In [2]:
```
import sys
print(sys.executable)
```

```
/opt/anaconda3/envs/gba462p/bin/python
```

In [3]:
```
%pip install pulp
```

```
Collecting pulp
  Using cached pulp-3.3.0-py3-none-any.whl.metadata (8.4 kB)
Using cached pulp-3.3.0-py3-none-any.whl (16.4 MB)
Installing collected packages: pulp
Successfully installed pulp-3.3.0
Note: you may need to restart the kernel to use updated packages.
```

In [23]:
```
from pulp import LpMaximize, LpMinimize, LpProblem, LpStatus, lpSum, LpVariable, LpAffineExpression
import pandas as pd
import numpy as np
```

In [24]:
```
# Question 1

# Maximize the total bid points for all students given their assignments.
# Determine which internship assignments each student receives in the optimal solution.
```

In [43]:
```
df = pd.read_csv("HW1_BidData.csv")
```

In [44]:
```
students = ["Keven", "Ayush", "Zan", "Kim", "Daria", "Yogesh", "Bonita", "Elayne", "Segev", "Abeer", "Wei", "Yilin", "Jeroen", "Yasmeen", "Kristen"]
# or
students = df["Student"].tolist()
```

In [45]:
```
internships = ["Quest_Realty", "CTG_Associates", "Innova", "Celeron_Capital", "Helder_LLC", "Felsen_Davis", "Sienna_Financial_Group"]
# or
internships = [c for c in df.columns if c != "Student"]
```

In [46]:
```
bid = {}

for s in students:
    for i in internships:
        score = df.loc[df["Student"] == s, i].values[0]
        bid[(s, i)] = float(score)
```

In [47]:
```
model = LpProblem("Internship_Assignment", LpMaximize)
```

In [48]:
```
x = LpVariable.dicts("x", [(s, i) for s in students for i in internships], lowBound=0, upBound=1, cat="Binary")
```

In [49]:
```
model += lpSum(bid[(s, i)] * x[(s, i)] for s in students for i in internships)
```

In [50]:
```
for s in students:
    model += lpSum(x[(s, i)] for i in internships) == 2, f"TwoInternships_{s}"
```

In [51]:
```
for i in internships:
    model += lpSum(x[(s, i)] for s in students) <= 5, f"Cap_{i}"
```

In [52]:
```
model.solve()
```

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

command line - /opt/anaconda3/envs/gba462p/lib/python3.13/site-packages/pulp/apis/../solverdir/cbc/osx/i64/cbc /var/folders/b1/672tkw4x48s5j8_p8n3dz5c40000gn/T/c82ff081237646fa9087fc730f1a408d-pulp.mps -max -timeMode elapsed -branch -printingOptions all -solution /var/folders/b1/672tkw4x48s5j8_p8n3dz5c40000gn/T/c82ff081237646fa9087fc730f1a408d-pulp.sol (default strategy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 27 COLUMNS
At line 507 RHS
At line 530 BOUNDS
At line 636 ENDATA
Problem MODEL has 22 rows, 105 columns and 210 elements
Coin0001I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Continuous objective value is 1160 - 0.00 seconds
Cgl0004I processed model has 22 rows, 105 columns (105 integer (105 of which binary)) and 210 elements
Cutoff increment increased from 1e-05 to 4.9999
Cbc0038I Initial state - 0 integers unsatisfied sum - 0
Cbc0038I Solution found of -1160
Cbc0038I Before mini branch and bound, 105 integers at bound fixed and 0 continuous
Cbc0038I Mini branch and bound did not improve solution (0.01 seconds)
Cbc0038I After 0.01 seconds - Feasibility pump exiting with objective of -1160 - took 0.00 seconds
Cbc0012I Integer solution of -1160 found by feasibility pump after 0 iterations and 0 nodes (0.01 seconds)
Cbc0001I Search completed - best objective -1160, took 0 iterations and 0 nodes (0.01 seconds)
Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost
Cuts at root node changed objective from -1160 to -1160
Probing was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Gomory was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Knapsack was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Clique was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
FlowCover was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
TwoMirCuts was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
ZeroHalf was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Result - Optimal solution found

Objective value:                1160.00000000
Enumerated nodes:               0
Total iterations:               0
Time (CPU seconds):             0.01
Time (Wallclock seconds):       0.02

Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.01   (Wallclock seconds):       0.02
```

Out[52]: 1

In [56]:
```
# Maximize the total bid points for all students given their assignments.
```

In [57]:
```
print("Status:", LpStatus[model.status])
print("Optimal total bid points:", model.objective.value())
```

```
Status: Optimal
Optimal total bid points: 1160.0
```

In [58]:
```
# Determine which internship assignments each student receives in the optimal solution.
```

In [59]:
```
print("\nAssignments by student:")
for s in students:
    chosen = [i for i in internships if x[(s, i)].value() == 1]
    print(f"{s}: {chosen}")
```

