**Ohio University**

**Industrial & Systems Engineering**

**Research & Virtual Environment Setup**

**Felipe Aros-Vera, Ph.D Office Room: Stocker 277**

**Email:** [**aros@ohio.edu**](mailto:aros@ohio.edu) **Laboratory Room: Stocker 271**

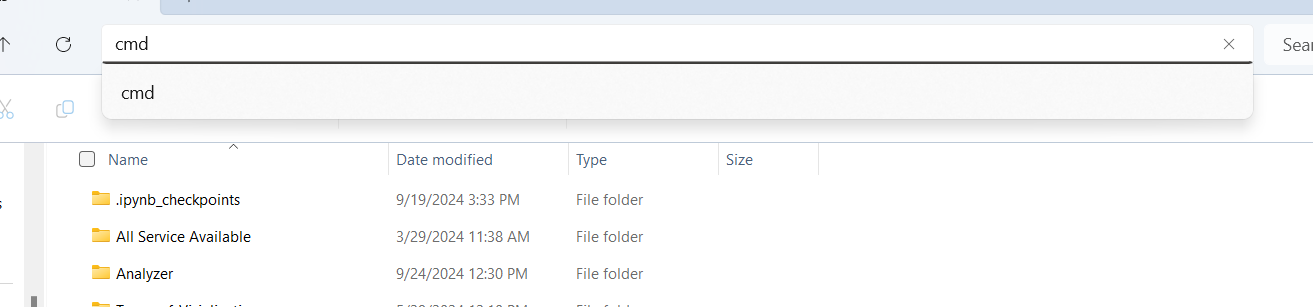
**Downloading Jupyter Notebook & Running Pandas w/ Gurobi**

**In this how-to you will learn how to install Jupyter notebook a create a simple file. This how-to is done on windows 11, but the setup should be similar for both Linux and Mac.**

**Setting up Python & Jupyter:**

**For Windows:**

* First go to [Download Python | Python.org](https://www.python.org/downloads/) and download the latest version for your device by clicking the yellow box:
* 
* Make sure that for these latest versions you are not running Windows 7 or earlier for Python to work.
* You can follow the other links to install a more specific version, but the button as shown above should be enough for this tutorial.
* Run the exe and **MAKE SURE YOU CHECK BOTH BOXES AT THE BOTTOM** as shown below and click “Install Now”: A screenshot of a computer

  Description automatically generated
* After that open command prompt and then check the version of python:
* **Command:** python --version
* Once you have confirmed that python is installed then run the command:
* **Command:** pip install jupyter
* Once that is completed, close the terminal and then in file explorer navigate to the directory that you wish to store the project.
* Once you are there you can just click on the part that shows which directory you are in. Then, when it shows the direct path to the directory. Delete it and just type cmd and press enter.
* 
* This will open the terminal in that directory. From there you can just type in the command to run Jupyter Notebook:
* **Command:** jupyter notebook
* This command will open jupyter notebook in your browser for you to use. To keep the program running you must keep the terminal that you opened with it running.

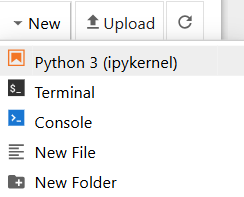
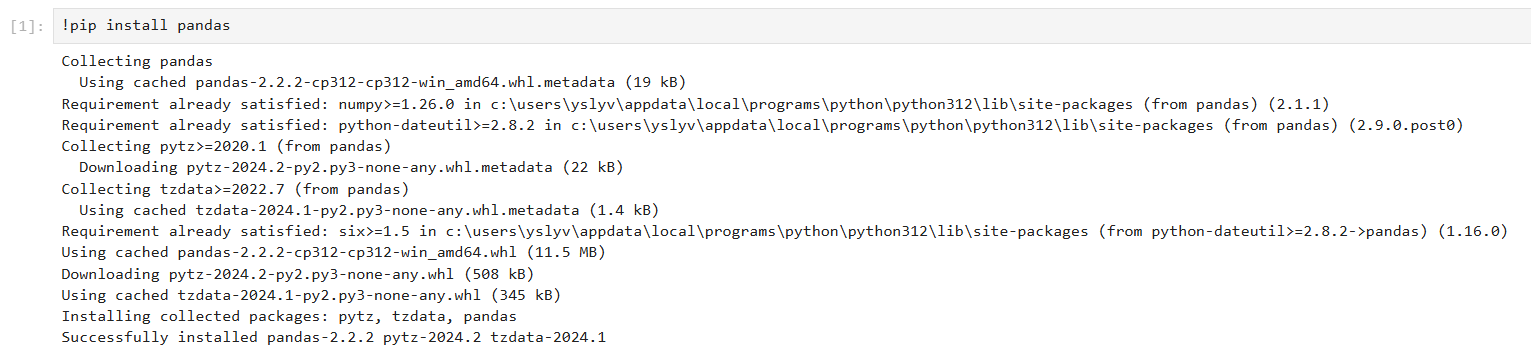
**For Mac OS:**

