In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import defaultdict
```

In [2]:
```
df_salesOrder = pd.read_excel('Input Data/Sales Orders.xlsx')
df_salesOrder.head()
```

Out[2]:

| | Plant | MATERIAL_NUMBER | SALES_ORDER | SALES_ORDER_ITEM | SALES_ORDER_ITEM_CREATE_DATE_FISCAL_WEEK | SALES_ORDER_I |
|---|---|---|---|---|---|---|
| **0** | WH1 | ITM3249 | 29273582 | 50 | 7 | |
| **1** | WH1 | ITM6 | 29273582 | 30 | 7 | |
| **2** | WH1 | ITM3249 | 29273582 | 20 | 7 | |
| **3** | WH1 | ITM2103 | 29311350 | 440 | 8 | |
| **4** | WH1 | ITM2103 | 29311350 | 310 | 8 | |

In [3]:
```
df_salesOrder['SALES_ORDER'] = df_salesOrder['SALES_ORDER'].astype('str')
```

In [4]:
```
df_salesOrder['MATERIAL_NUMBER'].value_counts()
```

Out[4]:
```
ITM2103    3422
ITM5010    1069
ITM742      734
ITM5011     562
ITM1130     557
           ...
ITM2924       1
ITM5353       1
ITM5116       1
ITM5257       1
ITM770        1
Name: MATERIAL_NUMBER, Length: 3180, dtype: int64
```

In [5]:
```
df = df_salesOrder[['MATERIAL_NUMBER', 'SALES_ORDER']].drop_duplicates()
df['MATERIAL_NUMBER'].value_counts().to_frame()
```

Out[5]:

| | MATERIAL_NUMBER |
|---|---|
| **ITM742** | 674 |

| | MATERIAL_NUMBER |
|---|---|
| **ITM5010** | 576 |
| **ITM1130** | 553 |
| **ITM2103** | 523 |
| **ITM157** | 463 |
| **...** | ... |
| **ITM2924** | 1 |
| **ITM5353** | 1 |
| **ITM5116** | 1 |
| **ITM5257** | 1 |
| **ITM770** | 1 |

3180 rows × 1 columns

In [6]:
```
Order_freq = df_salesOrder['MATERIAL_NUMBER'].value_counts().to_frame()
Order_freq = Order_freq.reset_index()
Order_freq = Order_freq.rename(columns={"MATERIAL_NUMBER": "Order Count", "index": "MATERIAL_NUMBER"})
Order_freq.head()
```

Out[6]:

| | MATERIAL_NUMBER | Order Count |
|---|---|---|
| **0** | ITM2103 | 3422 |
| **1** | ITM5010 | 1069 |
| **2** | ITM742 | 734 |
| **3** | ITM5011 | 562 |
| **4** | ITM1130 | 557 |

In [7]:
```
distinctOrder_freq = df['MATERIAL_NUMBER'].value_counts().to_frame()
distinctOrder_freq = distinctOrder_freq.reset_index()
distinctOrder_freq = distinctOrder_freq.rename(columns={"MATERIAL_NUMBER": "Order Count", "index": "MATERIAL_NU
distinctOrder_freq.head()
```

Out[7]:

| | MATERIAL_NUMBER | Order Count |
|---|---|---|

| | MATERIAL_NUMBER | Order Count |
|---|---|---|
| **0** | ITM742 | 674 |
| **1** | ITM5010 | 576 |
| **2** | ITM1130 | 553 |
| **3** | ITM2103 | 523 |
| **4** | ITM157 | 463 |

In [8]:
```
df_salesOrder.shape
```

Out[8]:
```
(68895, 7)
```

In [9]:
```
Total_salesOrderLines = df_salesOrder.shape[0]
Total_salesOrderLines
```

Out[9]:
```
68895
```

In [10]:
```
Total_distinct_salesOrders = df_salesOrder.SALES_ORDER.nunique()
Total_distinct_salesOrders
```

Out[10]:
```
18998
```

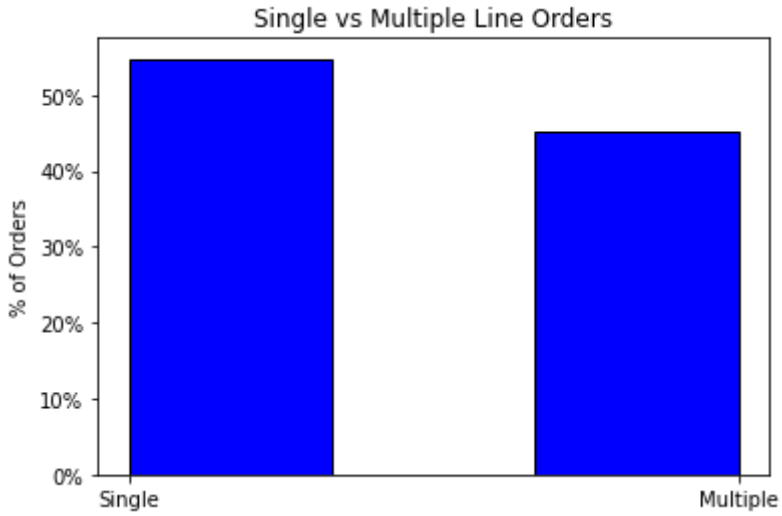
In [11]:
```
Total_distinct_Products_sold = df_salesOrder.MATERIAL_NUMBER.nunique()
Total_distinct_Products_sold
```

Out[11]:
```
3180
```

In [12]:
```
df = df_salesOrder[['MATERIAL_NUMBER', 'SALES_ORDER']].drop_duplicates()
df2 = df.groupby('SALES_ORDER')['MATERIAL_NUMBER'].agg(No_of_Lines='count').reset_index()
df2.loc[df2['No_of_Lines'] == 1, 'Single or Multiple'] = 'Single'
df2.loc[df2['No_of_Lines'] > 1, 'Single or Multiple'] = 'Multiple'

data = df2['Single or Multiple']

from matplotlib.ticker import PercentFormatter
fig = plt.figure()
ax = fig.add_subplot(111)
ax.hist(data, edgecolor='black', bins=3, color = 'blue',
        weights=np.ones_like(data)*100 / len(data))
```

```
ax.yaxis.set_major_formatter(PercentFormatter())
plt.ylabel('% of Orders')
plt.title('Single vs Multiple Line Orders')
```

Out[12]: Text(0.5, 1.0, 'Single vs Multiple Line Orders')

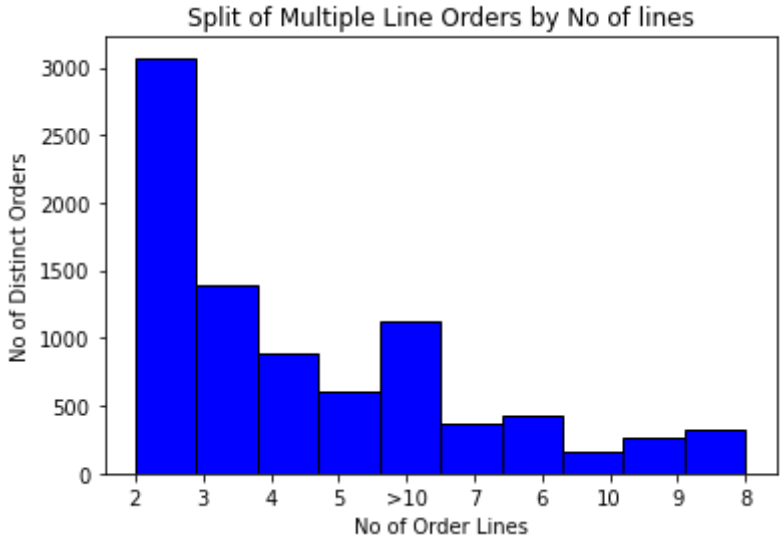



In [13]:
```
df = df_salesOrder[['MATERIAL_NUMBER', 'SALES_ORDER']].drop_duplicates()
df2 = df.groupby('SALES_ORDER')['MATERIAL_NUMBER'].agg(No_of_Lines='count').reset_index()
df2.loc[df2['No_of_Lines'] == 1, 'Single or Multiple'] = 'Single'
df2.loc[df2['No_of_Lines'] > 1, 'Single or Multiple'] = 'Multiple'

df2.loc[df2['No_of_Lines'] > 10, 'No_of_Lines'] = '>10'
df3 = df2[df2['No_of_Lines'] != 1]
df3['No_of_Lines'] = df3['No_of_Lines'].astype(str)

data = df3['No_of_Lines']
fig = plt.figure()
ax = fig.add_subplot(111)
ax.hist(data, edgecolor='black', color = 'blue')
plt.ylabel('No of Distinct Orders')
plt.xlabel('No of Order Lines')
plt.title('Split of Multiple Line Orders by No of lines')
```

```
/var/folders/19/_tyxbxcj0n57tt5s1rk8m0l40000gn/T/ipykernel_67885/2502679177.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
```

```
urning-a-view-versus-a-copy
  df3['No_of_Lines'] = df3['No_of_Lines'].astype(str)
```

Out[13]: `Text(0.5, 1.0, 'Split of Multiple Line Orders by No of lines')`




In [14]:
```
mybasket = (df_salesOrder.groupby(['SALES_ORDER','MATERIAL_NUMBER'])['SCHEDULE_QUANTITY'].
            sum().unstack().reset_index().fillna(0).set_index('SALES_ORDER'))
```

In [15]:
```
mybasket.head()
```

Out[15]:

| MATERIAL_NUMBER | ITM1 | ITM10 | ITM100 | ITM1000 | ITM1002 | ITM1004 | ITM1006 | ITM1007 | ITM1012 | ITM1017 | ... | ITM97 | ITM98 | I |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SALES_ORDER | | | | | | | | | | | | | | |
| 27034852 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | |
| 27546143 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | |
| 27546176 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | |
| 27728583 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | |
| 27997929 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | |

5 rows × 3180 columns

In [16]:
```
#Fucntion
def my_encode_units(x):
```

```
    if x <= 0:
        return False
    if x > 0:
        return True

mybasket = mybasket.applymap(my_encode_units)
```

In [17]:
```
mybasket.head()
```

Out[17]:

| MATERIAL_NUMBER<br>SALES_ORDER | ITM1 | ITM10 | ITM100 | ITM1000 | ITM1002 | ITM1004 | ITM1006 | ITM1007 | ITM1012 | ITM1017 | ... | ITM97 | ITM98 | I |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **27034852** | False | False | False | False | False | False | False | False | False | False | ... | False | False | |
| **27546143** | False | False | False | False | False | False | False | False | False | False | ... | False | False | |
| **27546176** | False | False | False | False | False | False | False | False | False | False | ... | False | False | |
| **27728583** | False | False | False | False | False | False | False | False | False | False | ... | False | False | |
| **27997929** | False | False | False | False | False | False | False | False | False | False | ... | False | False | |

5 rows × 3180 columns

In [18]:
```
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
my_freq_items = apriori(mybasket, min_support=.001, use_colnames=True)
```

In [19]:
```
my_freq_items.sort_values(by=['support'], ascending=False)
```

Out[19]:

| | support | itemsets |
|---|---|---|
| **691** | 0.035477 | (ITM742) |
| **242** | 0.030319 | (ITM5010) |
| **25** | 0.029108 | (ITM1130) |
| **159** | 0.027529 | (ITM2103) |
| **79** | 0.024371 | (ITM157) |
| **...** | ... | ... |
| **1934** | 0.001000 | (ITM789, ITM6326, ITM1130) |

| | support | itemsets |
|---|---|---|
| **1010** | 0.001000 | (ITM5808, ITM157) |
| **271** | 0.001000 | (ITM5094) |
| **2636** | 0.001000 | (ITM5, ITM328, ITM677, ITM3164) |
| **2269** | 0.001000 | (ITM5011, ITM677, ITM5012) |

2887 rows × 2 columns

In [20]:
```
my_rules = association_rules(my_freq_items, metric="lift",min_threshold=1)
```

In [21]:
```
my_rules.head()
```

Out[21]:

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|---|---|
| **0** | (ITM1114) | (ITM1) | 0.009369 | 0.012107 | 0.001053 | 0.112360 | 9.280899 | 0.000939 | 1.112943 |
| **1** | (ITM1) | (ITM1114) | 0.012107 | 0.009369 | 0.001053 | 0.086957 | 9.280899 | 0.000939 | 1.084976 |
| **2** | (ITM1) | (ITM2105) | 0.012107 | 0.008580 | 0.001000 | 0.082609 | 9.628221 | 0.000896 | 1.080695 |
| **3** | (ITM2105) | (ITM1) | 0.008580 | 0.012107 | 0.001000 | 0.116564 | 9.628221 | 0.000896 | 1.118241 |
| **4** | (ITM1) | (ITM2769) | 0.012107 | 0.007053 | 0.002474 | 0.204348 | 28.971642 | 0.002389 | 1.247966 |

In [22]:
```
my_rules = my_rules.sort_values(by=['antecedent support', 'consequent support','confidence'], ascending=False)
```

In [23]:
```
my_rules.head()
```

Out[23]:

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|---|---|
| **190** | (ITM742) | (ITM1130) | 0.035477 | 0.029108 | 0.008632 | 0.243323 | 8.359238 | 0.007600 | 1.283100 |
| **1329** | (ITM742) | (ITM5011) | 0.035477 | 0.023108 | 0.010212 | 0.287834 | 12.456189 | 0.009392 | 1.371720 |
| **2217** | (ITM742) | (ITM677) | 0.035477 | 0.021844 | 0.008580 | 0.241840 | 11.071016 | 0.007805 | 1.290170 |
| **1364** | (ITM742) | (ITM5012) | 0.035477 | 0.021634 | 0.008632 | 0.243323 | 11.247345 | 0.007865 | 1.292978 |
| **1136** | (ITM742) | (ITM373) | 0.035477 | 0.018949 | 0.007527 | 0.212166 | 11.196480 | 0.006855 | 1.245251 |

In [24]:
```
my_rules.to_csv('complete_rules.csv', index=False)
```

In [25]:
```
df = pd.read_csv('complete_rules.csv')
df.head()
```

Out[25]:

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|---|---|
| 0 | frozenset({'ITM742'}) | frozenset({'ITM1130'}) | 0.035477 | 0.029108 | 0.008632 | 0.243323 | 8.359238 | 0.007600 | 1.283100 |
| 1 | frozenset({'ITM742'}) | frozenset({'ITM5011'}) | 0.035477 | 0.023108 | 0.010212 | 0.287834 | 12.456189 | 0.009392 | 1.371720 |
| 2 | frozenset({'ITM742'}) | frozenset({'ITM677'}) | 0.035477 | 0.021844 | 0.008580 | 0.241840 | 11.071016 | 0.007805 | 1.290170 |
| 3 | frozenset({'ITM742'}) | frozenset({'ITM5012'}) | 0.035477 | 0.021634 | 0.008632 | 0.243323 | 11.247345 | 0.007865 | 1.292978 |
| 4 | frozenset({'ITM742'}) | frozenset({'ITM373'}) | 0.035477 | 0.018949 | 0.007527 | 0.212166 | 11.196480 | 0.006855 | 1.245251 |

In [26]:
```
df = df[df["antecedents"].str.contains(",") == False]
df = df[df["consequents"].str.contains(",") == False]

