L’obiettivo di questo lavoro è quello di costruire una semplice metodologia di Supply Chain Network Design che tenga conto della fluttuazione della domanda.
L'ottimizzazione della supply chain sfrutta al meglio l'analisi dei dati per trovare una combinazione ottimale di fabbriche e centri di distribuzione per soddisfare la domanda dei clienti. In molti software e soluzioni sul mercato, la struttura alla base è formata da un modello di programmazione lineare. Alcuni di questi trovano la giusta allocazione delle fabbriche per soddisfare la domanda e ridurre al minimo i costi assumendo una domanda costante.
Quello che, a questo punto, ci si chiede è cosa succede quando la domanda è fluttuante.
La rete potrebbe perdere robustezza, soprattutto quando si ha una stagionalità molto alta della domanda (e-commerce, cosmesi, fast fashion), quindi, sostanzialmente, quando si hanno dei periodi ad alta domanda e periodi a bassa domanda.
Quello che ci si propone di fare è di costruire una semplice metodologia per definire una Supply Chain Network robusta usando la simulazione Monte Carlo nel linguaggio di programmazione Python.
Il capo della Supply Chain Management di un’azienda internazionale manifatturiera vuole ridefinire la Supply Chain Network per i prossimi cinque anni. Quindi, si suppongono le seguenti caratteristiche per il mercato.
Per quanto riguarda la domanda, questa proviene da cinque mercati diversi secondo la seguente configurazione.

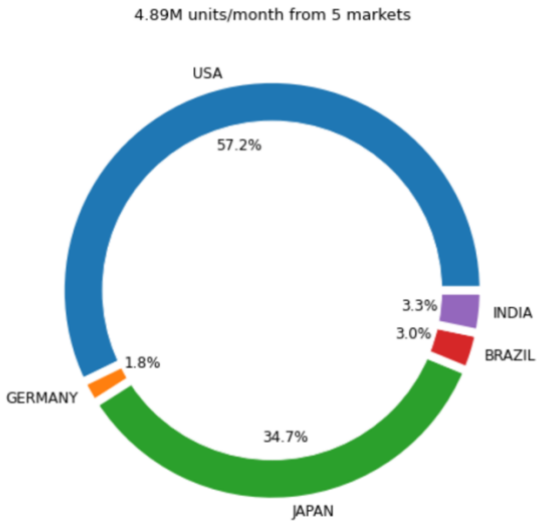


Figura 8: Domanda del mercato

Per quanto riguarda la capacità della supply chain, si possono aprire fabbriche nei cinque mercati. È possibile scegliere tra strutture a bassa e alta capacità. Inoltre, ciascuna apertura di una fabbrica in questi diversi cinque mercati, avrà costi fissi diversi, secondo il seguente schema.

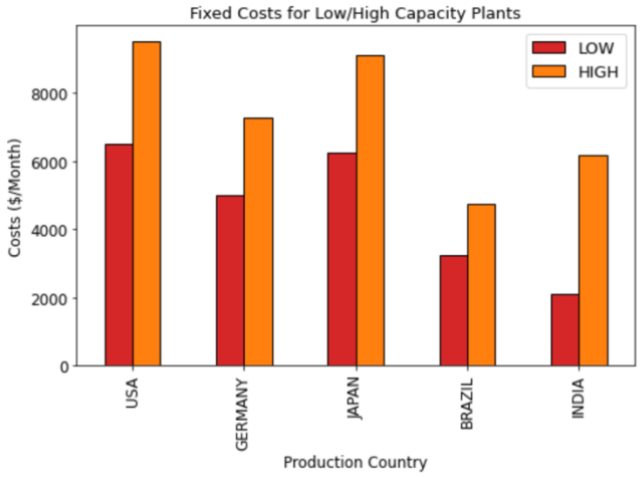


Figura 9: Costi fissi delle fabbriche

Di conseguenza, il costo totale di produzione sarà dato dalla somma di Costi Variabili e Costi Fissi, caratterizzando, per lo scenario in questione, i seguenti costi di produzione.