* First go to [Download Python | Python.org](https://www.python.org/downloads/) and download the latest version for your device
* Open the downloaded .pkg file and follow the instructions. Make sure to check the option to add python to your PATH
* Next open the Terminal (it can b found in Applications > Utilities) and check the version of Python:
* **Command:** python3 –version
* Once Python is for sure installed, run:
* **Command:** pip3 install jupyter
* Afterwards navigate to your project directory in the terminal and start Jupyter notebook:
* **Command:** jupyter notebook
* This will open Jupyter Notebook in your default browser. Keep the terminal open and running to keep Jupyter Notebook running.

**For Linux:**

* Most distrubutions come with Python pre-installed. Open a terminal and check to see:
* **Command:** python3 –version
* If Python is not installed, you can install it using your package manager.
* Ubuntu example:
* **Command:** sudo apt update
* **Command:** sudo apt install python3
* If pip is not installed, then install it using:
* **Command:** sudo apt install python3-pip
* Once python and pip are installed, run:
* **Command:** pip2 install jupyter
* After that, navigate to your project directory in the terminal and then type in:
* **Command:** jupyter notebook

**Running and using Python Pandas Library:**

* Open Jupyter Notebook in your project directory and then create a new Python 3 ipykernal.
* 
* Once that file is created, there is already a cell created. Within that cell paste the following command in with an “!” Infront to show that it is a command as shown below:
* **Command:** install pandas
* ****
* When everything is successfully installed, delete the command in the first cell and type in:

import pandas as pd

* Underneath in a separate cell create a simple dataframe. A dataframe is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns):
* You can create a new cell by pressing the new cell button at the top of the page on the navbar as shown: ()

data = {

'Name': ['Alice', 'Bob', 'Charlie'],

'Age': [25, 30, 35],

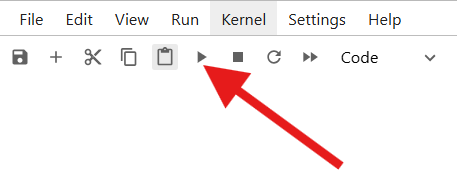
'City': ['New York', 'Los Angeles', 'Chicago']

}

* After that in a new cell add this line of code to save your dataframe to a variable and then print it:

df = pd.DataFrame(data)

print(df)

* Then you can just click on the first cell and then find the run button ()
* 
* **A screenshot of a computer

  Description automatically generated**
* That’s how you make a simple dataframe with pandas, for specific things refer to the documentation: [pandas documentation — pandas 2.2.3 documentation (pydata.org)](https://pandas.pydata.org/docs/)

**Gurobi:**

* Set up Gurobi on your machine. (See separate tutorial)
* Once you have gone through that open a new or existing file in Jupyter notebook.
* Run the command:
* **Command:** pip install gurobipy

**Optimization Model 1 (Knapsack Problem):**

* Here is a simple knapsack problem model: [Knapsack Problem - Optimization in Python with Gurobi (Part 4) (youtube.com)](https://www.youtube.com/watch?v=0AeGqnM04yc)
* First you set up the arrays containing the information:

# Step 1 Create the weights and values

W = [4,2,5,4,5,2,3,5]

V = [10,5,18,12,15,3,2,8]

C = 15

N = len(W)

* Then you import the Gurobipy Package:

# Step 2 Importing gurobipy package

from gurobipy import \*

* Now create the model:

# Step 3 Create an optimization model

knapsack\_model = Model('knapsack')

* Add the decision variables:

# Step 4 Add decision variables

#addVars (\*indicies, lb=0.0, ub=float('inf'), obj=0.0, vtype=GRB.CONTINUOUS, name="")

X = knapsack\_model.addVars(N, vtype=GRB.BINARY, name="x")

* Define the objective function:

# Step 5 Define the objective function

obj\_fn = sum(V[i]\*X[i] for i in range(N))

knapsack\_model.setObjective(obj\_fn, GRB.MAXIMIZE)

* Add the constraints:

# Step 6 Add the constraints

knapsack\_model.addConstr(sum(W[i]\*X[i] for i in range(N)) <= C)

* Solve the mode and output the solution:

# Step 7 Solve the model and output the solution

knapsack\_model.setParam('OutputFlag', False)

knapsack\_model.optimize()

print('Optimization is done. Objective Function Value: %.2f' % knapsack\_model.objVal)

for v in knapsack\_model.getVars():

print('%s: %g' % (v.varName, v.x))

**Optimization Model 2 (Polynomial Optimization):**

* Decision variables are x, y and z
* Objective Function: and Constraint:
* First set up some imports:

import gurobipy as gp  
from gurobipy import GRB

* Create the model:

model = gp.Model("simple\_optimization")

* Create the variables:

x = model.addVar(name="x")  
y = model.addVar(name="y")

z = model.addVar(name="z")

* Set the objective function:

model.setObjective(3 \* x \* x + y \* y - 2 \* z \* z, GRB.MAXIMIZE)

* Add the constraints:

model.addConstr(3 \* x + 2 \* y - 8 \* z == -50, "c0")

* Optimize the model:

model.optimize()

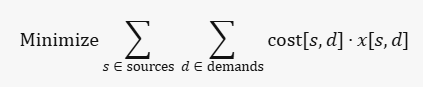
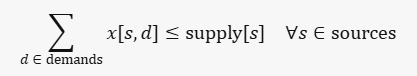
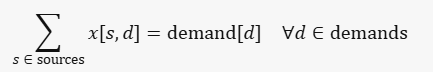
* Print the results:

# Print the results  
if model.status == GRB.OPTIMAL:  
 for v in model.getVars():  
 print(f'{v.varName}: {v.x}')  
 print(f'Objective: {model.objVal}')  
else:  
 print("No optimal solution found.")

**Optimization Model 3 (Transportation Problem):**

* 2 source locations, 3 demand locations
* 2 sets:
* Sources = {S1, S2} The set of source points
* Demands = {D1, D2, D3} The set of demand points
* Table with costs between each source and demand point:

|  |  |  |  |
| --- | --- | --- | --- |
|  | D1 | D2 | D2 |
| S1 | 8 | 6 | 10 |
| S2 | 9 | 12 | 7 |

* Objective: minimize the total transportation cost.
* 
* Constraint 1: The total amount of goods transported from each source should not exceed its supply capacity.
* 
* Constraint 2: The total amount of goods transported to each demand point should meet its demand.
* 
* Decision variables: (x[s, d]) represent the amount of goods transported from source (s) to demand point (d).
* First import the libraries:

import gurobipy as gp

from gurobipy import GRB

* Define the data:

# Define the data

sources = ['S1', 'S2']

demands = ['D1', 'D2', 'D3']

supply = {'S1': 20, 'S2': 30}

demand = {'D1': 10, 'D2': 25, 'D3': 15}

cost = {

('S1', 'D1'): 8, ('S1', 'D2'): 6, ('S1', 'D3'): 10,

('S2', 'D1'): 9, ('S2', 'D2'): 12, ('S2', 'D3'): 7

}

* Create a new model:

# Create a new model

model = gp.Model("transportation")

* Create the variables:

# Create variables

x = model.addVars(sources, demands, name="x")

* Set the objective:

# Set objective

model.setObjective(gp.quicksum(cost[s, d] \* x[s, d] for s in sources for d in demands), GRB.MINIMIZE)

* Add constraints:

# Add supply constraints

model.addConstrs((gp.quicksum(x[s, d] for d in demands) <= supply[s] for s in sources), name="Supply")

# Add demand constraints

model.addConstrs((gp.quicksum(x[s, d] for s in sources) == demand[d] for d in demands), name="Demand")

* Optimize the model:

# Optimize model

model.optimize()

