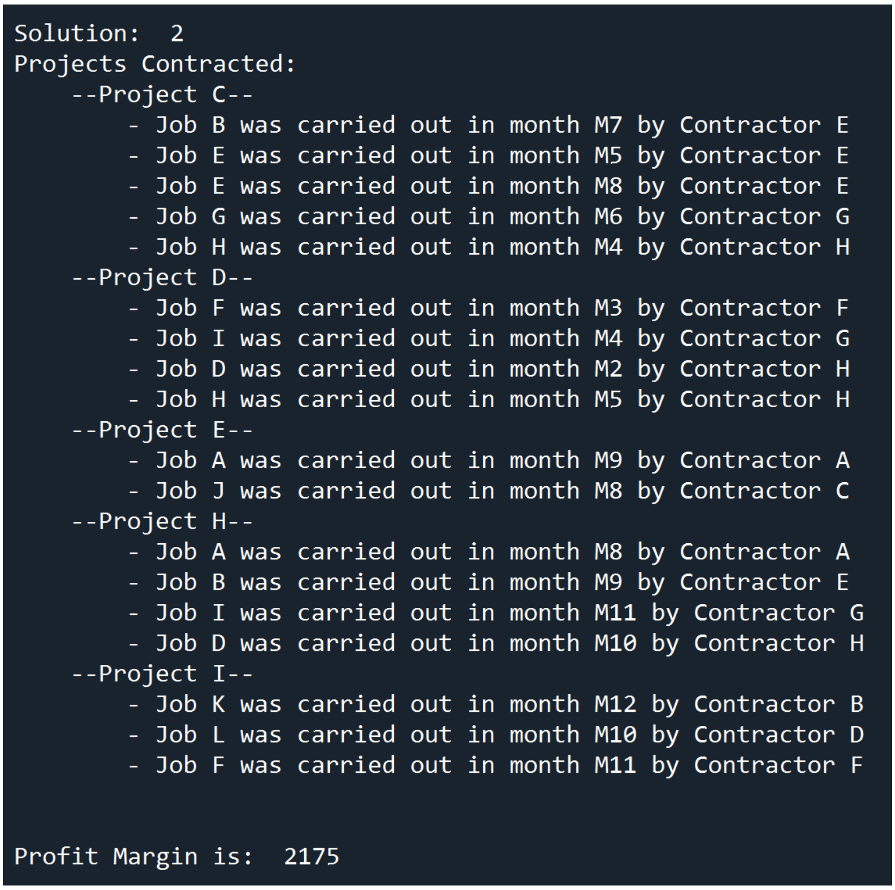


```
#CPSAT solver
solver = cp_model.CpSolver()
status = solver.Solve(model)
sp = SolutionPrinter_task3(proj_dict, pc_pair, pm)
status = solver.SearchForAllSolutions(model, sp)
print(f"There are {sp.solutions_} solutions")
```

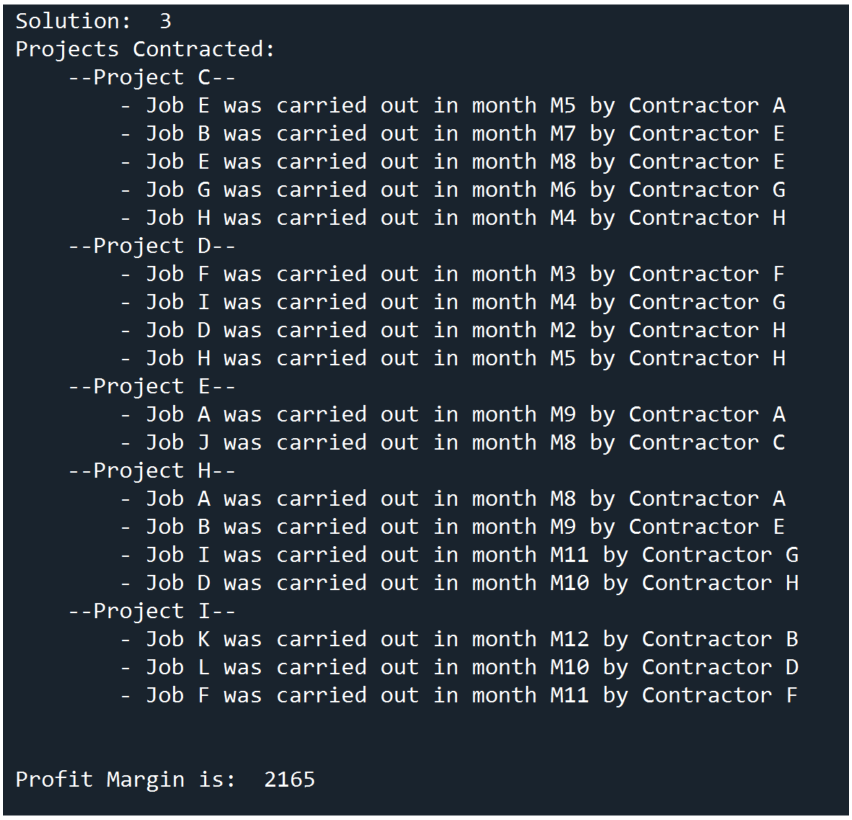
There are 5 optimal solutions found by CP-SAT solver and they are as shown in the snippets below



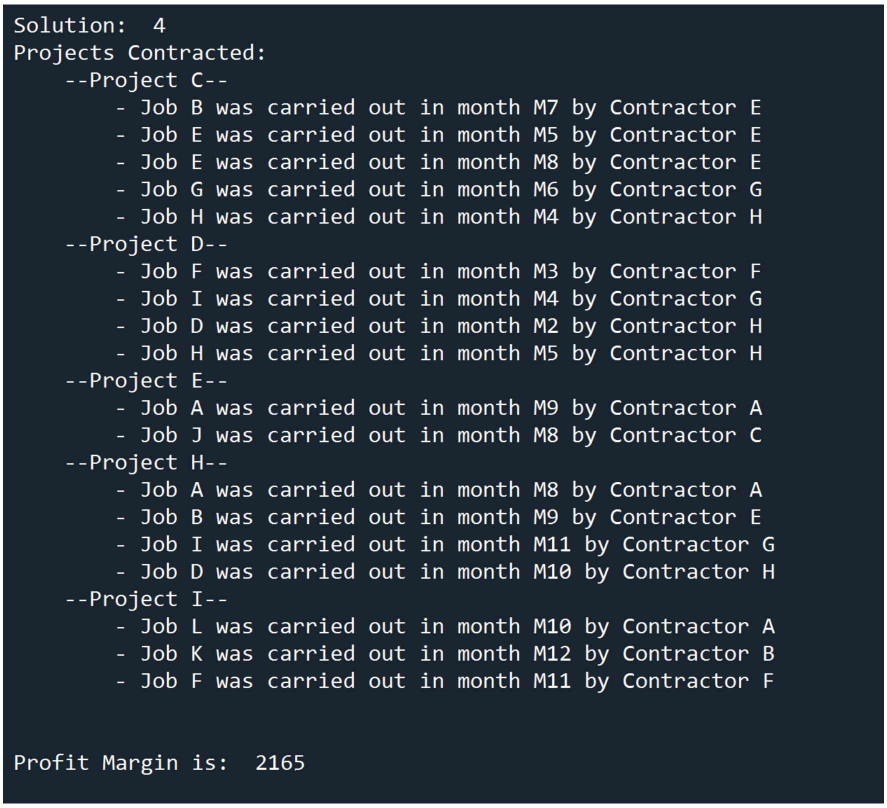
```
Solution:  1
Projects Contracted:
    --Project C--
        - Job B was carried out in month M7 by Contractor E
        - Job E was carried out in month M5 by Contractor E
        - Job E was carried out in month M8 by Contractor E
        - Job G was carried out in month M6 by Contractor G
        - Job H was carried out in month M4 by Contractor H
    --Project D--
        - Job F was carried out in month M3 by Contractor F
        - Job I was carried out in month M4 by Contractor G
        - Job D was carried out in month M2 by Contractor H
        - Job H was carried out in month M5 by Contractor H
    --Project E--
        - Job A was carried out in month M9 by Contractor A
        - Job J was carried out in month M8 by Contractor C
    --Project H--
        - Job A was carried out in month M8 by Contractor A
        - Job B was carried out in month M9 by Contractor E
        - Job I was carried out in month M11 by Contractor G
        - Job D was carried out in month M10 by Contractor H
    --Project I--
        - Job L was carried out in month M10 by Contractor D
        - Job F was carried out in month M11 by Contractor F
        - Job K was carried out in month M12 by Contractor K


Profit Margin is:  2165
```



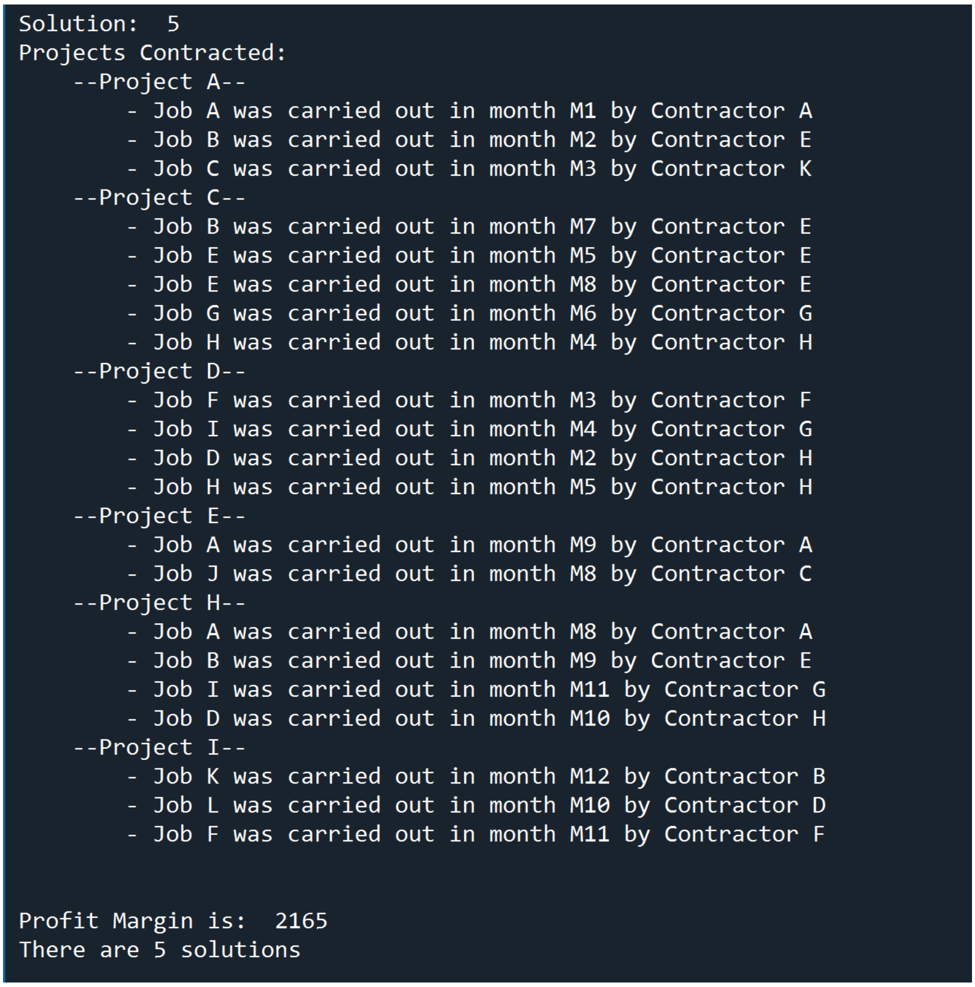