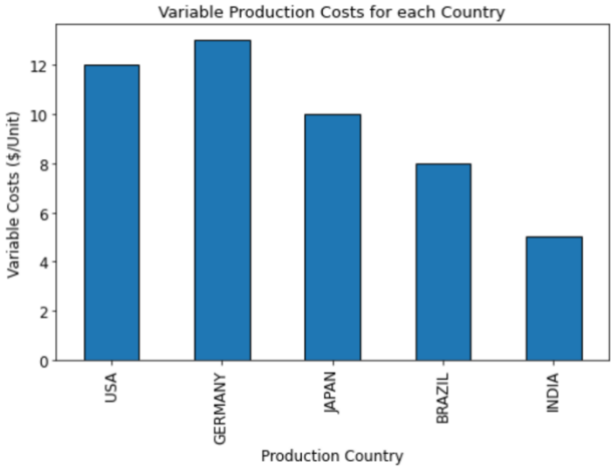


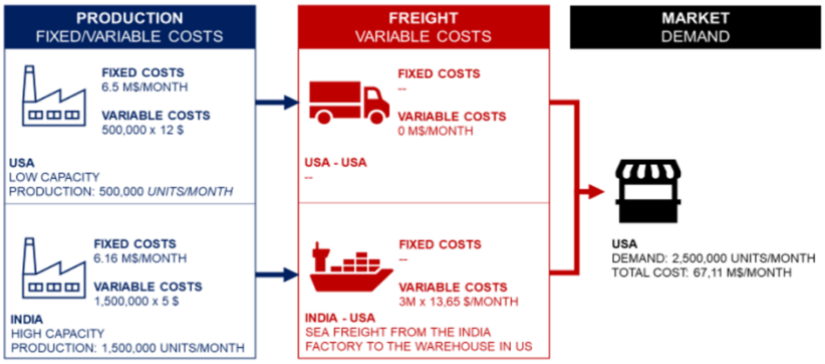
Figura 10: Costi totali di produzione

Inoltre, bisogna considerare il costo per spedire un container dal paese XXX al paese YYY. Il costo viene modellato in accordo al seguente schema.

	USA	GERMANY	JAPAN	BRAZIL	INDIA
USA	0	12250	1100	16100	8778
GERMANY	13335	0	8617	20244	10073
JAPAN	15400	22750	0	43610	14350
BRAZIL	16450	22050	28000	0	29750
INDIA	13650	15400	24500	29400	0

Figura 11: Costi per il trasporto ($/container)

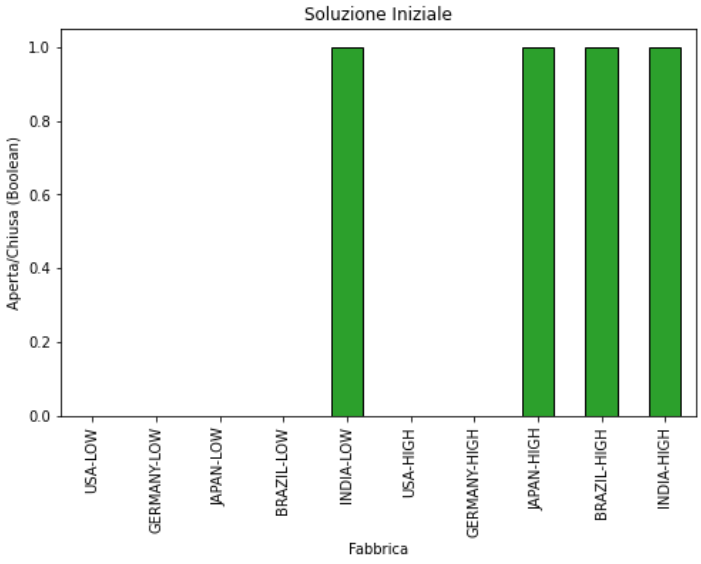
Quindi, si può individuare la caratterizzazione dei prezzi totali per produrre e spedire.