df['antecedents'] = df['antecedents'].str.replace('frozenset', '')
df['antecedents'] = df['antecedents'].str.replace('{', '')
df['antecedents'] = df['antecedents'].str.replace('}', '')
df['antecedents'] = df['antecedents'].str.replace('(', '')
df['antecedents'] = df['antecedents'].str.replace(')', '')
df['antecedents'] = df['antecedents'].str.replace("'", '')

df['consequents'] = df['consequents'].str.replace('frozenset', '')
df['consequents'] = df['consequents'].str.replace('{', '')
df['consequents'] = df['consequents'].str.replace('}', '')
df['consequents'] = df['consequents'].str.replace('(', '')
df['consequents'] = df['consequents'].str.replace(')', '')
df['consequents'] = df['consequents'].str.replace("'", '')

rules = df
rules
```

Out[26]:

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|---|---|
| 0 | ITM742 | ITM1130 | 0.035477 | 0.029108 | 0.008632 | 0.243323 | 8.359238 | 0.007600 | 1.283100 |
| 1 | ITM742 | ITM5011 | 0.035477 | 0.023108 | 0.010212 | 0.287834 | 12.456189 | 0.009392 | 1.371720 |
| 2 | ITM742 | ITM677 | 0.035477 | 0.021844 | 0.008580 | 0.241840 | 11.071016 | 0.007805 | 1.290170 |
| 3 | ITM742 | ITM5012 | 0.035477 | 0.021634 | 0.008632 | 0.243323 | 11.247345 | 0.007865 | 1.292978 |

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|---|---|
| **4** | ITM742 | ITM373 | 0.035477 | 0.018949 | 0.007527 | 0.212166 | 11.196480 | 0.006855 | 1.245251 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **14570** | ITM6610 | ITM6500 | 0.001158 | 0.004316 | 0.001000 | 0.863636 | 200.089800 | 0.000995 | 7.301681 |
| **14571** | ITM6610 | ITM1838 | 0.001158 | 0.001842 | 0.001000 | 0.863636 | 468.781818 | 0.000998 | 7.319823 |
| **14596** | ITM6418 | ITM5011 | 0.001105 | 0.023108 | 0.001105 | 1.000000 | 43.275626 | 0.001080 | inf |
| **14646** | ITM5650 | ITM5652 | 0.001105 | 0.003000 | 0.001053 | 0.952381 | 317.426901 | 0.001049 | 20.936993 |
| **14709** | ITM6427 | ITM6421 | 0.001053 | 0.001579 | 0.001000 | 0.950000 | 601.603333 | 0.000998 | 19.968418 |

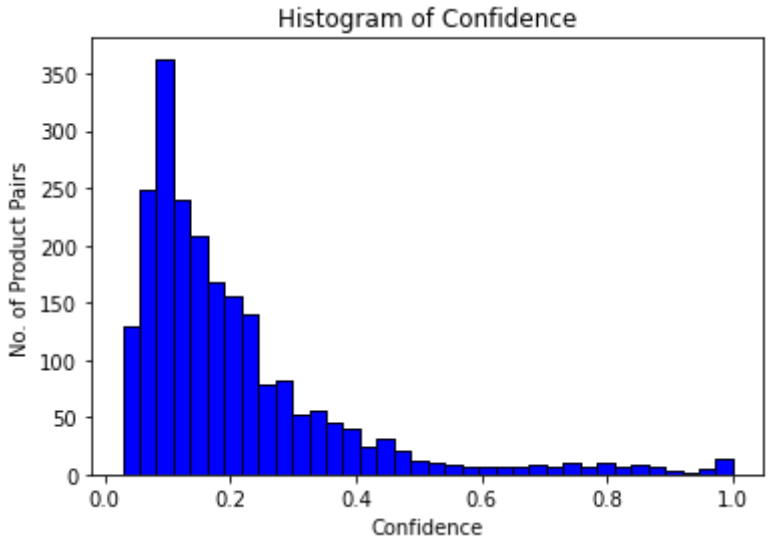
2228 rows × 9 columns

In [27]:
```
rules.to_csv('product affinity.csv', index=False)
```

In [28]:
```
plt.hist(rules['confidence'], color = 'blue', edgecolor = 'black',
         bins = int(180/5))

# seaborn histogram
sns.distplot(rules['confidence'], hist=True, kde=False,
             bins=int(180/5), color = 'blue',
             hist_kws={'edgecolor':'black'})
# Add labels
plt.title('Histogram of Confidence')
plt.xlabel('Confidence')
plt.ylabel('No. of Product Pairs')
```

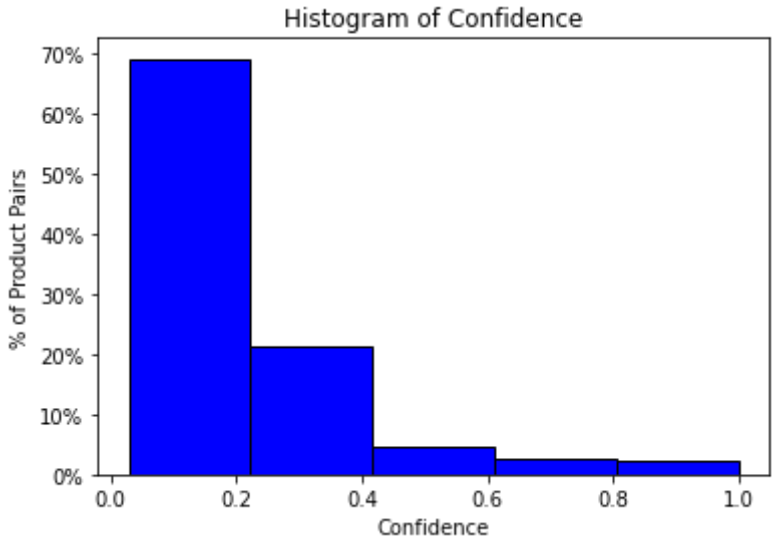
```
/Users/shyamsrikumar/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `d
istplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histogra
ms).
  warnings.warn(msg, FutureWarning)
```

Out[28]:
```
Text(0, 0.5, 'No. of Product Pairs')
```



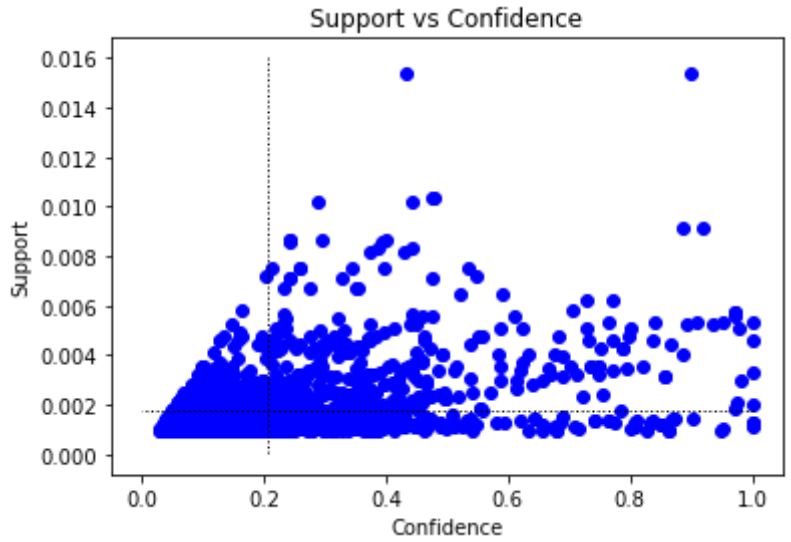

In [29]:
```
from matplotlib.ticker import PercentFormatter

#define data values
data = rules['confidence']

#create relative frequency histogram with percentages on y-axis
fig = plt.figure()
ax = fig.add_subplot(111)
ax.hist(data, edgecolor='black', bins=int(100/20), color = 'blue',
        weights=np.ones_like(data)*100 / len(data))
ax.yaxis.set_major_formatter(PercentFormatter())
plt.title('Histogram of Confidence')
plt.xlabel('Confidence')
plt.ylabel('% of Product Pairs')
```

Out[29]:
```
Text(0, 0.5, '% of Product Pairs')
```



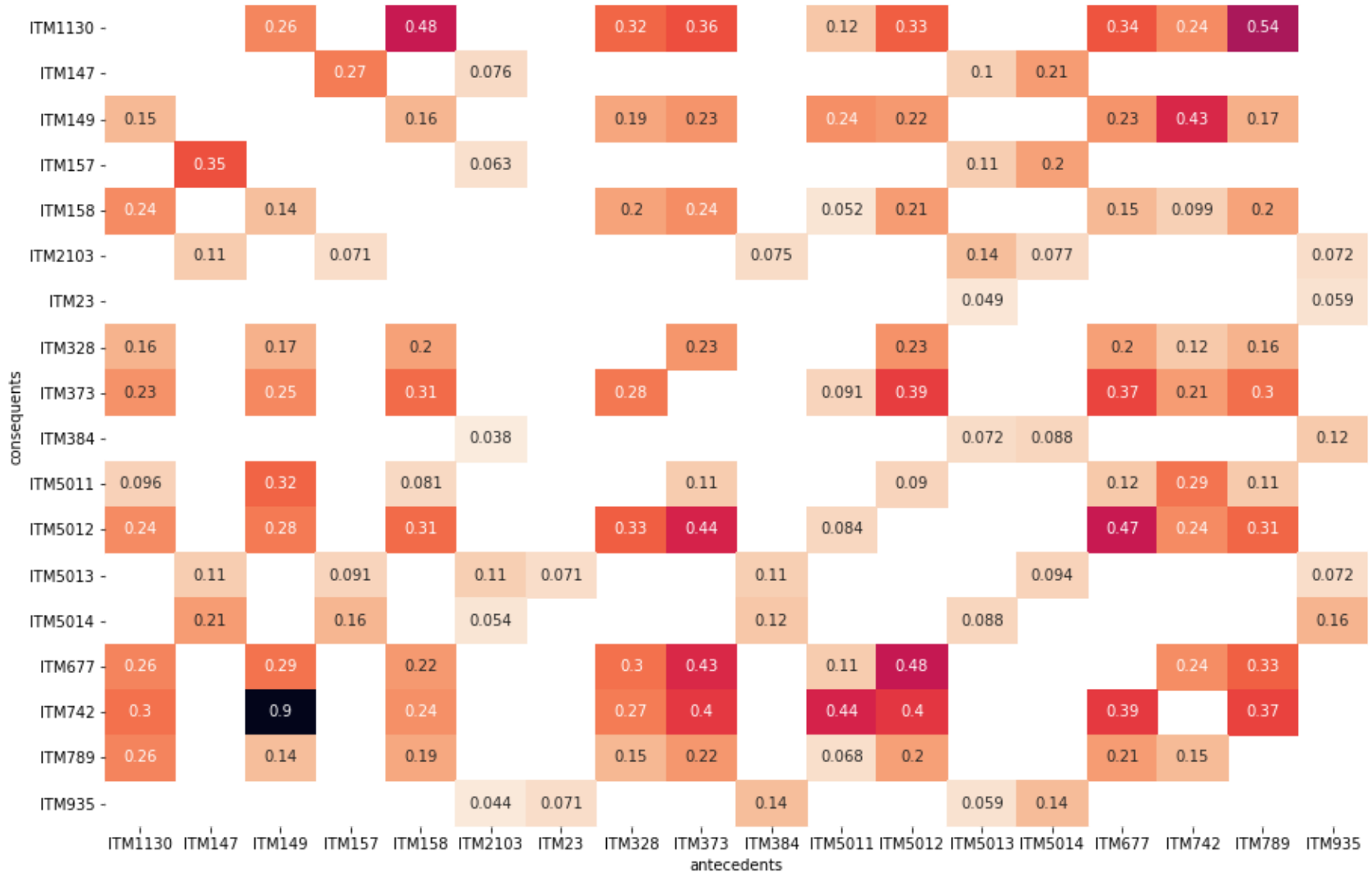

In [30]:

```
x = rules['confidence']
y = rules['support']

plt.scatter(x, y, c ="blue")
plt.plot([x.mean(),x.mean()],[.016,.000],'k-', linestyle = ":", lw=1)
plt.plot([1,0],[y.mean(),y.mean()],'k-', linestyle = ":", lw=1)

# To show the plot
plt.title('Support vs Confidence')
plt.xlabel('Confidence')
plt.ylabel('Support')
plt.show()
```




In [31]:
```
top20_materials = distinctOrder_freq.head(20)['MATERIAL_NUMBER'].tolist()
df = rules[rules['antecedents'].isin(top20_materials)]
df = df[df['consequents'].isin(top20_materials)]
```

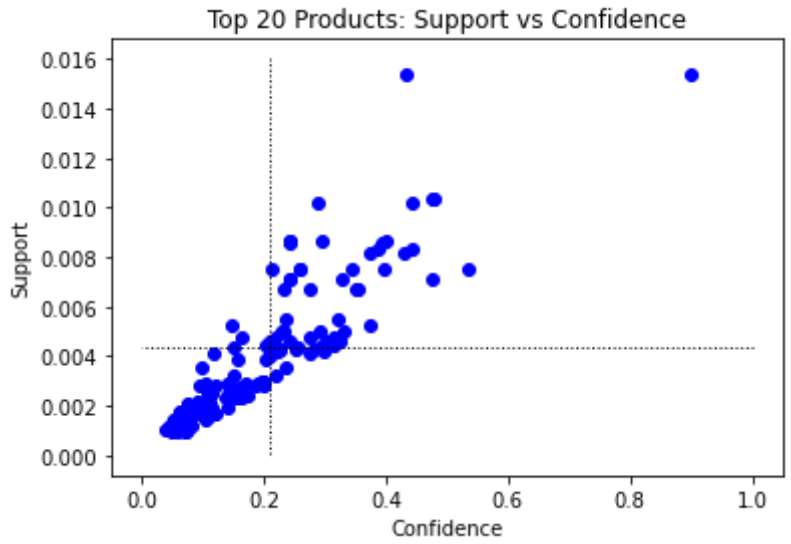
In [32]:
```
# Transform antecedent, consequent, and support columns into matrix
support_table = df.pivot(index='consequents', columns='antecedents', values='confidence')

plt.figure(figsize=(15,10))
cmap = sns.cm.rocket_r
sns.heatmap(support_table, cmap = cmap, annot=True, cbar=False)
plt.show()
```



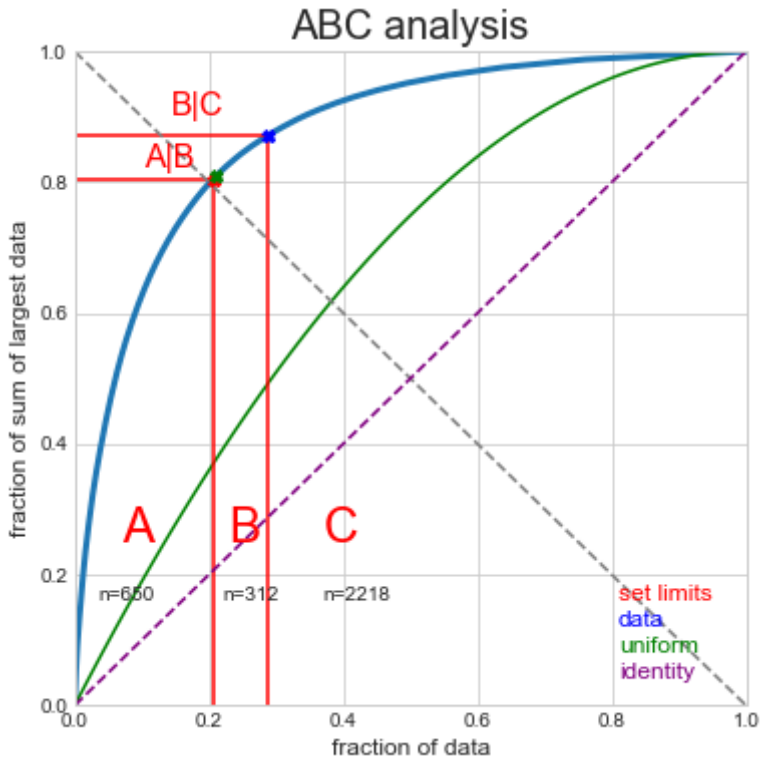

In [33]:

```
x = df['confidence']
y = df['support']

plt.scatter(x, y, c ="blue")
plt.plot([x.mean(),x.mean()],[.016,.000],'k-', linestyle = ":", lw=1)
plt.plot([1,0],[y.mean(),y.mean()],'k-', linestyle = ":", lw=1)

# To show the plot
plt.title('Top 20 Products: Support vs Confidence')
```

```
plt.xlabel('Confidence')
plt.ylabel('Support')
plt.show()
```




In [34]:
```
from abc_analysis import abc_analysis, abc_plot
abc = abc_analysis(distinctOrder_freq['Order Count'], True)
```




In [35]:
```
# index position of A, B, and C Videos
a_index = abc['Aind']
b_index = abc['Bind']
c_index = abc['Cind']

# New Column indicating A, B, or C
cond_list = [distinctOrder_freq.index.isin(a_index),
             distinctOrder_freq.index.isin(b_index),
             distinctOrder_freq.index.isin(c_index)]

choice_list = ['A','B','C']

distinctOrder_freq['ABC'] = np.select(cond_list, choice_list)
distinctOrder_freq
```

Out[35]:

| | MATERIAL_NUMBER | Order Count | ABC |
|---|---|---|---|
| **0** | ITM742 | 674 | A |
| **1** | ITM5010 | 576 | A |

| | MATERIAL_NUMBER | Order Count | ABC |
|---|---|---|---|
| **2** | ITM1130 | 553 | A |
| **3** | ITM2103 | 523 | A |
| **4** | ITM157 | 463 | A |
| **...** | ... | ... | ... |
| **3175** | ITM2924 | 1 | C |
| **3176** | ITM5353 | 1 | C |
| **3177** | ITM5116 | 1 | C |
| **3178** | ITM5257 | 1 | C |
| **3179** | ITM770 | 1 | C |

3180 rows × 3 columns

In [36]:
```
cnt_A = int(distinctOrder_freq[distinctOrder_freq['ABC']=='A'].MATERIAL_NUMBER.count())
Class_A_materials = distinctOrder_freq.head(cnt_A)['MATERIAL_NUMBER'].tolist()
df = rules[rules['antecedents'].isin(Class_A_materials)]
df = df[df['consequents'].isin(Class_A_materials)]

x = df['confidence']
y = df['support']

plt.scatter(x, y, c ="blue")
plt.plot([x.mean(),x.mean()],[.016,.000],'k-', linestyle = ":", lw=1)
plt.plot([1,0],[y.mean(),y.mean()],'k-', linestyle = ":", lw=1)

# To show the plot
plt.title('Class A Products: Support vs Confidence')
plt.xlabel('Confidence')
plt.ylabel('Support')
plt.show()
```


In [37]:
```
Product_dims = pd.read_excel('Input Data/Product dims.xlsx')
```

```
/Users/shyamsrikumar/opt/anaconda3/lib/python3.8/site-packages/xlrd/xlsx.py:266: DeprecationWarning: This metho
d will be removed in future versions.  Use 'tree.iter()' or 'list(tree.iter())' instead.
  for elem in self.tree.iter() if Element_has_iter else self.tree.getiterator():
/Users/shyamsrikumar/opt/anaconda3/lib/python3.8/site-packages/xlrd/xlsx.py:312: DeprecationWarning: This metho
d will be removed in future versions.  Use 'tree.iter()' or 'list(tree.iter())' instead.
  for elem in self.tree.iter() if Element_has_iter else self.tree.getiterator():
/Users/shyamsrikumar/opt/anaconda3/lib/python3.8/site-packages/xlrd/xlsx.py:266: DeprecationWarning: This metho
d will be removed in future versions.  Use 'tree.iter()' or 'list(tree.iter())' instead.
  for elem in self.tree.iter() if Element_has_iter else self.tree.getiterator():
```

In [185…
```
import math

STD_LOC_VOL = (46 * 48 * 60) * .8
nbrOfdays = 18 * 7
DaysOfSupply = 3
df = distinctOrder_freq
product = df['MATERIAL_NUMBER'].values.tolist()

distinctOrder_freq['PICK_FACE_NO_OF_LOC'] = 1

for ind, rec in Product_dims.iterrows():
    p = rec['Material']
    vol = 0.0
    if rec['Volume Cubic Unit'] == 'IN':
        vol = rec['Volume']
    elif rec['Volume Cubic Unit'] == 'CM':
```

```
        vol = rec['Volume'] * (0.393701)**3.0
    elif rec['Volume Cubic Unit'] == 'MM':
        vol = rec['Volume'] * (0.0393701)**3
    elif rec['Volume Cubic Unit'] == 'M':
        vol = rec['Volume'] * (39.3701)**3
    if vol > 0.0:
        if p in product:
            avg_demand = df[df['MATERIAL_NUMBER'] == p]['Order Count'] /nbrOfdays * DaysOfSupply
            distinctOrder_freq.loc[distinctOrder_freq['MATERIAL_NUMBER'] == p,
                                   'PICK_FACE_NO_OF_LOC'] = int(math.ceil(avg_demand*vol/STD_LOC_VOL))

distinctOrder_freq[distinctOrder_freq['PICK_FACE_NO_OF_LOC'] > 1]
```

Out[185…

| | MATERIAL_NUMBER | Order Count | ABC | PICK_FACE_NO_OF_LOC |
|---|---|---|---|---|
| **43** | ITM6182 | 203 | A | 3 |

In [324…

```
distinctOrder_freq.to_csv('Products.csv',index=False)
```

In [325…

```
Products = pd.read_csv('Products.csv')
Locations = pd.read_excel('Input Data/Location Distances.xlsx')
```

```
/Users/shyamsrikumar/opt/anaconda3/lib/python3.8/site-packages/xlrd/xlsx.py:266: DeprecationWarning: This metho
d will be removed in future versions.  Use 'tree.iter()' or 'list(tree.iter())' instead.
  for elem in self.tree.iter() if Element_has_iter else self.tree.getiterator():
/Users/shyamsrikumar/opt/anaconda3/lib/python3.8/site-packages/xlrd/xlsx.py:312: DeprecationWarning: This metho
d will be removed in future versions.  Use 'tree.iter()' or 'list(tree.iter())' instead.
  for elem in self.tree.iter() if Element_has_iter else self.tree.getiterator():
/Users/shyamsrikumar/opt/anaconda3/lib/python3.8/site-packages/xlrd/xlsx.py:266: DeprecationWarning: This metho
d will be removed in future versions.  Use 'tree.iter()' or 'list(tree.iter())' instead.
  for elem in self.tree.iter() if Element_has_iter else self.tree.getiterator():
```

In [326…

```
Locations.head()
```

Out[326…

| | Location | Aisle | Bin Position | Level | Oneway Pick Distance | Factored Vertical Distance | Final Pick Distance | Distance to dropoff | Final distance to drop off | Sort Field |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 50.54.A | 50 | 54 | A | 0.000000 | 0.0 | 0.000000 | 18.000000 | 18.000000 | 1 |
| **1** | 50.54.B | 50 | 54 | B | 0.000000 | 18.5 | 18.500000 | 18.000000 | 36.500000 | 2 |
| **2** | 50.54.C | 50 | 54 | C | 0.000000 | 37.0 | 37.000000 | 18.000000 | 55.000000 | 3 |
| **3** | 50.54.D | 50 | 54 | D | 0.000000 | 55.5 | 55.500000 | 18.000000 | 73.500000 | 4 |
| **4** | 50.53.A | 50 | 53 | A | 4.041667 | 0.0 | 4.041667 | 22.041667 | 22.041667 | 9 |

In [327…

```
No_of_Products = 100
product = Products.head(No_of_Products)['MATERIAL_NUMBER'].values.tolist()
No_of_Locations = 216
No_of_Loc_per_product = 1
locations = Locations[Locations['Level']=='A'].head(No_of_Locations)
locations = locations.sort_values(by=['Final distance to drop off', 'Final Pick Distance','Sort Field'],
                                  ascending=True)
location = locations['Location'].values.tolist()
```

In [328…

```
same_aisle = [{50,49},{48,47},{46,45},{44,43},{42,41},{40,39},{38,37},{36,35},
              {34,33},{32,31},{30,29},{28,27},{26,25},{24,23},{22,21},{20,19},
              {18,17},{16,15},{14,13},{12,11},{10,9},{8,7},{6,5},{4,3},{2,1}]
distance_Pick = pd.Series(locations['Final Pick Distance'].values,index=locations.Location).to_dict()
distance_IO = pd.Series(locations['Final distance to drop off'].values,index=locations.Location).to_dict()
distance_BTW = defaultdict(dict)
for l1 in location:
    distance_BTW[l1] = defaultdict(dict)
    for l2 in location:
        a1 = int(l1[:2])
        a2 = int(l2[:2])
        if {a1,a2} in same_aisle:
            if a1 > a2:
                d1 = distance_Pick[l1]
                d2 = distance_Pick[l2] - 12
            else:
                d1 = distance_Pick[l1] - 12
                d2 = distance_Pick[l2]
            distance_BTW[l1][l2] = abs(d1-d2) + 12
        else:
            distance_BTW[l1][l2] = abs(distance_Pick[l1] - distance_Pick[l2])

list(distance_BTW.items())[:1]
```

Out[328…

```
[('50.54.A',
  defaultdict(dict,
              {'50.54.A': 0.0,
               '50.53.A': 4.041666666666667,
               '50.52.A': 8.083333333333334,
               '49.54.A': 12.0,
               '50.51.A': 12.125,
               '49.53.A': 16.041666666666668,
               '50.50.A': 16.166666666666668,
               '49.52.A': 20.083333333333336,
               '50.49.A': 20.208333333333336,
```

```
'48.54.A': 453.45833333333394,
'49.51.A': 24.125,
'50.48.A': 24.250000000000004,
'48.53.A': 449.41666666666725,
'49.50.A': 28.166666666666668,
'50.47.A': 28.29166666666667,
'48.52.A': 445.37500000000057,
'49.49.A': 32.208333333333336,
'50.46.A': 32.333333333333336,
'47.54.A': 465.45833333333394,
'48.51.A': 441.3333333333339,
'49.48.A': 36.25,
'50.45.A': 36.375,
'47.53.A': 461.41666666666725,
'48.50.A': 437.2916666666672,
'49.47.A': 40.29166666666667,
'50.44.A': 40.416666666666664,
'47.52.A': 457.37500000000057,
'48.49.A': 433.2500000000005,
'49.46.A': 44.333333333333336,
'50.43.A': 44.45833333333333,
'47.51.A': 453.3333333333339,
'48.48.A': 429.2083333333338,
'49.45.A': 48.375,
'50.42.A': 48.49999999999999,
'47.50.A': 449.2916666666672,
'48.47.A': 425.16666666666714,
'49.44.A': 52.416666666666664,
'50.41.A': 52.54166666666666,
'47.49.A': 445.2500000000005,
'48.46.A': 421.12500000000045,
'49.43.A': 56.45833333333333,
'50.40.A': 56.58333333333332,
'47.48.A': 441.2083333333338,
'48.45.A': 417.08333333333377,
'49.42.A': 60.49999999999999,
'50.39.A': 60.624999999999986,
'47.47.A': 437.16666666666714,
'48.44.A': 413.0416666666671,
'49.41.A': 64.54166666666666,
'50.38.A': 64.66666666666666,
'47.46.A': 433.12500000000045,
'48.43.A': 409.0000000000004,
'49.40.A': 68.58333333333331,
'50.37.A': 68.70833333333333,
'47.45.A': 429.08333333333377,
'48.42.A': 404.9583333333337,
'49.39.A': 72.62499999999999,
'50.36.A': 72.75,
```

```
'47.44.A': 425.0416666666671,
'48.41.A': 400.916666666667,
'49.38.A': 76.66666666666666,
'50.35.A': 76.79166666666667,
'47.43.A': 421.0000000000004,
'48.40.A': 396.87500000000034,
'49.37.A': 80.70833333333333,
'50.34.A': 80.83333333333334,
'47.42.A': 416.9583333333337,
'48.39.A': 392.83333333333366,
'49.36.A': 84.75,
'50.33.A': 84.87500000000001,
'47.41.A': 412.916666666667,
'48.38.A': 388.79166666666697,
'49.35.A': 88.79166666666667,
'50.32.A': 88.91666666666669,
'47.40.A': 408.87500000000034,
'48.37.A': 384.7500000000003,
'49.34.A': 92.83333333333334,
'50.31.A': 92.95833333333336,
'47.39.A': 404.83333333333366,
'48.36.A': 380.7083333333336,
'49.33.A': 96.87500000000001,
'50.30.A': 97.00000000000003,
'47.38.A': 400.79166666666697,
'48.35.A': 376.6666666666669,
'49.32.A': 100.91666666666669,
'50.29.A': 101.0416666666667,
'47.37.A': 396.7500000000003,
'48.34.A': 372.6250000000002,
'49.31.A': 104.95833333333336,
'50.28.A': 105.08333333333337,
'47.36.A': 392.7083333333336,
'48.33.A': 368.58333333333354,
'49.30.A': 109.00000000000003,
'50.27.A': 109.12500000000004,
'47.35.A': 388.6666666666669,
'48.32.A': 364.54166666666686,
'49.29.A': 113.0416666666667,
'50.26.A': 113.16666666666671,
'47.34.A': 384.6250000000002,
'48.31.A': 360.50000000000017,
'49.28.A': 117.08333333333337,
'50.25.A': 117.20833333333339,
'47.33.A': 380.58333333333354,
'48.30.A': 356.4583333333335,
'49.27.A': 121.12500000000004,
'47.32.A': 376.54166666666686,
'48.29.A': 352.4166666666668,
```