```
Solution:  2
Projects Contracted:
    --Project C--
        - Job B was carried out in month M7 by Contractor E
        - Job E was carried out in month M5 by Contractor E
        - Job E was carried out in month M8 by Contractor E
        - Job G was carried out in month M6 by Contractor G
        - Job H was carried out in month M4 by Contractor H
    --Project D--
        - Job F was carried out in month M3 by Contractor F
        - Job I was carried out in month M4 by Contractor G
        - Job D was carried out in month M2 by Contractor H
        - Job H was carried out in month M5 by Contractor H
    --Project E--
        - Job A was carried out in month M9 by Contractor A
        - Job J was carried out in month M8 by Contractor C
    --Project H--
        - Job A was carried out in month M8 by Contractor A
        - Job B was carried out in month M9 by Contractor E
        - Job I was carried out in month M11 by Contractor G
        - Job D was carried out in month M10 by Contractor H
    --Project I--
        - Job K was carried out in month M12 by Contractor B
        - Job L was carried out in month M10 by Contractor D
        - Job F was carried out in month M11 by Contractor F


Profit Margin is:  2175
```



```
Solution:  3
Projects Contracted:
    --Project C--
        - Job E was carried out in month M5 by Contractor A
        - Job B was carried out in month M7 by Contractor E
        - Job E was carried out in month M8 by Contractor E
        - Job G was carried out in month M6 by Contractor G
        - Job H was carried out in month M4 by Contractor H
    --Project D--
        - Job F was carried out in month M3 by Contractor F
        - Job I was carried out in month M4 by Contractor G