A questo punto, quello che si va a comporre è un classico problema di programmazione lineare con una funzione obiettivo (minimizzazione dei costi) e vincoli (sulla domanda di mercato e sulla capacità di produzione). La domanda che ci pone è, evidentemente, dove conviene aprire le fabbriche basandosi sulla fluttuante domanda di mercato. Quindi, si va a simulare lo scenario iniziale. Di seguito il codice Python di tale implementazione.

```
# Definisco le variabili decisionali
loc = ['USA', 'GERMANY', 'JAPAN', 'BRAZIL', 'INDIA']
size = ['LOW', 'HIGH']
plant_name = [(i,s) for s in size for i in loc]
prod_name = [(i,j) for i in loc for j in loc]

# Inizializzazione
model = pulp.LpProblem("Capacitated Plant Location Model", pulp.LpMinimize)


# Creo le variabili decisionali
x = pulp.LpVariable.dicts("production_", prod_name,
                      lowBound=0, upBound=None, cat='continuous')
y = pulp.LpVariable.dicts("plant_",
                      plant_name, cat='Binary')

# Defisco la funzione obiettivo
model += (pulp.lpSum([fixed_costs.loc[i,s] * y[(i,s)] * 1000 for s in size for i in loc])

          + pulp.lpSum([var_cost.loc[i,j] * x[(i,j)]   for i in loc for j in loc]))

# Aggiungo i vincoli
for j in loc:
    model += pulp.lpSum([x[(i, j)] for i in loc]) == demand.loc[j,'Demand']
for i in loc:
    model += pulp.lpSum([x[(i, j)] for j in loc]) <= pulp.lpSum([cap.loc[i,s]*y[(i,s)] * 1
000
                                                             for s in size])


# Risolvo il modello
model.solve()
print("Status: {}".format(pulp.LpStatus[model.status]))
print("Total Costs: {:,} ($/Month)".format(int(pulp.value(model.objective))))

# Plot dei risultati
df_bool = pd.DataFrame(data = [y[plant_name[i]].varValue for i in range(len(plant_name))],
     index = [i + '-' + s for s in size for i in loc],
                          columns = ['Plant Opening'])
df_bool
```

Output:



	Apertura Fabbrica
USA-LOW	0.0
GERMANY-LOW	0.0
JAPAN-LOW	0.0
BRAZIL-LOW	0.0
INDIA-LOW	1.0
USA-HIGH	0.0
GERMANY-HIGH	0.0
JAPAN-HIGH	1.0
BRAZIL-HIGH	1.0
INDIA-HIGH	1.0

Basandosi sui dati (sintetici) a disposizione il problema di programmazione lineare viene risolto mediante il package open-source PuLP.
L'output è un booleano, dove 1 indica l'apertura della fabbrica in quel paese, 0 altrimenti.

Lo scenario iniziale è, di conseguenza, caratterizzato nel seguente modo: due sedi in India di cui una a bassa capacità, una sede, invece, per Giappone e Brasile, entrambe ad alta capacità. Si può osservare il mercato di destinazione X a cui sono dedicate le unità prodotte dalla fabbrica Y.

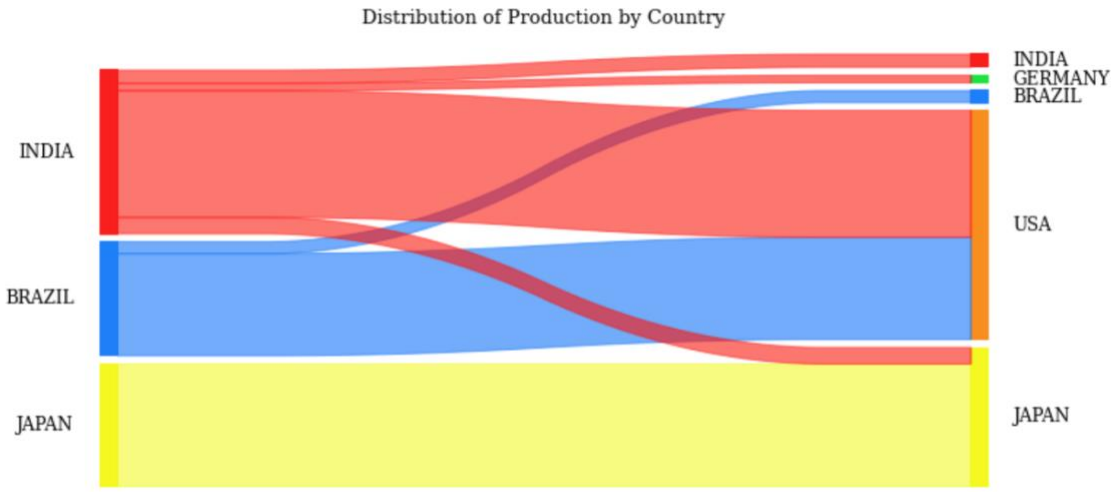
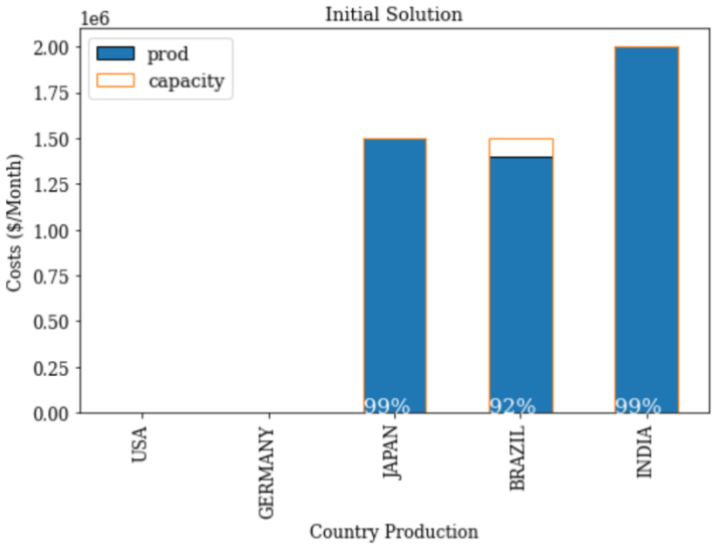