* Print results:

# Print the results

if model.status == GRB.OPTIMAL:

for s in sources:

for d in demands:

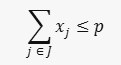
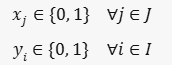
if x[s, d].x > 0:

print(f"Transport {x[s, d].x} units from {s} to {d} at cost {cost[s, d]} per unit")

else:

print("No optimal solution found")

**Optimization Model 4 (MCLP):**

* 2 Sets:
* ( I ) Set of demand points
* ( J ) Set of potential Facility locations
* 4 Parameters:
* - Distance between demand point ( I ) and facility location ( J )
* - Set of facility locations that can cover demand point ( I )
* – Number of facilities to be located
* - Weight (or demand) of demand point ( I )
* 2 Decision Variables:
* - Binary variable indicating whether a facility is located at site ( J )
* - Binary variable indicating whether demand point ( I ) is covered.
* Objective:
* Maximize the total covered demand
* 
* 3 Constraints:
* Each demand point ( I ) can be covered if at least one facility is located within its coverage distance
* 
* The number of facilities located should not exceed ()
* 
* Binary constraints for decision variables:
* 
* Code:
* First add the imports and define the data:

import gurobipy as gp  
from gurobipy import GRB  
  
# Define the data  
I = ['D1', 'D2', 'D3', 'D4']  
J = ['F1', 'F2', 'F3']  
d = {  
 ('D1', 'F1'): 2, ('D1', 'F2'): 4, ('D1', 'F3'): 5,  
 ('D2', 'F1'): 3, ('D2', 'F2'): 2, ('D2', 'F3'): 6,  
 ('D3', 'F1'): 4, ('D3', 'F2'): 3, ('D3', 'F3'): 2,  
 ('D4', 'F1'): 5, ('D4', 'F2'): 6, ('D4', 'F3'): 3  
}  
coverage\_distance = 3  
S = {i: [j for j in J if d[i, j] <= coverage\_distance] for i in I}  
p = 2  
w = {'D1': 10, 'D2': 20, 'D3': 30, 'D4': 40}

* Next create the model and variables:

# Create a new model  
model = gp.Model("MCLP")  
  
# Create variables  
x = model.addVars(J, vtype=GRB.BINARY, name="x")  
y = model.addVars(I, vtype=GRB.BINARY, name="y")

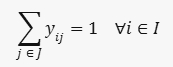
* Now set the objective and add the constraints:

# Set objective  
model.setObjective(gp.quicksum(w[i] \* y[i] for i in I), GRB.MAXIMIZE)  
  
# Add constraints  
model.addConstrs((y[i] <= gp.quicksum(x[j] for j in S[i]) for i in I), name="Coverage")  
model.addConstr(gp.quicksum(x[j] for j in J) <= p, name="FacilityLimit")

* Finally optimize the model and print the results:

# Optimize model  
model.optimize()  
  
# Print the results  
if model.status == GRB.OPTIMAL:  
 print("Optimal solution found:")  
 for j in J:  
 if x[j].x > 0.5:  
 print(f"Facility located at {j}")  
 for i in I:  
 if y[i].x > 0.5:  
 print(f"Demand point {i} is covered")  
else:  
 print("No optimal solution found")

**Optimization Model 5 (P-Median):**

* 2 Sets:
* ( I ) Set of demand points
* ( J ) Set of potential facility locations
* Parameters:
* - Distance between demand point ( I ) and facility location ( J )
* – Number of facilities to be located
* 2 Decision Variables:
* - Binary variable indicating whether a facility is located at site ( J )
* - Binary variable indicating whether demand point ( I ) is assigned to facility ( J )
* Objective:
* Minimize the total distance between demand points and their assigned facilities
* 
* 4 Constraints:
* Each demand point ( I ) must be assigned to exactly one facility
* 
* A demand point ( I ) can only be assigned to a facility ( J ) if that facility is open
* 
* The number of facilities located should be exactly ()
* 
* Binary constraints for decision variables
* 
* Code:
* Set up the imports and define the data:

import gurobipy as gp  
from gurobipy import GRB  
  
# Define the data  
I = ['D1', 'D2', 'D3', 'D4']  
J = ['F1', 'F2', 'F3']  
d = {  
 ('D1', 'F1'): 2, ('D1', 'F2'): 4, ('D1', 'F3'): 5,  
 ('D2', 'F1'): 3, ('D2', 'F2'): 2, ('D2', 'F3'): 6,  
 ('D3', 'F1'): 4, ('D3', 'F2'): 3, ('D3', 'F3'): 2,  
 ('D4', 'F1'): 5, ('D4', 'F2'): 6, ('D4', 'F3'): 3  
}  
p = 2