        - Job D was carried out in month M2 by Contractor H
        - Job H was carried out in month M5 by Contractor H
    --Project E--
        - Job A was carried out in month M9 by Contractor A
        - Job J was carried out in month M8 by Contractor C
    --Project H--
        - Job A was carried out in month M8 by Contractor A
        - Job B was carried out in month M9 by Contractor E
        - Job I was carried out in month M11 by Contractor G
        - Job D was carried out in month M10 by Contractor H
    --Project I--
        - Job K was carried out in month M12 by Contractor B
        - Job L was carried out in month M10 by Contractor D
        - Job F was carried out in month M11 by Contractor F


Profit Margin is:  2165
```



```
Solution:  4
Projects Contracted:
    --Project C--
        - Job B was carried out in month M7 by Contractor E
        - Job E was carried out in month M5 by Contractor E
        - Job E was carried out in month M8 by Contractor E
        - Job G was carried out in month M6 by Contractor G
        - Job H was carried out in month M4 by Contractor H
    --Project D--
        - Job F was carried out in month M3 by Contractor F
        - Job I was carried out in month M4 by Contractor G
        - Job D was carried out in month M2 by Contractor H
        - Job H was carried out in month M5 by Contractor H
    --Project E--
        - Job A was carried out in month M9 by Contractor A
        - Job J was carried out in month M8 by Contractor C
    --Project H--
        - Job A was carried out in month M8 by Contractor A
        - Job B was carried out in month M9 by Contractor E
        - Job I was carried out in month M11 by Contractor G