```
'49.26.A': 125.16666666666671,
'47.31.A': 372.50000000000017,
'48.28.A': 348.3750000000001,
'49.25.A': 129.20833333333337,
'47.30.A': 368.4583333333335,
'48.27.A': 344.3333333333334,
'50.24.A': 121.25000000000006,
'47.29.A': 364.4166666666668,
'48.26.A': 340.29166666666674,
'50.23.A': 125.29166666666673,
'47.28.A': 360.3750000000001,
'48.25.A': 336.25000000000006,
'50.22.A': 129.3333333333334,
'47.27.A': 356.3333333333334,
'49.24.A': 133.25000000000006,
'50.21.A': 133.37500000000006,
'47.26.A': 352.29166666666674,
'49.23.A': 137.29166666666674,
'50.20.A': 137.4166666666667,
'47.25.A': 348.25000000000006,
'49.22.A': 141.3333333333334,
'50.19.A': 141.45833333333337,
'48.24.A': 332.20833333333337,
'49.21.A': 145.37500000000006,
'50.18.A': 145.50000000000003,
'48.23.A': 328.1666666666667,
'49.20.A': 149.4166666666667,
'50.17.A': 149.54166666666669,
'48.22.A': 324.125,
'49.19.A': 153.45833333333337,
'50.16.A': 153.58333333333334,
'47.24.A': 344.20833333333337,
'48.21.A': 320.0833333333333,
'49.18.A': 157.50000000000003,
'50.15.A': 157.625,
'47.23.A': 340.1666666666667,
'48.20.A': 316.04166666666663,
'49.17.A': 161.54166666666669,
'50.14.A': 161.66666666666666,
'47.22.A': 336.125,
'48.19.A': 311.99999999999994,
'49.16.A': 165.58333333333334,
'50.13.A': 165.70833333333331,
'47.21.A': 332.0833333333333,
'48.18.A': 307.95833333333326,
'49.15.A': 169.625,
'50.12.A': 169.74999999999997,
'47.20.A': 328.04166666666663,
'48.17.A': 303.9166666666666,
```

```
'49.14.A': 173.66666666666666,
'50.11.A': 173.79166666666663,
'47.19.A': 323.99999999999994,
'48.16.A': 299.8749999999999,
'49.13.A': 177.70833333333331,
'50.10.A': 177.8333333333333,
'47.18.A': 319.95833333333326,
'48.15.A': 295.8333333333332,
'49.12.A': 181.74999999999997,
'50.09.A': 181.87499999999994,
'47.17.A': 315.9166666666666,
'48.14.A': 291.7916666666665,
'49.11.A': 185.79166666666663,
'50.08.A': 185.9166666666666,
'47.16.A': 311.8749999999999,
'48.13.A': 287.74999999999983,
'49.10.A': 189.8333333333333,
'50.07.A': 189.95833333333326,
'47.15.A': 307.8333333333332,
'48.12.A': 283.70833333333314,
'49.09.A': 193.87499999999994,
'50.06.A': 193.99999999999991,
'47.14.A': 303.7916666666665,
'48.11.A': 279.66666666666646,
'49.08.A': 197.9166666666666,
'50.05.A': 198.04166666666657,
'47.13.A': 299.74999999999983,
'48.10.A': 275.6249999999998,
'49.07.A': 201.95833333333326,
'50.04.A': 202.08333333333323,
'47.12.A': 295.70833333333314,
'48.09.A': 271.58333333333314,
'49.06.A': 205.99999999999991,
'50.03.A': 206.1249999999999,
'47.11.A': 291.66666666666646,
'48.08.A': 267.54166666666646,
'49.05.A': 210.04166666666657,
'50.02.A': 210.16666666666654,
'47.10.A': 287.6249999999998,
'48.07.A': 263.4999999999998,
'49.04.A': 214.08333333333323,
'50.01.A': 214.2083333333332,
'47.09.A': 283.58333333333314,
'48.06.A': 259.45833333333314,
'49.03.A': 218.1249999999999,
'47.08.A': 279.54166666666646,
'48.05.A': 255.4166666666665,
'49.02.A': 222.16666666666654,
'47.07.A': 275.4999999999998,
```

```
'48.04.A': 251.37499999999983,
'49.01.A': 226.2083333333332,
'47.06.A': 271.45833333333314,
'48.03.A': 247.33333333333317,
'47.05.A': 267.4166666666665,
'48.02.A': 243.29166666666652,
'47.04.A': 263.37499999999983,
'48.01.A': 239.24999999999986,
'47.03.A': 259.33333333333314,
'47.02.A': 255.29166666666652,
'47.01.A': 251.24999999999986}))]
```

In [329…

```
##STAGE 1
from pyomo.environ import *

noOfPrd = len(product)
supply = dict(zip(product, [No_of_Loc_per_product]*noOfPrd))

noOfLoc = len(location)
demand = dict(zip(location, [1]*noOfLoc))

orderFreq_dict = pd.Series(Products['Order Count'].values,index=Products.MATERIAL_NUMBER).to_dict()

supply["DUMMY_PART"] = 0
diff = sum(demand.values()) - sum(supply.values()) #comparing supply and demand
if 'DUMMY_PART' not in product:
    product.append("DUMMY_PART")
supply["DUMMY_PART"] = diff
orderFreq_dict["DUMMY_PART"] = 0

# instantiate Concrete Model
model = ConcreteModel()

# define variables
model.X = Var(product, location, domain=NonNegativeReals)

# define objective function
model.total_distance = Objective(expr=sum(orderFreq_dict[p] * distance_IO[l] * model.X[p, l]
                                          for p in product
                                          for l in location),
                                 sense=minimize)

# define constraints
model.supply_ct = ConstraintList()
for p in product:
    model.supply_ct.add(sum(model.X[p, l] for l in location) == supply[p])
```

```
model.demand_ct = ConstraintList()
for l in location:
    model.demand_ct.add(
        sum(model.X[p, l] for p in product) == demand[l])
```

In [330…
```
# solve
solver = SolverFactory('glpk')
solver.solve(model)
```

Out[330…
{'Problem': [{'Name': 'unknown', 'Lower bound': 1535940.16666667, 'Upper bound': 1535940.16666667, 'Number of objectives': 1, 'Number of constraints': 318, 'Number of variables': 21817, 'Number of nonzeros': 43633, 'Sense': 'minimize'}], 'Solver': [{'Status': 'ok', 'Termination condition': 'optimal', 'Statistics': {'Branch and bound': {'Number of bounded subproblems': 0, 'Number of created subproblems': 0}}, 'Error rc': 0, 'Time': 0.4175701141357422}], 'Solution': [OrderedDict([('number of solutions', 0), ('number of solutions displayed', 0)])]}

In [331…
```
# convert model into a Pandas data frame for nicer display
import pandas as pd
assignment = pd.DataFrame(0, index=product, columns=location)
STG1_assignment = defaultdict(dict)
cols = ['Product', 'Distance']
STG1_distance_df = pd.DataFrame(columns = cols)
for p in product:
    STG1_assignment[p] = []
    for l in location:
        if model.X[p, l].value > 0:
            assignment.loc[p, l] = 'S'
            STG1_assignment[p] += [l]
            if p != 'DUMMY_PART':
                STG1_distance_df = STG1_distance_df.append({'Product': p, 'Distance IO':distance_IO[l]},
                                                           ignore_index=True)
        else:
            assignment.loc[p, l] = ''

# display
print(f"\nTotal Distance minimized to = {model.total_distance():,.2f}")
```

```
Total Distance minimized to = 1,535,940.17
```

In [332…
```
print("Final assignment: ")
assignment
```

```
Final assignment:
```

Out[332…

| | 50.54.A | 50.53.A | 50.52.A | 49.54.A | 50.51.A | 49.53.A | 50.50.A | 49.52.A | 50.49.A | 48.54.A | ... | 49.01.A | 47.06.A | 48.( |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

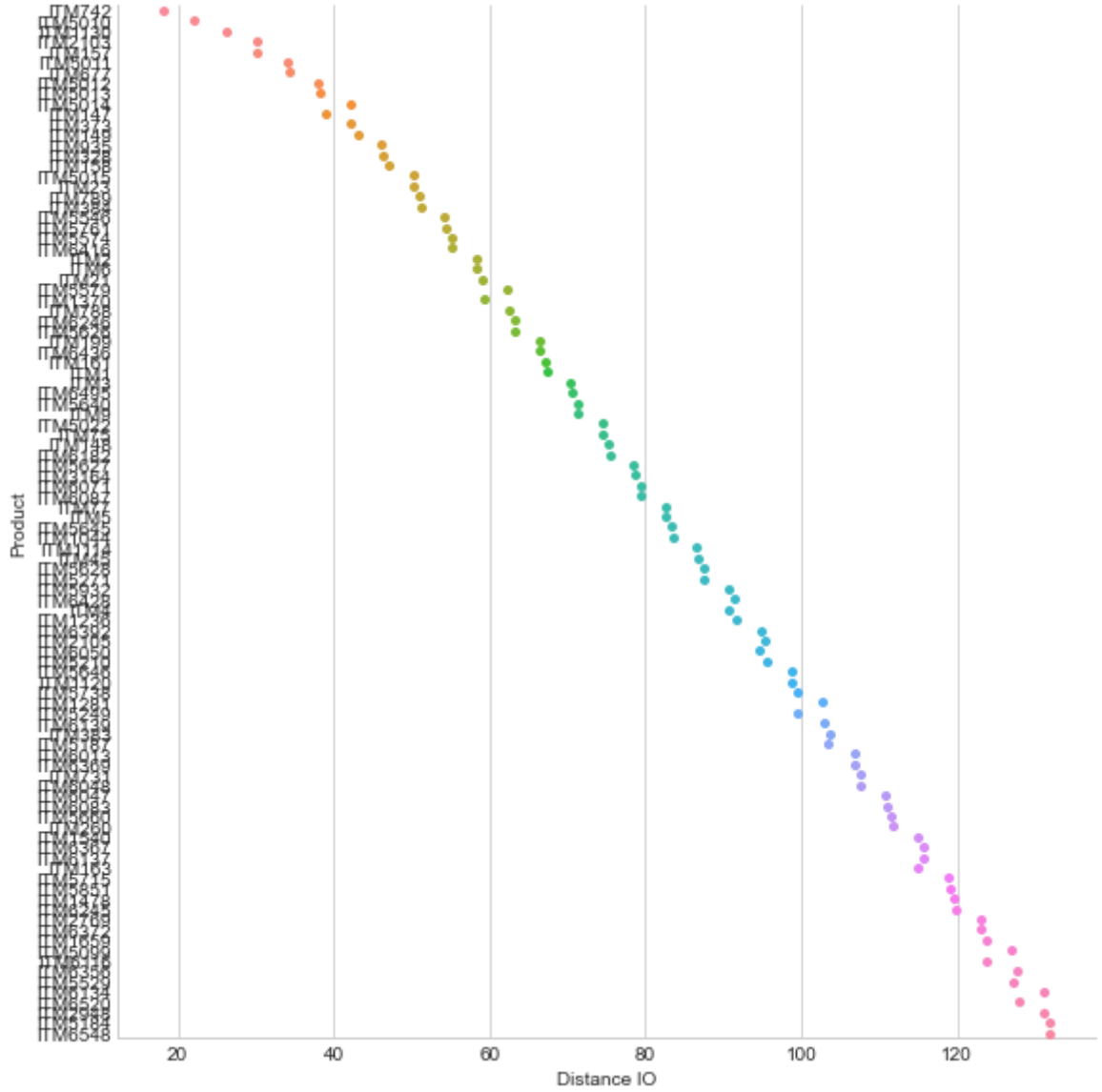
| | 50.54.A | 50.53.A | 50.52.A | 49.54.A | 50.51.A | 49.53.A | 50.50.A | 49.52.A | 50.49.A | 48.54.A | ... | 49.01.A | 47.06.A | 48.( |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ITM742** | S | | | | | | | | | | ... | | | |
| **ITM5010** | | S | | | | | | | | | ... | | | |
| **ITM1130** | | | S | | | | | | | | ... | | | |
| **ITM2103** | | | | S | | | | | | | ... | | | |
| **ITM157** | | | | | S | | | | | | ... | | | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **ITM6520** | | | | | | | | | | | ... | | | |
| **ITM2948** | | | | | | | | | | | ... | | | |
| **ITM5184** | | | | | | | | | | | ... | | | |
| **ITM6548** | | | | | | | | | | | ... | | | |
| **DUMMY_PART** | | | | | | | | | | | ... | S | S | |

101 rows × 216 columns

In [333…
```
assignment.to_csv('STG1_result.csv')
```

In [334…
```
sns.catplot(x="Distance IO", y="Product", data=STG1_distance_df,kind='swarm', height=8, aspect=1)
```

Out[334…
```
<seaborn.axisgrid.FacetGrid at 0x7fe351f52d30>
```




In [335…
```
rules['antecedents'] = rules['antecedents'].astype(str)
rules['consequents'] = rules['consequents'].astype(str)

rules = rules[rules['antecedents'].isin(product)]
rules = rules[rules['consequents'].isin(product)]
```

In [336…
```
distinctOrder_freq['MATERIAL_NUMBER'] = distinctOrder_freq['MATERIAL_NUMBER'].astype(str)
```

In [337…

```
##STAGE 2
product = Products.head(No_of_Products)['MATERIAL_NUMBER'].values.tolist() #list of products
noOfPrd = len(product)
demand = dict(zip(product, [No_of_Loc_per_product]*noOfPrd)) #number of locations needed. Time being 1 per proc

confidence_list = defaultdict(dict)  #confidence_list[p1][p2] will give confidence that p2 is bought for each p
for ind, rec in rules.iterrows():
    ant = (rec['antecedents'])
    con = (rec['consequents'])
    conf = rec['confidence']
    confidence_list[ant][con]= conf

#relative order freq
order_freq = pd.Series(Products['Order Count'].values,index=Products.MATERIAL_NUMBER).to_dict()

location = locations['Location'].values.tolist()  #list of all locations
noOfLoc = len(location)
supply = dict(zip(location,
                  [1]*noOfLoc)) #number of products that can be assigned to a location. Default 1 - one product

demand["DUMMY_PART"] = 0
diff = sum(supply.values()) - sum(demand.values()) #comparing supply and demand
if 'DUMMY_PART' not in product:
    product.append("DUMMY_PART") #adding dummy product as supply is more than demand
demand["DUMMY_PART"] = diff #assign these many extra locations to dummy part
order_freq["DUMMY_PART"] = 0 #no sales for dummy part