* Now create the model and the variables:

# Create a new model  
model = gp.Model("p-median")  
  
# Create variables  
x = model.addVars(J, vtype=GRB.BINARY, name="x")  
y = model.addVars(I, J, vtype=GRB.BINARY, name="y")

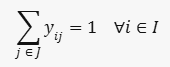
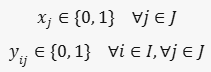
* Next define the objective and add the constraints:

# Set objective  
model.setObjective(gp.quicksum(d[i, j] \* y[i, j] for i in I for j in J), GRB.MINIMIZE)  
  
# Add constraints  
model.addConstrs((gp.quicksum(y[i, j] for j in J) == 1 for i in I), name="Assign")  
model.addConstrs((y[i, j] <= x[j] for i in I for j in J), name="Open")  
model.addConstr(gp.quicksum(x[j] for j in J) == p, name="FacilityCount")

* Now optimize the model and print the results:

# Optimize model  
model.optimize()  
  
# Print the results  
if model.status == GRB.OPTIMAL:  
 print("Optimal solution found:")  
 for j in J:  
 if x[j].x > 0.5:  
 print(f"Facility located at {j}")  
 for i in I:  
 for j in J:  
 if y[i, j].x > 0.5:  
 print(f"Demand point {i} is assigned to facility {j}")  
else:  
 print("No optimal solution found")

**Optimization Model 6 (P-Center):**

* 2 Sets:
* ( I ) Set of demand points
* ( J ) Set of potential facility locations
* Parameters:
* - Distance between demand point ( I ) and facility location ( J )
* – Number of facilities to be located
* Decision Variables:
* - Binary variable indicating whether a facility is located at site ( J )
* - Binary variable indicating whether demand point ( I ) is assigned to facility ( J )
* – Continuous variable representing the maximum distance any demand point has to travel to its nearest facility
* Objective:
* Minimize the maximum distance any demand point has to travel to its nearest facility
* 
* 5 Constraints:
* Each demand point ( I ) must be assigned to exactly one facility
* 
* A demand point ( I ) can only be assigned to a facility ( J ) if that facility is open
* 
* The number of facilities should be exactly ( )
* 
* The distance from any demand point to its assigned facility should not exceed ()
* 
* Binary constraints for decision variables
* 
* Code:
* First set up the imports and define the data:

import gurobipy as gp  
from gurobipy import GRB  
  
# Define the data  
I = ['D1', 'D2', 'D3', 'D4']  
J = ['F1', 'F2', 'F3']  
d = {  
 ('D1', 'F1'): 2, ('D1', 'F2'): 4, ('D1', 'F3'): 5,  
 ('D2', 'F1'): 3, ('D2', 'F2'): 2, ('D2', 'F3'): 6,  
 ('D3', 'F1'): 4, ('D3', 'F2'): 3, ('D3', 'F3'): 2,  
 ('D4', 'F1'): 5, ('D4', 'F2'): 6, ('D4', 'F3'): 3  
}  
p = 2

* Next create the model and the variables:

# Create a new model  
model = gp.Model("p-center")  
  
# Create variables  
x = model.addVars(J, vtype=GRB.BINARY, name="x")  
y = model.addVars(I, J, vtype=GRB.BINARY, name="y")  
z = model.addVar(vtype=GRB.CONTINUOUS, name="z")