        - Job D was carried out in month M10 by Contractor H
    --Project I--
        - Job L was carried out in month M10 by Contractor A
        - Job K was carried out in month M12 by Contractor B
        - Job F was carried out in month M11 by Contractor F


Profit Margin is:  2165
```



```
Solution:  5
Projects Contracted:
    --Project A--
        - Job A was carried out in month M1 by Contractor A
        - Job B was carried out in month M2 by Contractor E
        - Job C was carried out in month M3 by Contractor K
    --Project C--
        - Job B was carried out in month M7 by Contractor E
        - Job E was carried out in month M5 by Contractor E
        - Job E was carried out in month M8 by Contractor E
        - Job G was carried out in month M6 by Contractor G
        - Job H was carried out in month M4 by Contractor H
    --Project D--
        - Job F was carried out in month M3 by Contractor F
        - Job I was carried out in month M4 by Contractor G
        - Job D was carried out in month M2 by Contractor H
        - Job H was carried out in month M5 by Contractor H
    --Project E--
        - Job A was carried out in month M9 by Contractor A
        - Job J was carried out in month M8 by Contractor C
    --Project H--
        - Job A was carried out in month M8 by Contractor A
        - Job B was carried out in month M9 by Contractor E
        - Job I was carried out in month M11 by Contractor G
        - Job D was carried out in month M10 by Contractor H
    --Project I--
        - Job K was carried out in month M12 by Contractor B
        - Job L was carried out in month M10 by Contractor D
        - Job F was carried out in month M11 by Contractor F


Profit Margin is:  2165
There are 5 solutions
```