Figura 12: Distribuzione della produzione per paese

La fabbrica giapponese produce solo per il mercato locale, mentre Brasile e India sono principalmente guidate dalla domanda di esportazione.

Dato questo scenario iniziale, si vuole analizzare l'impatto sulla supply chain a fronte di un incremento della domanda del 10% in Giappone e del 20% negli Stati Uniti.



Guardando i tassi di utilizzazione, è facile intuire che la soluzione iniziale non riuscirà ad adattarsi all'incremento della domanda.
Di conseguenza è evidente che non si può fare affidamento su un'unica soluzione e, quindi, attendere che la rete assorbirà la domanda durante tutto l'anno.
Solitamente, questi parametri sono calcolati sulla base della media annuale delle vendite.

La figura 13 mostra la metodologia di simulazione che si va ad applicare.
In sostanza, si vanno a valutare vari cambiamenti nella domanda, modificandone la variabilità: vengono generati 50 scenari, assumendo una distribuzione normale, per simulare la variabilità della domanda, viene effettuato il run del modello di ottimizzazione, trovando la soluzione ottima per ciascuno dei 50 scenari, con l'obiettivo di capire come il modello si adatta alla variabilità della domanda e, infine, si analizzano i risultati, in modo da definire quale soluzione è più robusta per il problema.