```
Assignments by student:
Kevin: ['Quest_Realty', 'Felsen_Davis']
Ayush: ['Innova', 'Felsen_Davis']
Zan: ['Quest_Realty', 'CTG_Associates']
Kim: ['Celeron_Capital', 'Helder_LLC']
Daria: ['Quest_Realty', 'Innova']
Yogesh: ['CTG_Associates', 'Felsen_Davis']
Bonita: ['CTG_Associates', 'Celeron_Capital']
Elayne: ['Innova', 'Felsen_Davis']
Segev: ['Innova', 'Felsen_Davis']
Abeer: ['Celeron_Capital', 'Sienna_Financial_Group']
Wei: ['Celeron_Capital', 'Helder_LLC']
Yilin: ['CTG_Associates', 'Helder_LLC']
Jeroen: ['Quest_Realty', 'Helder_LLC']
Yasmeen: ['Helder_LLC', 'Sienna_Financial_Group']
Kristen: ['Quest_Realty', 'Celeron_Capital']
```

In [55]:
```
print("\nCounts by internship:")
for i in internships:
    count_i = sum(x[(s, i)].value() for s in students)
    print(f"{i}: {int(count_i)}")
```

```
Counts by internship:
Quest_Realty: 5
CTG_Associates: 4
Innova: 4
Celeron_Capital: 5
Helder_LLC: 5
Felsen_Davis: 5
Sienna_Financial_Group: 2
```

In [60]:
```
# The optimal solution assigns each student to exactly two internships and respects the capacity constraint
# (no internship has more than 5 students). The maximum total bid points is 1160.
```

In [ ]:
```

```

In [ ]:
```

```

In [61]:
```
# Question 2

# Build a linear programming model to identify the production levels for MNC's four products that maximize total VM.

# If MNC wanted to decrease the production of any product, which product would you recommend and why

# Which machine capacities would you recommend the company look into expanding?
# If they can only expand one machine capacity, which machine should they target?
```

In [62]:
```
from pulp import LpMaximize, LpProblem, LpStatus, lpSum, LpVariable, PULP_CBC_CMD
import pandas as pd
import numpy as np
```

In [82]:
```
products = ['Whole', 'Cluster', 'Crunch', 'Roasted']
machines = ['Hulling', 'Roasting', 'Coating', 'Packaging']
```

In [83]:
```
vm = {'Whole': 1.93, 'Cluster': 1.04, 'Crunch': 1.15, 'Roasted': 1.33}
```

In [84]:
```
nut_pct = {'Whole': 0.60, 'Cluster': 0.40, 'Crunch': 0.20, 'Roasted': 1.00}
choc_pct = {p: 1 - nut_pct[p] for p in products}
```

In [85]:
```
nuts_available = 1100
choc_available = 800
```

In [86]:
```
cap_minutes = 60 * 60
```

In [87]:
```
minutes = {
    'Hulling':    {'Whole': 1.00, 'Cluster': 1.00, 'Crunch': 1.00, 'Roasted': 1.00},
    'Roasting':   {'Whole': 2.00, 'Cluster': 1.50, 'Crunch': 1.00, 'Roasted': 4.00},
    'Coating':    {'Whole': 1.00, 'Cluster': 0.70, 'Crunch': 0.20, 'Roasted': 0.00},
    'Packaging':  {'Whole': 2.50, 'Cluster': 1.60, 'Crunch': 1.25, 'Roasted': 1.00},
}
```

In [88]:
```
model = LpProblem("MolokaiNutCompany_Q2", LpMaximize)
```

In [89]:
```
prod = LpVariable.dicts("Prod", products, lowBound=0)
```

In [90]:
```
model += lpSum(vm[p] * prod[p] for p in products), "Total_VM"
```

In [91]:
```
model += prod['Whole'] >= 1000, "Dem_Whole_min"
model += prod['Cluster'] >= 400, "Dem_Cluster_min"
model += prod['Cluster'] <= 500, "Dem_Cluster_max"
model += prod['Crunch'] <= 150, "Dem_Crunch_max"
model += prod['Roasted'] <= 200, "Dem_Roasted_max"
```

In [92]:
```
model += lpSum(nut_pct[p]  * prod[p] for p in products) <= nuts_available, "Nuts"
model += lpSum(choc_pct[p] * prod[p] for p in products) <= choc_available, "Chocolate"
```

In [93]:
```
for m in machines:
    model += lpSum(minutes[m][p] * prod[p] for p in products) <= cap_minutes, f"Cap_{m}"
```

In [94]:
```
model.solve(PULP_CBC_CMD(msg=False))

print("Status:", LpStatus[model.status])
print("Max total VM:", model.objective.value())

print("\nOptimal production (lbs):")
for p in products:
    print(f"{p}: {prod[p].value()}")
```

```
Status: Optimal
Max total VM: 2839.075

Optimal production (lbs):
Whole: 1030.0
Cluster: 400.0
Crunch: 150.0
Roasted: 197.5
```

In [95]:
```
print("\nMachine constraints (slack & shadow price):")
for m in machines:
    con = model.constraints[f"Cap_{m}"]
    print(f"{m:10s} | slack = {con.slack:8.2f} | pi (shadow price) = {con.pi:.6f}")
```