Aff_wt = 1 #to control weightage of product affinity in the optimization equation

############################################################
#all product pairs should have confidence. If a product pair is not in input file (Apriori output), assign ZERO
for p in product:
    for c in product:
        if c not in confidence_list[p]:
            if p==c:
                confidence_list[p][c] = 1 #confidence of 1 when product pair is itself
            else:
                confidence_list[p][c] = 0 #confidence of 0 when product pair has not been purchased together (a
############################################################

# instantiate Concrete Model
model = ConcreteModel()

# define variables
model.X = Var(product, location, domain=NonNegativeIntegers)
```

```
# define objective function
model.total_distance = Objective(expr=sum(order_freq[p] * distance_IO[l] * model.X[p, l]
                                          for p in product
                                          for l in location)
                                 +
                                 sum(order_freq[pc] * distance_IO[lc] * model.X[pc, lc]
                                     for pc in product
                                     for lc in location)
                                 +
                                 Aff_wt *
                                 sum(confidence_list[p][pc] * order_freq[p]
                                     * abs(distance_BTW[l][lc])
                                     * model.X[p, l] * model.X[pc, lc]
                                      for p in product
                                      for l in location
                                      for pc in product
                                      for lc in location),
                                 sense=minimize)

# define constraints
model.supply_ct = ConstraintList()
for l in location:
    model.supply_ct.add(
        sum(model.X[p, l] for p in product) == supply[l])

model.demand_ct = ConstraintList()
for p in product:
    model.demand_ct.add(sum(model.X[p, l] for l in location) == demand[p])
```

In [338…

```
# solve
solver = SolverFactory('ipopt')
solver.solve(model)
```

Out[338…

```
{'Problem': [{'Lower bound': -inf, 'Upper bound': inf, 'Number of objectives': 1, 'Number of constraints': 317,
'Number of variables': 21816, 'Sense': 'unknown'}], 'Solver': [{'Status': 'ok', 'Message': 'Ipopt 3.12.12\\x3a
Optimal Solution Found', 'Termination condition': 'optimal', 'Id': 0, 'Error rc': 0, 'Time': 3825.974211931228
6}], 'Solution': [OrderedDict([('number of solutions', 0), ('number of solutions displayed', 0)])]}
```

In [339…

```
# convert model into a Pandas data frame for nicer display
assignment = pd.DataFrame(0, index=product, columns=location)
for p in product:
    for l in location:
        assignment.loc[p, l] = model.X[p, l].value
```

```
for p in product:
    dmd = demand[p]
    for l in location:
        if assignment.loc[p, l] > 0.1:
            if dmd != 0:
                assignment.loc[:, l] = 0
                assignment.loc[p, l] = 1
                dmd -= 1
            else:
                assignment.loc[p, l] = 0

STG2_assignment = defaultdict(dict)
cols = ['Product', 'Distance']
STG2_distance_df = pd.DataFrame(columns = cols)
for p in product:
    STG2_assignment[p] = []
    for l in location:
        if assignment.loc[p, l] > 0:
            assignment.loc[p, l] = 'S'
            STG2_assignment[p] += [l]
            if p != 'DUMMY_PART':
                STG2_distance_df = STG2_distance_df.append({'Product': p, 'Distance IO':distance_IO[l]},
                                                           ignore_index=True)
        else:
            assignment.loc[p, l] = ''

# display
print(f"\nThe best distance that can be achived = {model.total_distance():,.2f}")
print("Final assignment: ")
assignment
```

```
The best distance that can be achived = 3,292,878.41
Final assignment:
```

Out[339…

| | 50.54.A | 50.53.A | 50.52.A | 49.54.A | 50.51.A | 49.53.A | 50.50.A | 49.52.A | 50.49.A | 48.54.A | ... | 49.01.A | 47.06.A | 48.( |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ITM742** | | | | | | | | | S | | ... | | | |
| **ITM5010** | S | | | | | | | | | | ... | | | |
| **ITM1130** | | | | | | | | | | | ... | | | |
| **ITM2103** | | S | | | | | | | | | ... | | | |
| **ITM157** | | | S | | | | | | | | ... | | | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

| | 50.54.A | 50.53.A | 50.52.A | 49.54.A | 50.51.A | 49.53.A | 50.50.A | 49.52.A | 50.49.A | 48.54.A | ... | 49.01.A | 47.06.A | 48.( |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ITM6520** | | | | | | | | | | | ... | | | |
| **ITM2948** | | | | | | | | | | | ... | | | |
| **ITM5184** | | | | | | | | | | | ... | | | |
| **ITM6548** | | | | | | | | | | | ... | | | |
| **DUMMY_PART** | | | | | | | | | | | ... | S | S | |

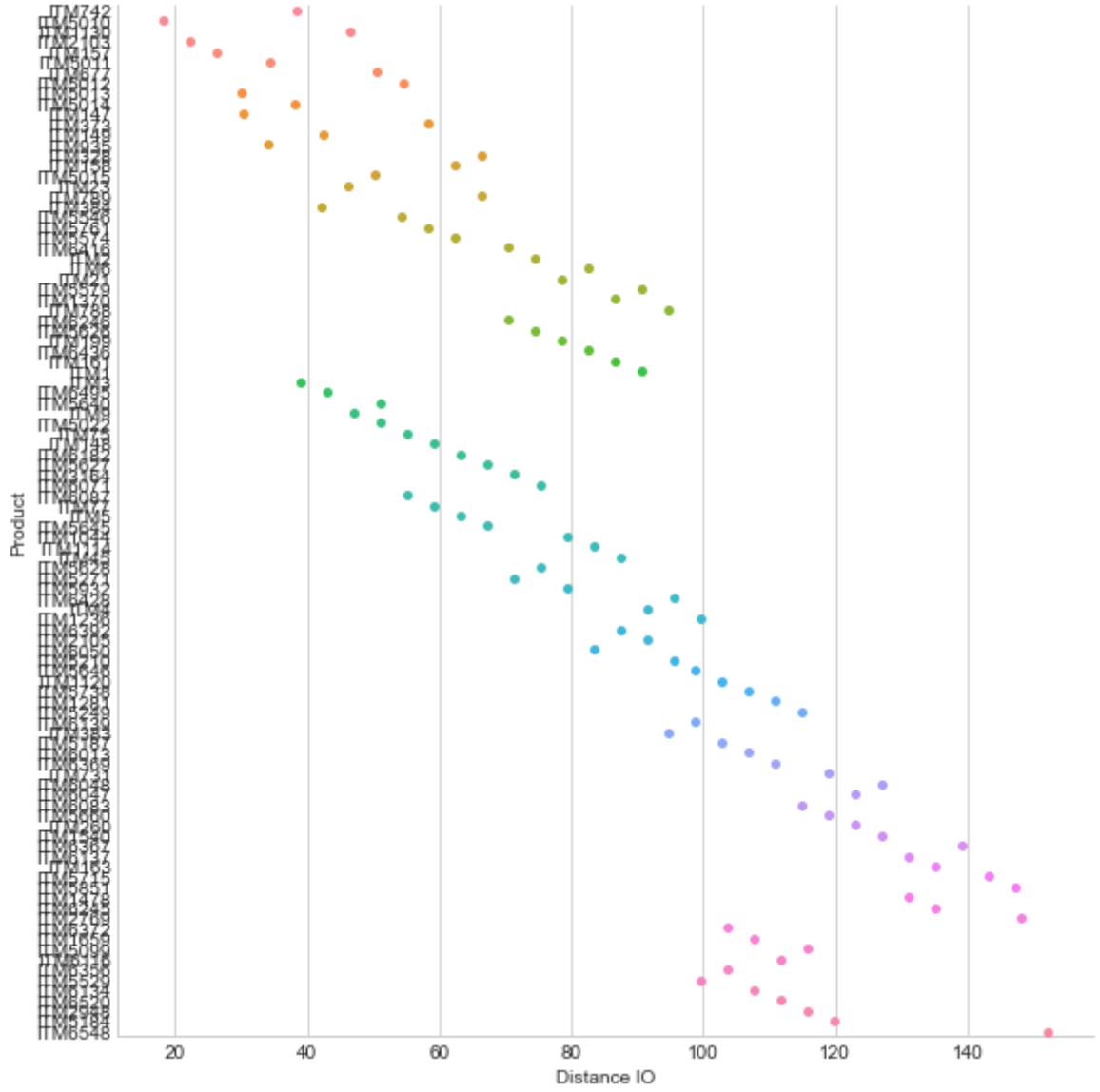
101 rows × 216 columns

In [340…
```
assignment.to_csv('STG2_result.csv')
```

In [341…
```
# convert model into a Pandas data frame for nicer display
assignment = pd.DataFrame(0, index=product, columns=location)
for p in product:
    for l in location:
        assignment.loc[p, l] = model.X[p, l].value

assignment.to_csv('STG2_result_raw.csv')
```

In [342…
```
sns.catplot(x="Distance IO", y="Product", data=STG2_distance_df,kind='swarm', height=8, aspect=1)
```

Out[342… <seaborn.axisgrid.FacetGrid at 0x7fe298621c10>




In [343…
```
pick_orders = df_salesOrder[df_salesOrder['MATERIAL_NUMBER'].isin(product)]
pick_orders.drop(['SALES_ORDER_ITEM_CREATE_DATE_FISCAL_WEEK', 'SALES_ORDER_ITEM_CREATE_DATE_FISCAL_YEAR',
                  'SALES_ORDER_ITEM','SCHEDULE_QUANTITY'], axis=1)
pick_orders = pick_orders[['MATERIAL_NUMBER', 'SALES_ORDER']].drop_duplicates()
pick_orders = pick_orders.sort_values(by=['SALES_ORDER'], ascending=True)
```

In [344…
```
pick_orders.shape
```

Out[344… (22071, 2)

In [345…
```
Total_orders_toPick = pick_orders.SALES_ORDER.nunique()
Total_orders_toPick
```

Out[345… 10195

In [346…
```
Total_picks = pick_orders.shape[0]
Total_picks
```

Out[346… 22071

In [347…
```
order_list = pick_orders.SALES_ORDER.unique().tolist()
```

In [348…
```
stage = 'STG1'
STG_Assignment = STG1_assignment

pick_orders['Stage'] = [stage]*Total_picks
pick_orders['Location'] = ['']*Total_picks
pick_orders['Sort Field'] = [0]*Total_picks
for ind,rec in pick_orders.iterrows():
    p = rec['MATERIAL_NUMBER']
    l = STG_Assignment[p][0]
    sf = int(Locations[Locations['Location'] == l]['Sort Field'])
    pick_orders.loc[pick_orders['MATERIAL_NUMBER'] == p, 'Location'] = l
    pick_orders.loc[pick_orders['MATERIAL_NUMBER'] == p, 'Sort Field'] = sf

pick_orders = pick_orders.sort_values(by=['SALES_ORDER', 'Sort Field'], ascending=True)
pick_orders

Pick_time = pd.DataFrame(order_list, columns=['Sales Order'])
Pick_time['Stage'] = [stage] * Total_orders_toPick
Pick_time['Total Distance'] = [stage] * Total_orders_toPick
for order in order_list:
    total_dist = 0
    df = pick_orders[pick_orders['SALES_ORDER'] == order]
    first_time = True
    for ind, rec in df.iterrows():
        if first_time:
            first_time = False
            l = str(rec['Location'])
            total_dist += distance_IO[l]
            #print("first time: ",order, total_dist)
```

```
            l1 = l
            l2 = l
        else:
            l2 = str(rec['Location'])
            total_dist += abs(distance_BTW[l1][l2])
            #print("next  time: ",order, total_dist)
    total_dist += distance_IO[l2]
    #print("last  time: ",order, total_dist)
    Pick_time.loc[Pick_time['Sales Order'] == order, 'Total Distance'] = total_dist

print(Pick_time)
tot_pick_dist = Pick_time['Total Distance'].sum()
print(f"\nTotal pick distance for {stage} = {tot_pick_dist:,.2f}")
```

```
      Sales Order Stage Total Distance
0        27997929  STG1           84.5
1        28018206  STG1           84.5
2        28025370  STG1        231.583
3        28057994  STG1        231.583
4        28100693  STG1          223.5
...           ...   ...            ...
10190    55875321  STG1        141.083
10191    55875328  STG1        2748.58
10192    55875339  STG1        165.083
10193    55875348  STG1        1032.25
10194    55875362  STG1        1052.46

[10195 rows x 3 columns]

Total pick distance for STG1 = 3,924,439.00
```

In [349…

```
Pick_time.to_csv('STG1_order pick times.csv')
```

In [350…

```
stage = 'STG2'
STG_Assignment = STG2_assignment

pick_orders['Stage'] = [stage]*Total_picks
pick_orders['Location'] = ['']*Total_picks
pick_orders['Sort Field'] = [0]*Total_picks
for ind,rec in pick_orders.iterrows():
    p = rec['MATERIAL_NUMBER']
    l = STG_Assignment[p][0]
    sf = int(Locations[Locations['Location'] == l]['Sort Field'])
    pick_orders.loc[pick_orders['MATERIAL_NUMBER'] == p, 'Location'] = l
    pick_orders.loc[pick_orders['MATERIAL_NUMBER'] == p, 'Sort Field'] = sf
```

```
pick_orders = pick_orders.sort_values(by=['SALES_ORDER', 'Sort Field'], ascending=True)
pick_orders

Pick_time = pd.DataFrame(order_list, columns=['Sales Order'])
Pick_time['Stage'] = [stage] * Total_orders_toPick
Pick_time['Total Distance'] = [stage] * Total_orders_toPick
for order in order_list:
    total_dist = 0
    df = pick_orders[pick_orders['SALES_ORDER'] == order]
    first_time = True
    for ind, rec in df.iterrows():
        if first_time:
            first_time = False
            l = str(rec['Location'])
            total_dist += distance_IO[l]
            #print("first time: ",order, total_dist)
            l1 = l
            l2 = l
        else:
            l2 = str(rec['Location'])
            total_dist += abs(distance_BTW[l1][l2])
            #print("next  time: ",order, total_dist)
    total_dist += distance_IO[l2]
    #print("last  time: ",order, total_dist)
    Pick_time.loc[Pick_time['Sales Order'] == order, 'Total Distance'] = total_dist

print(Pick_time)
tot_pick_dist = Pick_time['Total Distance'].sum()
print(f"\nTotal pick distance for {stage} = {tot_pick_dist:,.2f}")
```