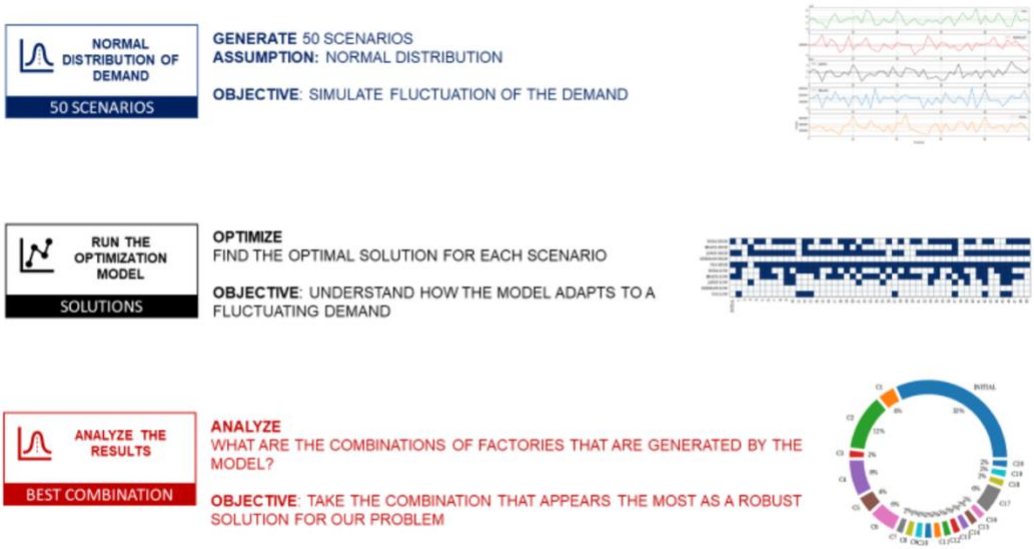


Figura 13: Metodologia di Simulazione

Generazione dei 50 scenari e ricerca della soluzione ottimale per ciascun scenario.
Si assume che la domanda (generata a partire da dati sintetici) segua una distribuzione normale con un coefficiente di variazione CV = 0,5, ma, chiaramente, questo si può adattare a seconda delle proprie necessità. Di seguito il codice Python per la definizione del modello di ottimizzazione.

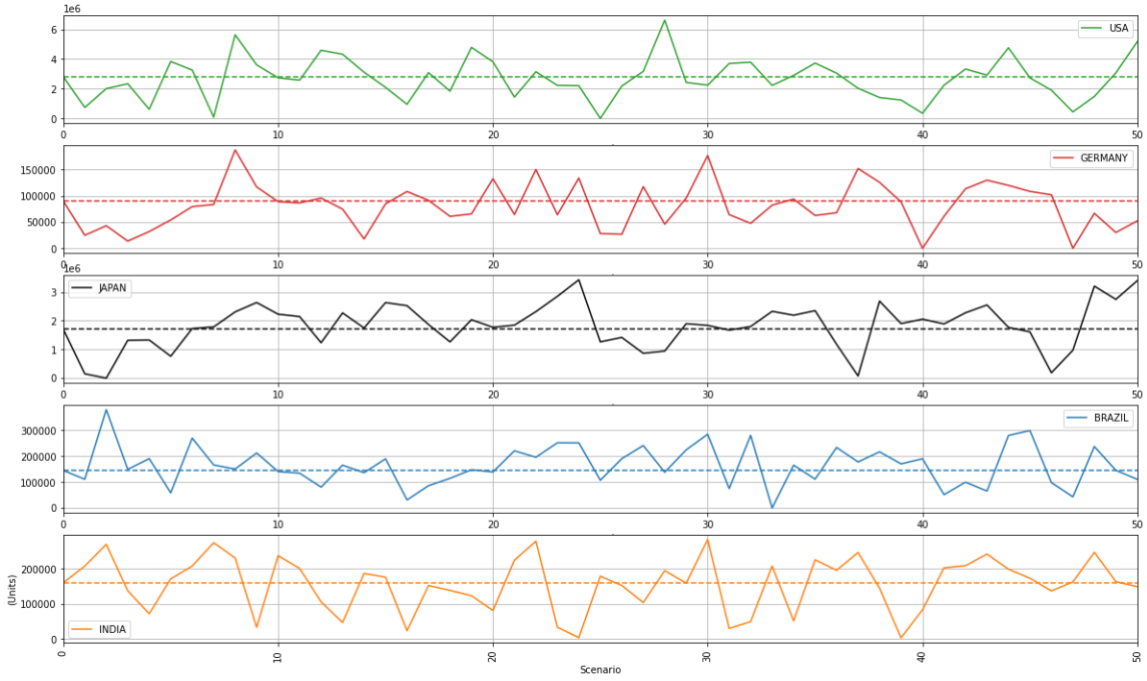
```
def optimization_model(fixed_costs, var_cost, demand, demand_col, cap):
    '''Costruzione del modello di ottimizzazione basandosi sui parametri di input'''

    # Definisco le variabili decisionali
    loc = ['USA', 'GERMANY', 'JAPAN', 'BRAZIL', 'INDIA']
    size = ['LOW', 'HIGH']
    plant_name = [(i,s) for s in size for i in loc]
    prod_name = [(i,j) for i in loc for j in loc]

    # Inizializzazione
    model = pulp.LpProblem("Capacitated Plant Location Model", pulp.LpMinimize)

    # Creo le variabili decisionali
    x = pulp.LpVariable.dicts("production_", prod_name,
                              lowBound=0, upBound=None, cat='continuous')
    y = pulp.LpVariable.dicts("plant_",
                              plant_name, cat='Binary')

    # Definisco la funzione obiettivo
    model += (pulp.lpSum([fixed_costs.loc[i,s] * y[(i,s)] * 1000 for s in size for i in loc])
                + pulp.lpSum([var_cost.loc[i,j] * x[(i,j)]   for i in loc for j in loc]))

    # Aggiungo i Vincoli
    for j in loc:
        model += pulp.lpSum([x[(i, j)] for i in loc]) == demand.loc[j,demand_col]
    for i in loc:
        model += pulp.lpSum([x[(i, j)] for j in loc]) <= pulp.lpSum([cap.loc[i,s]*y[(i,s)] * 1000
                                                                     for s in size])

    # Risolvo il modello
    model.solve()

    # Risultati
    status_out = pulp.LpStatus[model.status]
    objective_out  = pulp.value(model.objective)
    plant_bool = [y[plant_name[i]].varValue for i in range(len(plant_name))]
    fix = sum([fixed_costs.loc[i,s] * y[(i,s)].varValue * 1000 for s in size for i in loc])
    var = sum([var_cost.loc[i,j] * x[(i,j)].varValue for i in loc for j in loc])
    plant_prod = [x[prod_name[i]].varValue for i in range(len(prod_name))]
    return status_out, objective_out, y, x, fix, var
```