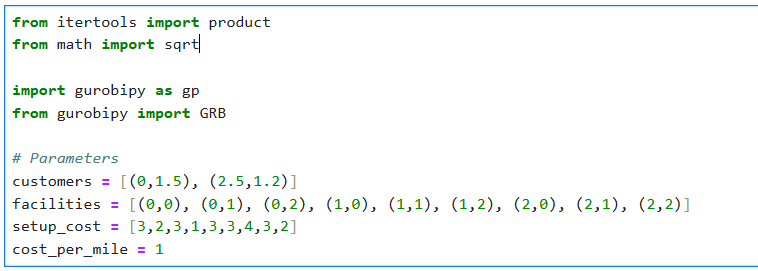
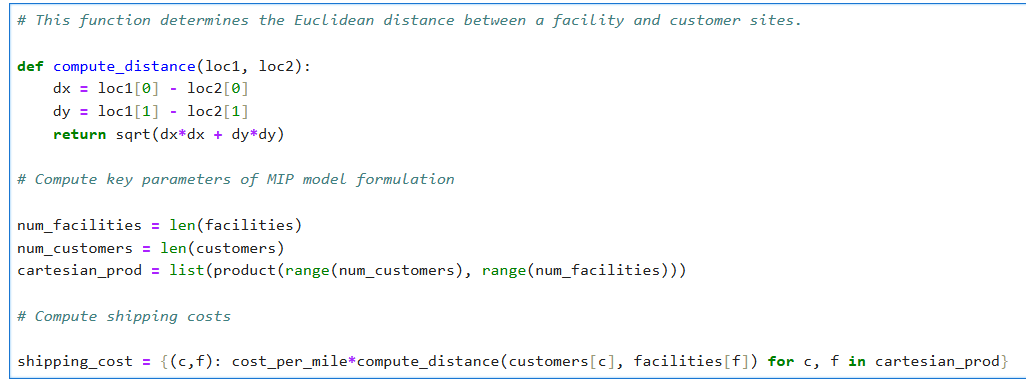
* Now set the objective and add the constraints:

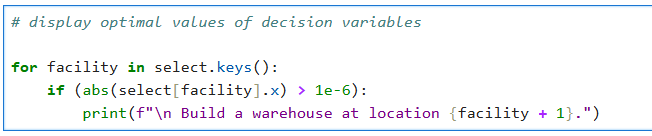
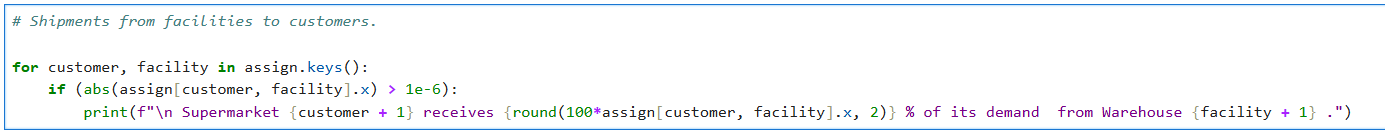
# Set objective  
model.setObjective(z, GRB.MINIMIZE)  
  
# Add constraints  
model.addConstrs((gp.quicksum(y[i, j] for j in J) == 1 for i in I), name="Assign")  
model.addConstrs((y[i, j] <= x[j] for i in I for j in J), name="Open")  
model.addConstr(gp.quicksum(x[j] for j in J) == p, name="FacilityCount")  
model.addConstrs((d[i, j] \* y[i, j] <= z for i in I for j in J), name="MaxDistance")

* Finally optimize the model and print the results:

# Optimize model  
model.optimize()  
  
# Print the results  
if model.status == GRB.OPTIMAL:  
 print("Optimal solution found:")  
 for j in J:  
 if x[j].x > 0.5:  
 print(f"Facility located at {j}")  
 for i in I:  
 for j in J:  
 if y[i, j].x > 0.5:  
 print(f"Demand point {i} is assigned to facility {j}")  
 print(f"Maximum distance: {z.x}")  
else:  
 print("No optimal solution found")

**Extra Optimization Model:**

* Here is another example model: [facility\_location.ipynb - Colab (google.com)](https://colab.research.google.com/github/Gurobi/modeling-examples/blob/master/facility_location/facility_location.ipynb#scrollTo=RFrCWRre7QQl)
* Code is found in the link
* First the imports and values that will be used are setup:
* ****
* Then needed functions alongside key parameters:
* ****
* The model is then setup, the name of the model, decision variables, constraints and objective functions. And then Gurobi begins the optimization process:
* **A white background with text

  Description automatically generated with medium confidence**
* Once the model finishes it will display the optimal build location:
* ****
* And display what percentage of shipments are supplied by which warehouses
* 
* This is the basic structure of the facility location problem as a MIP model.