```
      Sales Order Stage Total Distance
0        27997929  STG2        116.833
1        28018206  STG2        116.833
2        28025370  STG2        262.083
3        28057994  STG2        262.083
4        28100693  STG2        246.167
...           ...   ...            ...
10190    55875321  STG2        86.0833
10191    55875328  STG2        765.083
10192    55875339  STG2        510.458
10193    55875348  STG2        294.667
10194    55875362  STG2         671.75

[10195 rows x 3 columns]

Total pick distance for STG2 = 3,404,703.46
```

In [351…
```
Pick_time.to_csv('STG2_order pick times.csv')
```

In [352…
```
##STAGE 3
product = Products.head(No_of_Products)['MATERIAL_NUMBER'].values.tolist() #list of products
noOfPrd = len(product)
demand = dict(zip(product, [No_of_Loc_per_product]*noOfPrd)) #number of locations needed. Time being 1 per proc

confidence_list = defaultdict(dict)  #confidence_list[p1][p2] will give confidence that p2 is bought for each p
for ind, rec in rules.iterrows():
    ant = (rec['antecedents'])
    con = (rec['consequents'])
    conf = rec['confidence']
    if conf > .5:
        confidence_list[ant][con]= conf
    else:
        confidence_list[ant][con]= 0

#relative order freq
order_freq = pd.Series(Products['Order Count'].values,index=Products.MATERIAL_NUMBER).to_dict()

location = locations['Location'].values.tolist()  #list of all locations
noOfLoc = len(location)
supply = dict(zip(location,
                  [1]*noOfLoc)) #number of products that can be assigned to a location. Default 1 - one product

demand["DUMMY_PART"] = 0
diff = sum(supply.values()) - sum(demand.values()) #comparing supply and demand
if 'DUMMY_PART' not in product:
    product.append("DUMMY_PART") #adding dummy product as supply is more than demand
demand["DUMMY_PART"] = diff #assign these many extra locations to dummy part
order_freq["DUMMY_PART"] = 0 #no sales for dummy part

Aff_wt = 1 #to control weightage of product affinity in the optimization equation

############################################################
#all product pairs should have confidence. If a product pair is not in input file (Apriori output), assign ZERO
for p in product:
    for c in product:
        if c not in confidence_list[p]:
            if p==c:
                confidence_list[p][c] = 1 #confidence of 1 when product pair is itself
            else:
                confidence_list[p][c] = 0 #confidence of 0 when product pair has not been purchased together (a
############################################################
```

```
# instantiate Concrete Model
model = ConcreteModel()

# define variables
model.X = Var(product, location, domain=NonNegativeIntegers)

# define objective function
model.total_distance = Objective(expr=sum(order_freq[p] * distance_IO[l] * model.X[p, l]
                                          for p in product
                                          for l in location)
                                 +
                                 sum(order_freq[pc] * distance_IO[lc] * model.X[pc, lc]
                                     for pc in product
                                     for lc in location)
                                 +
                                 Aff_wt *
                                 sum(confidence_list[p][pc] * order_freq[p]
                                     * abs(distance_BTW[l][lc])
                                     * model.X[p, l] * model.X[pc, lc]
                                      for p in product
                                      for l in location
                                      for pc in product
                                      for lc in location),
                                 sense=minimize)

# define constraints
model.supply_ct = ConstraintList()
for l in location:
    model.supply_ct.add(
        sum(model.X[p, l] for p in product) == supply[l])

model.demand_ct = ConstraintList()
for p in product:
    model.demand_ct.add(sum(model.X[p, l] for l in location) == demand[p])
```

In [353…

```
# solve
solver = SolverFactory('ipopt')
solver.solve(model)
```

Out[353…

```
{'Problem': [{'Lower bound': -inf, 'Upper bound': inf, 'Number of objectives': 1, 'Number of constraints': 317,
'Number of variables': 21816, 'Sense': 'unknown'}], 'Solver': [{'Status': 'ok', 'Message': 'Ipopt 3.12.12\\x3a
Optimal Solution Found', 'Termination condition': 'optimal', 'Id': 0, 'Error rc': 0, 'Time': 1182.537739038467
4}], 'Solution': [OrderedDict([('number of solutions', 0), ('number of solutions displayed', 0)])]}
```

In [354…

```
# convert model into a Pandas data frame for nicer display
```

```
assignment = pd.DataFrame(0, index=product, columns=location)
for p in product:
    for l in location:
        assignment.loc[p, l] = model.X[p, l].value

for p in product:
    dmd = demand[p]
    for l in location:
        if assignment.loc[p, l] > 0.1:
            if dmd != 0:
                assignment.loc[:, l] = 0
                assignment.loc[p, l] = 1
                dmd -= 1
            else:
                assignment.loc[p, l] = 0

STG3_assignment = defaultdict(dict)
cols = ['Product', 'Distance']
STG3_distance_df = pd.DataFrame(columns = cols)
for p in product:
    STG3_assignment[p] = []
    for l in location:
        if assignment.loc[p, l] > 0:
            assignment.loc[p, l] = 'S'
            STG3_assignment[p] += [l]
            if p != 'DUMMY_PART':
                STG3_distance_df = STG3_distance_df.append({'Product': p, 'Distance IO':distance_IO[l]},
                                                           ignore_index=True)
        else:
            assignment.loc[p, l] = ''

# display
print(f"\nThe best distance that can be achived = {model.total_distance():,.2f}")
print("Final assignment: ")
assignment
```

```
The best distance that can be achived = 3,203,268.32
Final assignment:
```

Out[354…

| | 50.54.A | 50.53.A | 50.52.A | 49.54.A | 50.51.A | 49.53.A | 50.50.A | 49.52.A | 50.49.A | 48.54.A | ... | 49.01.A | 47.06.A | 48.( |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ITM742** | S | | | | | | | | | | ... | | | |
| **ITM5010** | | S | | | | | | | | | ... | | | |
| **ITM1130** | | | S | | | | | | | | ... | | | |

| | 50.54.A | 50.53.A | 50.52.A | 49.54.A | 50.51.A | 49.53.A | 50.50.A | 49.52.A | 50.49.A | 48.54.A | ... | 49.01.A | 47.06.A | 48.( |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ITM2103** | | | | S | | | | | | | ... | | | |
| **ITM157** | | | | | | S | | | | | ... | | | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **ITM6520** | | | | | | | | | | | ... | | | |
| **ITM2948** | | | | | | | | | | | ... | | | |
| **ITM5184** | | | | | | | | | | | ... | | | |
| **ITM6548** | | | | | | | | | | | ... | | | |
| **DUMMY_PART** | | | | | | | | | | | ... | S | S | |

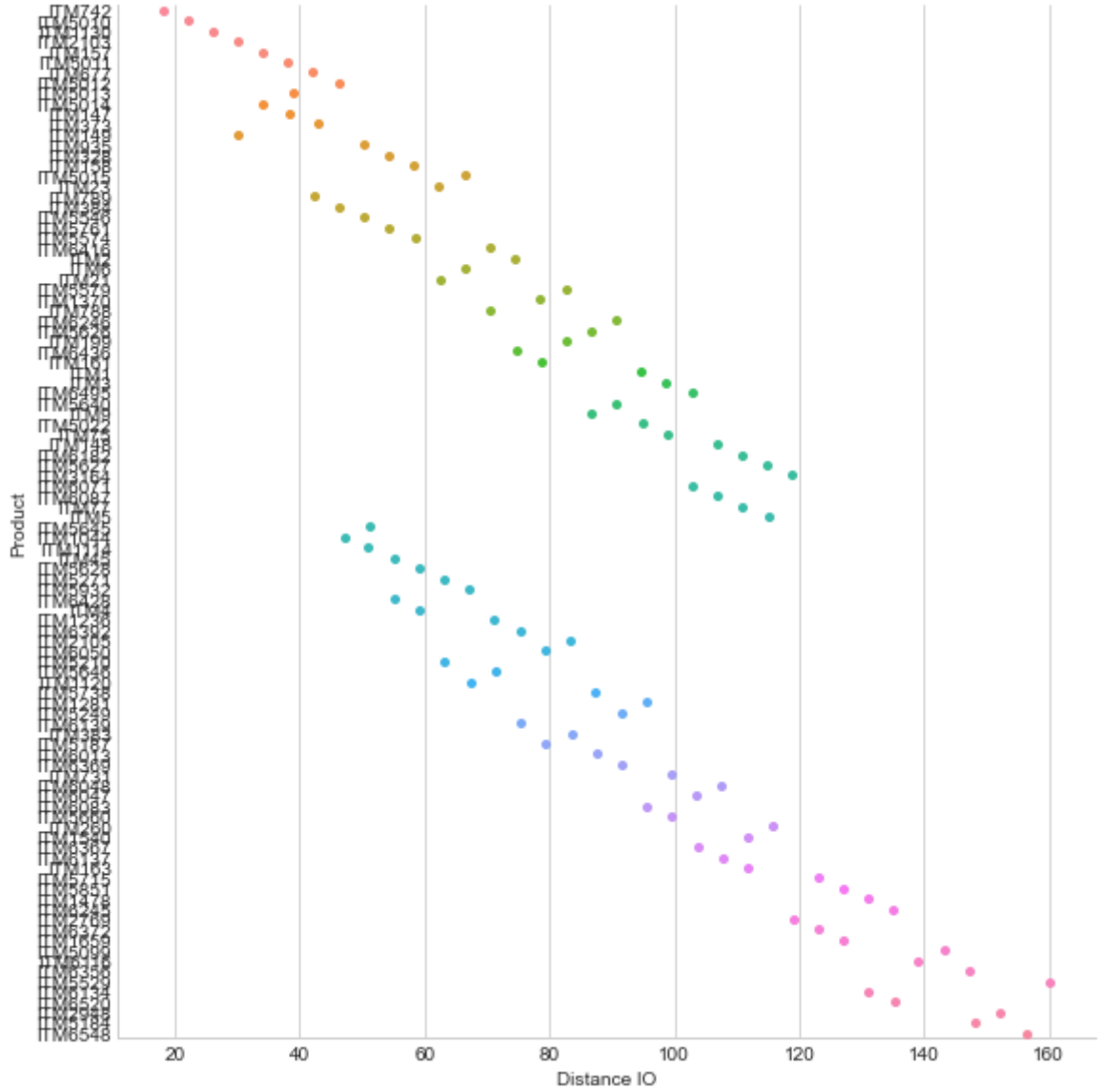
101 rows × 216 columns

In [355…
```
assignment.to_csv('STG3_result.csv')
```

In [356…
```
# convert model into a Pandas data frame for nicer display
assignment = pd.DataFrame(0, index=product, columns=location)
for p in product:
    for l in location:
        assignment.loc[p, l] = model.X[p, l].value

assignment.to_csv('STG3_result_raw.csv')
```

In [357…
```
sns.catplot(x="Distance IO", y="Product", data=STG3_distance_df,kind='swarm', height=8, aspect=1)
```

Out[357… <seaborn.axisgrid.FacetGrid at 0x7fe36bd9e5e0>




In [358…

```
stage = 'STG3'
STG_Assignment = STG3_assignment

pick_orders['Stage'] = [stage]*Total_picks
pick_orders['Location'] = ['']*Total_picks
pick_orders['Sort Field'] = [0]*Total_picks
for ind,rec in pick_orders.iterrows():
    p = rec['MATERIAL_NUMBER']
```

```
    l = STG_Assignment[p][0]
    sf = int(Locations[Locations['Location'] == l]['Sort Field'])
    pick_orders.loc[pick_orders['MATERIAL_NUMBER'] == p, 'Location'] = l
    pick_orders.loc[pick_orders['MATERIAL_NUMBER'] == p, 'Sort Field'] = sf

pick_orders = pick_orders.sort_values(by=['SALES_ORDER', 'Sort Field'], ascending=True)
pick_orders

Pick_time = pd.DataFrame(order_list, columns=['Sales Order'])
Pick_time['Stage'] = [stage] * Total_orders_toPick
Pick_time['Total Distance'] = [stage] * Total_orders_toPick
for order in order_list:
    total_dist = 0
    df = pick_orders[pick_orders['SALES_ORDER'] == order]
    first_time = True
    for ind, rec in df.iterrows():
        if first_time:
            first_time = False
            l = str(rec['Location'])
            total_dist += distance_IO[l]
            #print("first time: ",order, total_dist)
            l1 = l
            l2 = l
        else:
            l2 = str(rec['Location'])
            total_dist += abs(distance_BTW[l1][l2])
            #print("next  time: ",order, total_dist)
    total_dist += distance_IO[l2]
    #print("last  time: ",order, total_dist)
    Pick_time.loc[Pick_time['Sales Order'] == order, 'Total Distance'] = total_dist

print(Pick_time)
tot_pick_dist = Pick_time['Total Distance'].sum()
print(f"\nTotal pick distance for {stage} = {tot_pick_dist:,.2f}")
```

```
      Sales Order Stage Total Distance
0        27997929  STG3        86.0833
1        28018206  STG3        86.0833
2        28025370  STG3         215.417
3        28057994  STG3         215.417
4        28100693  STG3         231.583
...           ...   ...             ...
10190    55875321  STG3           205.5
10191    55875328  STG3         1266.67
10192    55875339  STG3           205.5
10193    55875348  STG3          947.75
```

```
10194    55875362  STG3        411.375

[10195 rows x 3 columns]

Total pick distance for STG3 = 3,366,151.96
```

In [359…
```
Pick_time.to_csv('STG3_order pick times.csv')
```

In [360…
```
Current_WH_loc_assignment = pd.read_excel('Input Data/Current Material Location.XLSX')
```

```
/Users/shyamsrikumar/opt/anaconda3/lib/python3.8/site-packages/xlrd/xlsx.py:266: DeprecationWarning: This metho
d will be removed in future versions.  Use 'tree.iter()' or 'list(tree.iter())' instead.
  for elem in self.tree.iter() if Element_has_iter else self.tree.getiterator():
/Users/shyamsrikumar/opt/anaconda3/lib/python3.8/site-packages/xlrd/xlsx.py:312: DeprecationWarning: This metho
d will be removed in future versions.  Use 'tree.iter()' or 'list(tree.iter())' instead.
  for elem in self.tree.iter() if Element_has_iter else self.tree.getiterator():
/Users/shyamsrikumar/opt/anaconda3/lib/python3.8/site-packages/xlrd/xlsx.py:266: DeprecationWarning: This metho
d will be removed in future versions.  Use 'tree.iter()' or 'list(tree.iter())' instead.
  for elem in self.tree.iter() if Element_has_iter else self.tree.getiterator():
```

In [361…
```
Current_WH_loc_assignment = Current_WH_loc_assignment.sort_values(by=['Storage Bin'], ascending=False)
current_assignment = Current_WH_loc_assignment[~Current_WH_loc_assignment.isnull().any(axis=1)]
current_assignment
```

Out[361…

| | Storage Bin | level | Material |
|---|---|---|---|
| **3716** | 50.51.D | D | ITM866 |
| **10829** | 50.51.C | C | ITM4591 |
| **10546** | 50.51.B | B | ITM4402 |
| **1377** | 50.51.A | A | ITM5015 |
| **3715** | 50.50.D | D | ITM865 |
| **...** | ... | ... | ... |
| **9632** | 01.21.C | C | ITM3981 |
| **3194** | 01.05.B | B | ITM434 |
| **8515** | 01.01.D | D | ITM3256 |
| **8412** | 01.01.C | C | ITM3195 |
| **3193** | 01.01.B | B | ITM433 |

8830 rows × 3 columns

In [362…