Quindi, si va a costruire la distribuzione normale della domanda.

```
# Distribuzione normale
N = 50
df_demand = pd.DataFrame({'scenario': np.array(range(1, N + 1))})
data = demand.reset_index()
# Domanda
CV = 0.5
markets = data['(Units/month)'].values
for col, value in zip(markets, data['Demand'].values):
    sigma = CV * value
    df_demand[col] = np.random.normal(value, sigma, N)
    df_demand[col] = df_demand[col].apply(lambda t: t if t>=0 else 0)

# Aggiungo lo scenario iniziale
COLS = ['scenario'] + list(demand.index)
VALS = [0] + list(demand['Demand'].values)
df_init = pd.DataFrame(dict(zip(COLS, VALS)), index = [0])

# Concat
df_demand = pd.concat([df_init, df_demand])
df_demand.to_excel('data/df_demand-{}PC.xlsx'.format(int(CV * 100)))

df_demand.astype(int).head()
```

```
# Plot
figure, axes = plt.subplots(len(markets), 1)
colors = ['tab:green', 'tab:red', 'black', 'tab:blue', 'tab:orange']
for i in range(len(markets)):
    df_demand.plot(figsize=(20, 12), xlim=[0,N], x='scenario', y=markets[i], ax=axes[i], grid = True, color = colors[i])
    axes[i].axhline(df_demand[markets[i]].values[0], color=colors[i], linestyle="--")
plt.xlabel('Scenario')
plt.ylabel('(Units)')
plt.xticks(rotation=90)
plt.show()
```

Outputs:



	scenario	USA	GERMANY	JAPAN	BRAZIL	INDIA
0	0	2800000	90000	1700000	145000	160000
0	1	4560194	124908	835151	204332	178467
1	2	2404533	80362	3403190	147720	101232
2	3	1285090	34784	644133	5433	130291
3	4	1214410	189157	594207	172870	87432

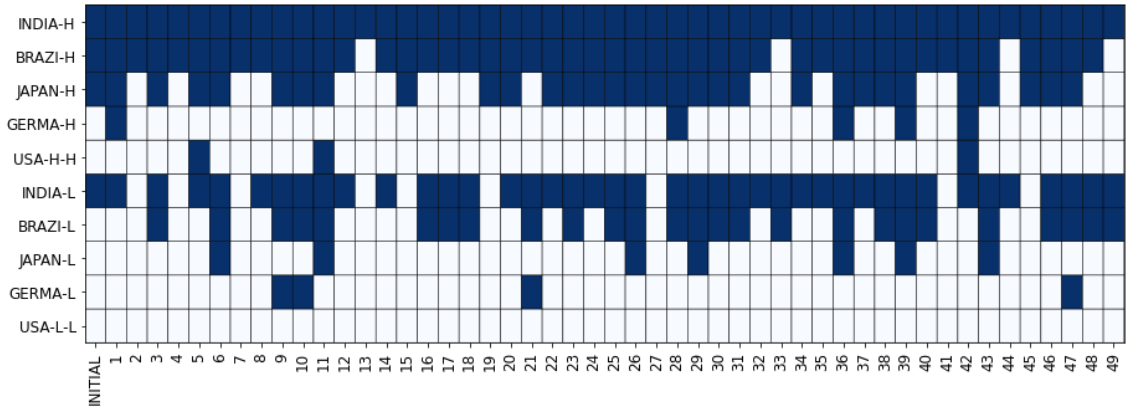
I plot di cui sopra mostrano, rispettivamente, come fluttua la domanda del mercato (in termini di unità per mese) per i cinque paesi nei 50 scenari simulati e, numericamente, la domanda del mercato per i primi cinque scenari simulati.