```
Machine constraints (slack & shadow price):
Hulling    | slack =  1822.50 | pi (shadow price) = -0.000000
Roasting   | slack =    -0.00 | pi (shadow price) = 0.174375
Coating    | slack =  2260.00 | pi (shadow price) = -0.000000
Packaging  | slack =    -0.00 | pi (shadow price) = 0.632500
```

In [96]:
```
# Answer for 2A

# Using a linear programming model, the optimal weekly production plan is:
# Whole: 1030.0 lbs
# Cluster: 400.0 lbs
# Crunch: 150.0 lbs
# Roasted: 197.5 lbs

# This plan yields a maximum total variable margin of 2839.075.
```

In [97]:
```
# Answer for 2B

# If MNC decides to reduce production, the best candidate to reduce first is Cluster.
# In the optimal solution, Cluster is produced exactly at its minimum required level (400 lbs),
# which indicates it is being produced primarily to satisfy the marketing constraint rather than
# because it is the most profitable use of limited resources.

# Reducing Cluster would free capacity in the binding operations,
# allowing production to shift toward higher-value products.
```

In [98]:
```
# Answer for 2C

# MNC should expand Packaging capacity.

# Packaging is a binding constraint (slack = 0) and has the largest shadow price, π = 0.6325.
# This means that each additional minute of Packaging capacity increases the total variable margin by about 0.6325.
```

In [ ]:
```

```

In [ ]:
```

```

In [99]:
```
# Question 3

# Determine the optimal leasing plan for the production company, in order to have all the storage space
# it needs in each month at the minimum cost.

# Min(Costs)
```

In [100]:
```
from pulp import LpProblem, LpVariable, LpMinimize, lpSum, LpStatus, PULP_CBC_CMD
```

In [101]:
```
T = [1, 2, 3, 4, 5] # months
D = [1, 2, 3, 4, 5] #possible lease durations (months)
```

In [102]:
```
demand = {1: 30000, 2: 20000, 3: 40000, 4: 10000, 5: 50000}
```

In [103]:
```
cost = {1: 65, 2: 100, 3: 135, 4: 160, 5: 190}
```

In [104]:
```
valid_td = [(t, d) for t in T for d in D if t + d - 1 <= 5]
```

In [106]:
```
model = LpProblem("Warehouse_Leasing_Q3", LpMinimize)
```

In [107]:
```
y = LpVariable.dicts("LeaseSqft", valid_td, lowBound=0)
```

In [108]:
```
model += lpSum(cost[d] * y[(t, d)] for (t, d) in valid_td), "Total_Cost"
```

In [109]:
```
for m in T:
    model += lpSum(
        y[(t, d)]
        for (t, d) in valid_td
        if t <= m <= (t + d - 1)
    ) >= demand[m], f"Demand_M{m}"
```

In [110]:
```
model.solve(PULP_CBC_CMD(msg=False))
```

Out[110]: 1

In [111]:
```
print("Status:", LpStatus[model.status])
print("Min total cost:", model.objective.value())
```

```
Status: Optimal
Min total cost: 7650000.0
```

In [112]:
```
print("\nLeases selected (only non-zero):")
for (t, d) in valid_td:
    val = y[(t, d)].value()
    if val is not None and val > 1e-6:
        end_month = t + d - 1
        print(f"Start M{t}, duration {d} month(s) (covers M{t}-M{end_month}), sqft = {val:,.0f}")
```

```
Leases selected (only non-zero):
Start M1, duration 5 month(s) (covers M1-M5), sqft = 30,000
Start M3, duration 1 month(s) (covers M3-M3), sqft = 10,000
Start M5, duration 1 month(s) (covers M5-M5), sqft = 20,000
```

In [113]:
```
print("\nMonthly coverage check:")
for m in T:
    coverage = sum(
        y[(t, d)].value()
        for (t, d) in valid_td
        if t <= m <= (t + d - 1)
    )
    print(f"Month {m}: coverage = {coverage:,.0f} | demand = {demand[m]:,.0f} | extra = {coverage - demand[m]:,.0f}")
```

```
Monthly coverage check:
Month 1: coverage = 30,000 | demand = 30,000 | extra = 0
Month 2: coverage = 30,000 | demand = 20,000 | extra = 10,000
Month 3: coverage = 40,000 | demand = 40,000 | extra = 0
Month 4: coverage = 30,000 | demand = 10,000 | extra = 20,000
Month 5: coverage = 50,000 | demand = 50,000 | extra = 0
```

In [114]:
```
# The linear programming model returns an optimal solution with a minimum total cost of $7,650,000.

# Optimal leasing plan:
# Start Month 1, lease for 5 months (covers Months 1-5): 30,000 sqft
# Start Month 3, lease for 1 month (covers Month 3 only): 10,000 sqft
# Start Month 5, lease for 1 month (covers Month 5 only): 20,000 sqft
```

In [ ]:
```

```