```
Unassigned_locations = Current_WH_loc_assignment[Current_WH_loc_assignment.isnull().any(axis=1)]
Unassigned_locations = Unassigned_locations[(Unassigned_locations['level'] == 'A')|(Unassigned_locations['level
Unassigned_locations
```

Out[362…

| | Storage Bin | level | Material |
|---|---|---|---|
| **1387** | 50.54.B | B | NaN |
| **1386** | 50.54.A | A | NaN |
| **1383** | 50.53.B | B | NaN |
| **1382** | 50.53.A | A | NaN |
| **1379** | 50.52.B | B | NaN |
| **...** | ... | ... | ... |
| **2089** | 01.03.B | B | NaN |
| **2088** | 01.03.A | A | NaN |
| **2085** | 01.02.B | B | NaN |
| **2084** | 01.02.A | A | NaN |
| **2083** | 01.01.A | A | NaN |

1133 rows × 3 columns

In [363…

```
STG0_assignment = defaultdict(dict)
for p in product:
    STG0_assignment[p] = []

for ind, rec in current_assignment.iterrows():
    p = rec['Material']
    l = rec['Storage Bin']
    if p in product:
        STG0_assignment[p] += [l]

for p in product:
    l = STG0_assignment[p]
    if len(l) == 0:
        l1 = Unassigned_locations['Storage Bin'].iat[0]
        STG0_assignment[p] = [l1]
        Current_WH_loc_assignment.loc[Current_WH_loc_assignment['Storage Bin'] == l1, 'Material'] = p
```

```
        Unassigned_locations = Current_WH_loc_assignment[Current_WH_loc_assignment.isnull().any(axis=1)]
        Unassigned_locations = Unassigned_locations[(Unassigned_locations['level'] == 'A')|(Unassigned_location
```

In [364…

```
STG0_assignment
```

Out[364…

```
defaultdict(dict,
            {'ITM742': ['43.54.A'],
             'ITM5010': ['50.01.A'],
             'ITM1130': ['49.13.A'],
             'ITM2103': ['37.32.A'],
             'ITM157': ['29.11.A'],
             'ITM5011': ['50.22.A', '50.21.A'],
             'ITM677': ['50.36.C'],
             'ITM5012': ['50.31.A'],
             'ITM5013': ['50.32.A'],
             'ITM5014': ['50.50.A', '50.35.A'],
             'ITM147': ['49.32.D',
              '49.31.D',
              '30.16.C',
              '30.14.C',
              '29.25.C',
              '27.03.A'],
             'ITM373': ['48.37.B'],
             'ITM149': ['42.51.A', '27.24.A'],
             'ITM935': ['42.44.D'],
             'ITM328': ['47.04.D'],
             'ITM158': ['40.29.B', '29.32.A'],
             'ITM5015': ['50.51.A'],
             'ITM23': ['48.19.A',
              '48.09.A',
              '47.25.A',
              '43.13.A',
              '28.05.A',
              '28.02.A',
              '28.01.A'],
             'ITM789': ['49.29.A'],
             'ITM384': ['49.31.C'],
             'ITM5546': ['50.38.A'],
             'ITM5761': ['50.39.A'],
             'ITM5574': ['49.03.A', '49.01.A'],
             'ITM6416': ['49.09.A', '49.07.A'],
             'ITM2': ['27.33.A'],
             'ITM6': ['30.09.A', '29.17.A', '29.03.A'],
             'ITM21': ['46.20.A', '42.24.A', '42.08.A', '29.01.A'],
             'ITM5579': ['48.22.A'],
             'ITM1370': ['50.37.A'],
             'ITM788': ['49.29.A', '36.19.A', '29.08.A'],
```

```
'ITM6246': ['50.54.B'],
'ITM5626': ['48.24.A'],
'ITM199': ['39.35.B'],
'ITM6436': ['48.28.A'],
'ITM161': ['30.31.A'],
'ITM1': ['49.28.A',
 '49.02.A',
 '48.21.A',
 '48.08.A',
 '46.20.C',
 '43.14.A',
 '36.17.B',
 '36.13.A',
 '30.50.A',
 '29.08.A',
 '28.34.A',
 '28.19.A',
 '27.25.A',
 '27.05.B'],
'ITM3': ['48.42.A', '44.24.A', '28.49.A', '28.42.A', '27.37.A'],
'ITM6495': ['50.54.A'],
'ITM5640': ['50.53.B'],
'ITM9': ['31.24.A', '31.23.A', '31.19.A', '29.12.A'],
'ITM5022': ['50.53.A'],
'ITM75': ['50.24.B',
 '50.09.A',
 '50.06.D',
 '49.19.A',
 '49.18.A',
 '49.05.A',
 '39.50.B'],
'ITM148': ['30.34.A', '30.25.A', '27.06.A', '27.05.A', '27.04.A'],
'ITM6182': ['50.52.B'],
'ITM5627': ['50.52.A'],
'ITM3164': ['44.39.A'],
'ITM6071': ['50.49.A'],
'ITM6087': ['48.53.A', '48.52.A'],
'ITM77': ['49.29.B'],
'ITM5': ['49.38.B', '49.24.A', '29.06.A'],
'ITM5645': ['50.48.A'],
'ITM1044': ['44.07.D',
 '42.17.C',
 '40.38.C',
 '38.54.A',
 '37.27.B',
 '37.23.B',
 '37.06.B'],
'ITM1114': ['49.04.A', '35.21.A'],
'ITM45': ['47.26.A', '37.44.A'],
```

```
'ITM5628': ['50.43.A'],
'ITM5271': ['50.42.A'],
'ITM5932': ['50.40.A'],
'ITM6428': ['50.29.A'],
'ITM4': ['39.39.D', '28.38.A'],
'ITM1236': ['46.34.C'],
'ITM6392': ['50.24.A'],
'ITM2105': ['39.11.A'],
'ITM6050': ['50.23.A'],
'ITM5210': ['50.17.B'],
'ITM5646': ['50.17.A'],
'ITM1120': ['46.37.B', '37.32.B'],
'ITM5738': ['50.15.A'],
'ITM1281': ['36.34.D', '35.39.A'],
'ITM5249': ['50.14.A'],
'ITM6139': ['50.10.A'],
'ITM383': ['49.24.B'],
'ITM5187': ['50.08.A'],
'ITM6013': ['50.06.A'],
'ITM6369': ['50.05.A'],
'ITM731': ['42.12.C'],
'ITM6048': ['50.04.A'],
'ITM6047': ['50.03.A'],
'ITM6083': ['50.02.A'],
'ITM5660': ['49.54.B'],
'ITM260': ['49.36.A', '47.29.C', '44.33.A', '43.07.D', '30.50.A'],
'ITM1540': ['29.13.A'],
'ITM6367': ['49.54.A'],
'ITM6137': ['49.53.B'],
'ITM163': ['50.18.B',
 '48.16.B',
 '47.21.C',
 '31.11.D',
 '31.10.B',
 '31.09.D',
 '31.07.D'],
'ITM5715': ['49.53.A'],
'ITM5851': ['49.52.B'],
'ITM1478': ['35.17.A'],
'ITM6245': ['49.52.A'],
'ITM2769': ['30.26.D'],
'ITM6372': ['49.51.A'],
'ITM1659': ['47.11.A', '40.10.A', '37.13.C'],
'ITM5099': ['49.50.A'],
'ITM6116': ['49.49.A'],
'ITM6356': ['49.48.A'],
'ITM5529': ['49.44.A'],
'ITM6134': ['49.42.A'],
'ITM6520': ['49.41.A'],
```

```
            'ITM2948': ['27.18.A'],
            'ITM5184': ['49.40.A'],
            'ITM6548': ['49.39.A'],
            'DUMMY_PART': ['49.35.A']})
```

In [365…

```
location = Locations['Location'].values.tolist()

same_aisle = [{50,49},{48,47},{46,45},{44,43},{42,41},{40,39},{38,37},{36,35},
              {34,33},{32,31},{30,29},{28,27},{26,25},{24,23},{22,21},{20,19},
              {18,17},{16,15},{14,13},{12,11},{10,9},{8,7},{6,5},{4,3},{2,1}]
distance_Pick = pd.Series(Locations['Final Pick Distance'].values,index=Locations.Location).to_dict()
distance_IO = pd.Series(Locations['Final distance to drop off'].values,index=Locations.Location).to_dict()
distance_BTW = defaultdict(dict)
for l1 in location:
    distance_BTW[l1] = defaultdict(dict)
    for l2 in location:
        a1 = int(l1[:2])
        a2 = int(l2[:2])
        if {a1,a2} in same_aisle:
            if a1 > a2:
                d1 = distance_Pick[l1]
                d2 = distance_Pick[l2] - 12
            else:
                d1 = distance_Pick[l1] - 12
                d2 = distance_Pick[l2]
            distance_BTW[l1][l2] = abs(d1-d2) + 12
        else:
            distance_BTW[l1][l2] = abs(distance_Pick[l1] - distance_Pick[l2])

#list(distance_BTW.items())[:1]
```

In [366…

```
stage = 'STG0'
STG_Assignment = STG0_assignment

pick_orders['Stage'] = [stage]*Total_picks
pick_orders['Location'] = ['']*Total_picks
pick_orders['Sort Field'] = [0]*Total_picks
for ind,rec in pick_orders.iterrows():
    p = rec['MATERIAL_NUMBER']
    l = STG_Assignment[p][0]
    sf = int(Locations[Locations['Location'] == l]['Sort Field'])
    pick_orders.loc[pick_orders['MATERIAL_NUMBER'] == p, 'Location'] = l
    pick_orders.loc[pick_orders['MATERIAL_NUMBER'] == p, 'Sort Field'] = sf

pick_orders = pick_orders.sort_values(by=['SALES_ORDER', 'Sort Field'], ascending=True)
```

```
pick_orders

Pick_time = pd.DataFrame(order_list, columns=['Sales Order'])
Pick_time['Stage'] = [stage] * Total_orders_toPick
Pick_time['Total Distance'] = [stage] * Total_orders_toPick
for order in order_list:
    total_dist = 0
    df = pick_orders[pick_orders['SALES_ORDER'] == order]
    first_time = True
    for ind, rec in df.iterrows():
        if first_time:
            first_time = False
            l = str(rec['Location'])
            total_dist += distance_IO[l]
            #print("first time: ",order, total_dist)
            l1 = l
            l2 = l
        else:
            l2 = str(rec['Location'])
            total_dist += abs(distance_BTW[l1][l2])
            #print("next  time: ",order, total_dist)
    total_dist += distance_IO[l2]
    #print("last  time: ",order, total_dist)
    Pick_time.loc[Pick_time['Sales Order'] == order, 'Total Distance'] = total_dist

print(Pick_time)
tot_pick_dist = Pick_time['Total Distance'].sum()
print(f"\nTotal pick distance for {stage} = {tot_pick_dist:,.2f}")
```

```
      Sales Order Stage Total Distance
0        27997929  STG0        252.417
1        28018206  STG0        252.417
2        28025370  STG0        105.083
3        28057994  STG0        105.083
4        28100693  STG0          205.5
...           ...   ...            ...
10190    55875321  STG0             36
10191    55875328  STG0         5750.5
10192    55875339  STG0        165.333
10193    55875348  STG0        1240.62
10194    55875362  STG0         1089.5

[10195 rows x 3 columns]