A questo punto, basandosi sulla fluttuazione della domanda per ciascuno dei diversi paesi nei 50 scenari simulati, si possono simulare gli stabilimenti aperti. Di seguito il codice Python.

```
list_scenario, list_status, list_results, list_totald, list_fixcost, list_varcost = [], [], [], [], [], []

# Scenario iniziale
status_out, objective_out, y, x, fix, var = optimization_model(fixed_costs, var_cost, demand, 'Demand', cap)

# Aggiungo i  risultati
list_scenario.append('INITIAL')
total_demand = demand['Demand'].sum()
list_totald.append(total_demand)
list_status.append(status_out)
list_results.append(objective_out)
list_fixcost.append(fix)
list_varcost.append(var)

# Dataframe per memorizzare gli scenari
df_bool = pd.DataFrame(data = [y[plant_name[i]].varValue for i in range(len(plant_name))], index = [i + '-' + s for s in size for i in loc],
columns = ['INITIAL'])
```

```
demand_var = df_demand.drop(['scenario'], axis = 1).T

for i in range(1, 50): # 0 is the initial scenario
    status_out, objective_out, y, x, fix, var = optimization_model(fixed_costs, var_cost, demand_var, i, cap)

    list_status.append(status_out)
    list_results.append(objective_out)
    df_bool[i] = [y[plant_name[i]].varValue for i in range(len(plant_name))]
    list_fixcost.append(fix)
    list_varcost.append(var)
    total_demand = demand_var[i].sum()
    list_totald.append(total_demand)
    list_scenario.append(i)

df_bool = df_bool.astype(int)
df_bool.to_excel('data/boolean-{}PC.xlsx'.format(int(CV * 100)))
plt.figure(figsize=(15, 5))
plt.pcolor(df_bool, cmap = 'Blues', edgecolors='k', linewidths=0.5)   #
plt.xticks([i + 0.5 for i in range(df_bool.shape[1])], df_bool.columns, rotation = 90, fontsize=12)
plt.yticks([i + 0.5 for i in range(df_bool.shape[0])], [d[0:5]+ '-H' * ('HIGH' in d) + '-L' * ('LOW' in d) for d in df_bool.index], fontsize=12)
plt.xticks(rotation=90)
plt.show()
```

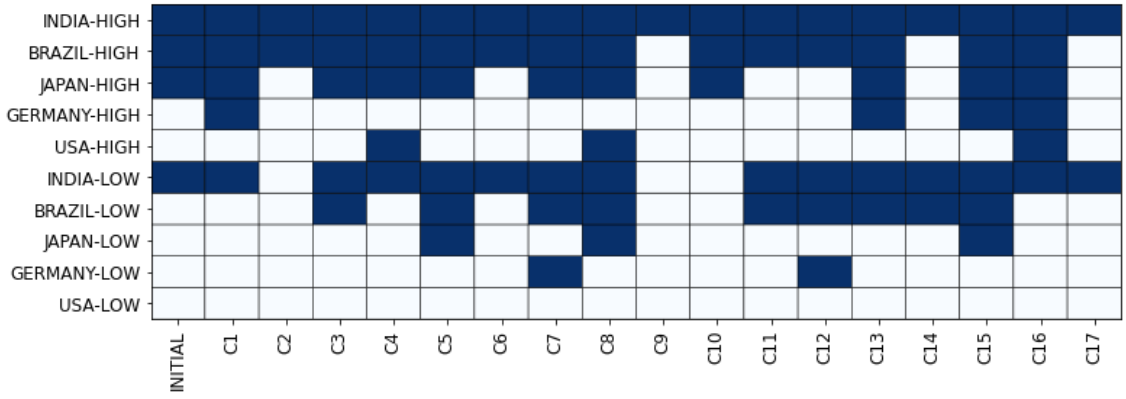
Output:



L'output è una matrice di 50 colonne che indica, per ogni scenario, la miglior soluzione rispetto alla funzione obiettivo del modello d'ottimizzazione. In particolare, sull'asse delle x abbiamo i diversi scenari, mentre sull'asse delle y si hanno le diverse tipologie di stabilimenti. Ogni cella i,j della matrice indica se uno stabilimento è aperto (=1, casella blu scuro) o non lo è (=0, casella celeste).
Dal grafico si possono osservare diversi comportamenti: l'India ha sempre uno stabilimento ad alta capacità aperto, lo scenario 13 necessita solo di una struttura ad alta capacità aperta in India (con la quale, quindi, riesce a soddisfare la domanda del mercato).

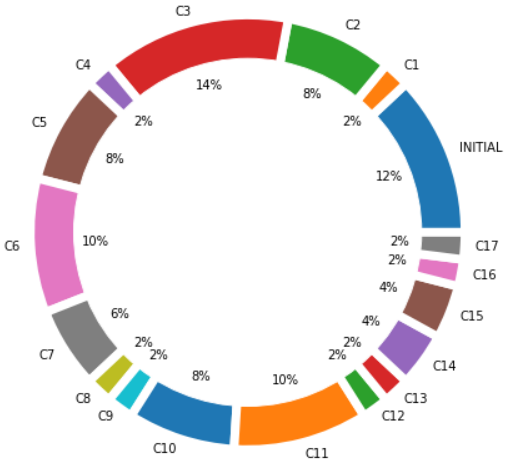
Distribuzione delle combinazioni ottimali
Si vanno ad eliminare le soluzioni duplicate, cioè soluzioni uguali per scenari diversi. Come si può osservare dal seguente grafico le soluzioni uniche sono in numero pari a 18.
Di seguito il codice Python.

```
# Combinazioni uniche
df_unique = df_bool.T.drop_duplicates().T
df_unique.columns = ['INITIAL'] + ['C' + str(i) for i in range(1, len(df_unique.columns))]

# Plot
plt.figure(figsize = (12,4))
plt.pcolor( df_unique, cmap = 'Blues', edgecolors='k', linewidths=0.5)   #
plt.xticks([i + 0.5 for i in range(df_unique.shape[1])], df_unique.columns, rotation = 90,
 fontsize=12)
plt.yticks([i + 0.5 for i in range(df_unique.shape[0])], df_unique.index, fontsize=12)
plt.show()
```



Infine, partendo da queste soluzioni uniche si possono identificare quelle ‘più robuste’, cioè le soluzioni che sono adottate, dal modello di ottimizzazione, per più scenari.

```
COL_NAME, COL_NUMBER = [], []
for col1 in df_unique.columns:
    count = 0
    COL_NAME.append(col1)
    for col2 in df_bool.columns:
        if (df_bool[col2]!=df_unique[col1]).sum()==0:
            count += 1
    COL_NUMBER.append(count)
df_comb = pd.DataFrame({'column':COL_NAME, 'count':COL_NUMBER}).set_index('column')

my_circle = plt.Circle( (0,0), 0.8, color='white')
df_comb.plot.pie(figsize=(8, 8), x='column', y='count', legend= False, pctdistance=0.7,
                                            autopct='%1.0f%%', labeldistance=1.05,
                                            wedgeprops = { 'linewidth' : 7, 'edgecolor' : 'white' })
plt.xlabel('Business Vertical')
p = plt.gcf()
p.gca().add_artist(my_circle)
plt.axis('off')
plt.show()
```



Dal grafico di cui sopra, si evince come la soluzione C3, cioè quella con fabbriche ad alta capacità in Brasile, India e Giappone e fabbriche a bassa capacità in Brasile ed India sia la più robusta. Queste combinazioni tendono a massimizzare la produzione nei paesi con bassi costi di produzione e, nella maggior parte degli scenari, si riesce a soddisfare la domanda.

Non esiste una soluzione perfetta per questo problema poiché è necessario bilanciare la priorità tra la messa in sicurezza della supply chain e la riduzione dei costi. Questo tipo di metodologia può essere un buon inizio per guidare le discussioni sull'obiettivo del livello di servizio del magazzino e dei costi di produzione.