Total pick distance for STG0 = 10,975,520.63
```

In [367…

```
Pick_time.to_csv('STG0_order pick times.csv')
```

In [368…
```
df0 = pd.read_csv('STG0_order pick times.csv')
df1 = pd.read_csv('STG1_order pick times.csv')
df2 = pd.read_csv('STG2_order pick times.csv')
df3 = pd.read_csv('STG3_order pick times.csv')
```

In [369…
```
df = pd.concat([df0, df1, df2, df3])
df.rename(columns = {'Unnamed: 0':'Index'}, inplace = True)
df.rename(columns = {'Sales Order':'SALES_ORDER'}, inplace = True)
df.loc[df['Stage'] == 'STG0', 'Stage'] = 'AS-IS'
df.loc[df['Stage'] == 'STG1', 'Stage'] = 'Popularity Based'
df.loc[df['Stage'] == 'STG2', 'Stage'] = 'Affinity Based'
df.loc[df['Stage'] == 'STG3', 'Stage'] = 'Hybrid'
```

In [370…
```
df1 = df_salesOrder[df_salesOrder['MATERIAL_NUMBER'].isin(product)][['MATERIAL_NUMBER', 'SALES_ORDER']].drop_du
df2 = df1.groupby('SALES_ORDER')['MATERIAL_NUMBER'].agg(No_of_Lines='count').reset_index()
df2.loc[df2['No_of_Lines'] == 1, 'Single or Multiple'] = 'Single'
df2.loc[df2['No_of_Lines'] > 1, 'Single or Multiple'] = 'Multiple'
```

In [371…
```
df['SALES_ORDER'] = df['SALES_ORDER'].astype(str)
df2['SALES_ORDER'] = df2['SALES_ORDER'].astype(str)
Run_Results = pd.merge(df, df2, on="SALES_ORDER", how="left")
Run_Results.reset_index()
Run_Results
```

Out[371…

| | Index | SALES_ORDER | Stage | Total Distance | No_of_Lines | Single or Multiple |
|---|---|---|---|---|---|---|
| **0** | 0 | 27997929 | AS-IS | 252.416667 | 1 | Single |
| **1** | 1 | 28018206 | AS-IS | 252.416667 | 1 | Single |
| **2** | 2 | 28025370 | AS-IS | 105.083333 | 1 | Single |
| **3** | 3 | 28057994 | AS-IS | 105.083333 | 1 | Single |
| **4** | 4 | 28100693 | AS-IS | 205.500000 | 1 | Single |
| **...** | ... | ... | ... | ... | ... | ... |
| **40775** | 10190 | 55875321 | Hybrid | 205.500000 | 1 | Single |
| **40776** | 10191 | 55875328 | Hybrid | 1266.666667 | 11 | Multiple |
| **40777** | 10192 | 55875339 | Hybrid | 205.500000 | 2 | Multiple |
| **40778** | 10193 | 55875348 | Hybrid | 947.750000 | 4 | Multiple |

| | Index | SALES_ORDER | Stage | Total Distance | No_of_Lines | Single or Multiple |
|---|---|---|---|---|---|---|
| **40779** | 10194 | 55875362 | Hybrid | 411.375000 | 4 | Multiple |

40780 rows × 6 columns

In [372…
```
Run_Results.groupby('Stage').agg({'Total Distance':'sum', 'SALES_ORDER': 'count'})
```

Out[372…

| Stage | Total Distance | SALES_ORDER |
|---|---|---|
| **AS-IS** | 1.097552e+07 | 10195 |
| **Affinity Based** | 3.404703e+06 | 10195 |
| **Hybrid** | 3.366152e+06 | 10195 |
| **Popularity Based** | 3.924439e+06 | 10195 |

In [373…
```
Run_Results.groupby(['Stage','Single or Multiple']).agg({'Total Distance':'sum', 'SALES_ORDER': 'count'})
```

Out[373…

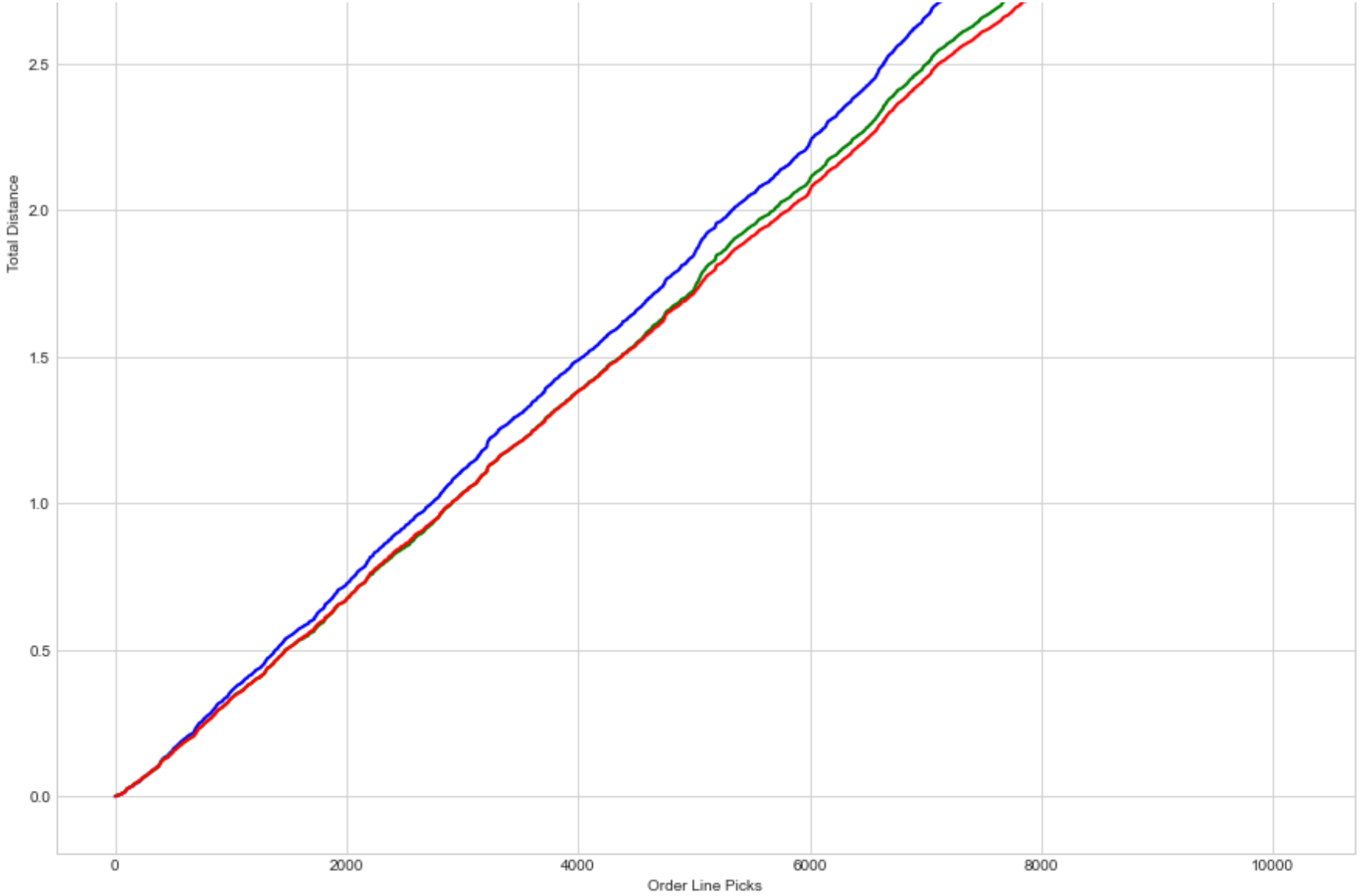
| Stage | Single or Multiple | Total Distance | SALES_ORDER |
|---|---|---|---|
| **AS-IS** | **Multiple** | 8.938918e+06 | 4101 |
| | **Single** | 2.036602e+06 | 6094 |
| **Affinity Based** | **Multiple** | 2.570237e+06 | 4101 |
| | **Single** | 8.344663e+05 | 6094 |
| **Hybrid** | **Multiple** | 2.520086e+06 | 4101 |
| | **Single** | 8.460664e+05 | 6094 |
| **Popularity Based** | **Multiple** | 3.106628e+06 | 4101 |
| | **Single** | 8.178111e+05 | 6094 |

In [374…
```
# Python code to get the Cumulative sum of a list
def Cumulative(lists):
    cu_list = []
    length = len(lists)
```

```
    cu_list = [sum(lists[0:x:1]) for x in range(0, length+1)]
    return cu_list[1:]
```

In [383…

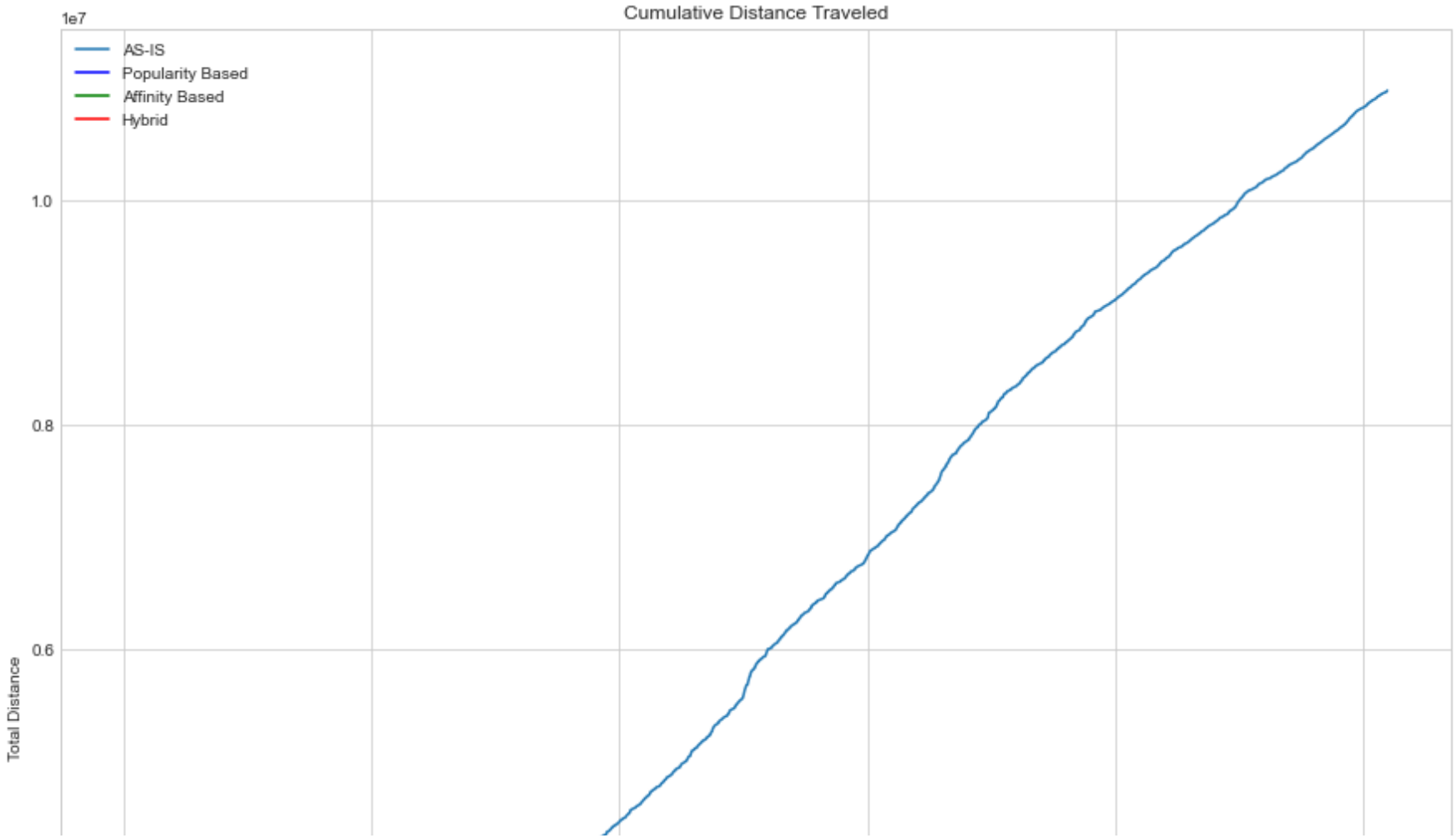
```
x = Run_Results.loc[Run_Results['Stage'] == 'AS-IS']['Index']
y1 = Run_Results.loc[Run_Results['Stage'] == 'AS-IS']['Total Distance']
y1 = Cumulative(y1)

y2 = Run_Results.loc[Run_Results['Stage'] == 'Popularity Based']['Total Distance']
y2 = Cumulative(y2)

y3 = Run_Results.loc[Run_Results['Stage'] == 'Affinity Based']['Total Distance']
y3 = Cumulative(y3)

y4 = Run_Results.loc[Run_Results['Stage'] == 'Hybrid']['Total Distance']
y4 = Cumulative(y4)

plt.rcParams["figure.figsize"] = (15,15)

#plt.plot(x,y1)
plt.plot(x,y2, color = "blue", linewidth=2)
plt.plot(x,y3, color = "green", linewidth=2)
plt.plot(x,y4, color = "red", linewidth=2)
plt.title('Cumulative Distance Traveled')
plt.xlabel('Order Line Picks')
plt.ylabel('Total Distance')
plt.gca().legend(('Popularity Based','Affinity Based','Hybrid'))
plt.show()
```

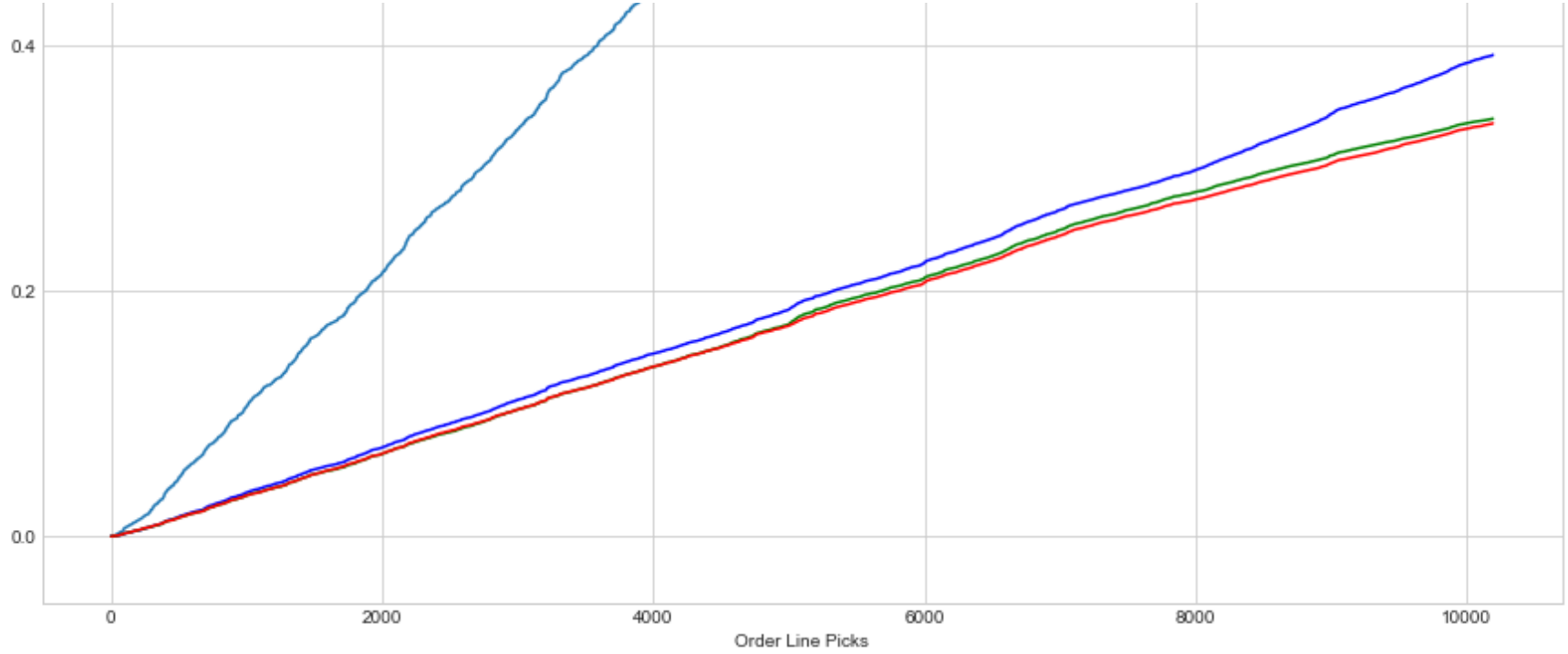



In [384…

```
x = Run_Results.loc[Run_Results['Stage'] == 'AS-IS']['Index']
y1 = Run_Results.loc[Run_Results['Stage'] == 'AS-IS']['Total Distance']
y1 = Cumulative(y1)

y2 = Run_Results.loc[Run_Results['Stage'] == 'Popularity Based']['Total Distance']
y2 = Cumulative(y2)

y3 = Run_Results.loc[Run_Results['Stage'] == 'Affinity Based']['Total Distance']
```

```
y3 = Cumulative(y3)

y4 = Run_Results.loc[Run_Results['Stage'] == 'Hybrid']['Total Distance']
y4 = Cumulative(y4)

plt.plot(x,y1)
plt.plot(x,y2, color = "blue")
plt.plot(x,y3, color = "green")
plt.plot(x,y4, color = "red")
plt.title('Cumulative Distance Traveled')
plt.xlabel('Order Line Picks')
plt.ylabel('Total Distance')
plt.gca().legend(('AS-IS','Popularity Based','Affinity Based','Hybrid'))
plt.show()
```






In [ ]: