Production Planning - Material Yield Prediction System

SAP Manufacturing Logic

- **101** = INPUT material consumption (raw materials, BFIN)
- **261** = OUTPUT material production (finished goods, BFOUT derived from dimensions)
- Input and Output materials are **DIFFERENT**
- Join **ONLY** on `MANUFACTURINGORDER`
- `Yield = Total_Output_BF / Total_Input_BF`

Real-World Use Cases

1. **Forward Planning**: "If I consume X BF of raw material, how much output will I get?"
2. **Reverse Planning**: "If I need Y BF of finished goods, how much raw material do I need?"
3. **Material Selection**: "Which raw material gives the best yield for my needs?"
4. **Anomaly Detection**: "Is this manufacturing order producing abnormal loss?"

1. Setup and Imports

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display, HTML

# Plot settings
plt.style.use('seaborn-v0_8-whitegrid')
pd.set_option('display.max_columns', None)
pd.set_option('display.float_format', '{:.2f}'.format)

print("Imports successful!")
```

```
Imports successful!
```

2. Load and Clean Data

```
# Load raw data
df_101 = pd.read_csv('/content/2025_101Data.csv')
df_261 = pd.read_csv('/content/2025_261Data.csv')

print(f"101.csv (Inputs): {df_101.shape[0]:,} rows, {df_101.shape[1]} columns")
print(f"261.csv (Outputs): {df_261.shape[0]:,} rows, {df_261.shape[1]} columns")
```

```
101.csv (Inputs): 1,815,471 rows, 15 columns
261.csv (Outputs): 2,013,848 rows, 15 columns
```

```
Start coding or generate with AI.
```

Task

Filter `df_101` for `PLANT` '1Y01', `MATERIALSPECIE` 'POP', `MATERIALTHICKNESS` 4, and `MATERIALGRADE` '1C', then calculate the number of rows and the sum of the `BFIN` column for the filtered data and display the results.

Filter df_101

Subtask:

Filter df_101 for PLANT '1Y01', MATERIALSPECIE 'POP', MATERIALTHICKNESS 4, and MATERIALGRADE '1C'.

Reasoning: To filter the `df_101` DataFrame based on the specified conditions, I will use boolean indexing and store the result in `df_101_filtered`.

```
df_101_filtered = df_101[
    (df_101['PLANT'] == '1Y01') &
    (df_101['MATERIALSPECIE'] == 'POP') &
    (df_101['MATERIALTHICKNESS'] == 4)

]

print(f"Filtered df_101: {df_101_filtered.shape[0]:,} rows, {df_101_filtered.shape[1]} columns")
display(df_101_filtered.head())
```

```
Filtered df_101: 17,841 rows, 15 columns
```

	MANUFACTURINGORDER	PLANT	MATERIAL	MATERIALTHICKNESS	MATERIALSPECIE	TALLYLENGTH	TALLYGRADE	GOODSMOVEMENTTYPE	TALLYWI
251	1090897	1Y01	4POCGKD	4	POP	72.00	1C	101	
252	1090897	1Y01	4POCGKD	4	POP	156.00	1C	101	1
253	1090897	1Y01	4POCGKD	4	POP	156.00	1C	101	1
254	1090897	1Y01	4POCGKD	4	POP	156.00	1C	101	
255	1090897	1Y01	4POCGKD	4	POP	192.00	1C	101	

```
total_bfout = df_101_filtered['BFOUT'].sum()

print(f"Total BFOUT: {total_bfout:,}")
```

```
Total BFOUT: 1,243,652
```

```
# Group and sum
bfout_per_order = (
    df_101_filtered
    .groupby('MANUFACTURINGORDER', as_index=False)['BFOUT']
    .sum()
    .sort_values('BFOUT', ascending=False)
)

# Display result
display(bfout_per_order)
```

	MANUFACTURINGORDER	BFOUT
12	1098326	213182
9	1094958	176510
0	1086218	115478
4	1090897	97225
1	1087027	93089
10	1096084	91638
6	1092935	90749
11	1097259	90726
5	1092387	89224
8	1094006	79121
2	1089302	74400
3	1090215	31483
7	1093813	827

Next steps: Generate code with bfout_per_order | New interactive sheet

```
df_261_filtered = df_261[
    (df_261['PLANT'] == '1Y01') &
    (df_261['MATERIALSPECIE'] == 'POP') &
    (df_261['MATERIALTHICKNESS'] == 4)
]

print(f"Filtered df_261: {df_261_filtered.shape[0]:,} rows, {df_261_filtered.shape[1]} columns")
display(df_261_filtered.head())
```

```
Filtered df_261: 36,354 rows, 15 columns
```

	MANUFACTURINGORDER	PLANT	MATERIAL	MATERIALTHICKNESS	MATERIALSPECIE	TALLYLENGTH	TALLYGRADE	GOODSMOVEMENTTYPE	TALLYWI
253	1090897	1Y01	4PO3BKS	4	POP	144.00	1C	261	8
254	1090897	1Y01	4PO3BKS	4	POP	144.00	1C	261	4
255	1090897	1Y01	4PO3BKS	4	POP	168.00	1C	261	15
256	1090897	1Y01	4PO3BKS	4	POP	168.00	1C	261	5
257	1090897	1Y01	4PO3BKS	4	POP	168.00	1C	261	4

```
total_bfin = df_261_filtered['BFIN'].sum()

print(f"Total BFIN: {total_bfin:,}")

total_bfout = df_101_filtered['BFOUT'].sum()

print(f"Total BFOUT: {total_bfout:,}")
```

```
Total BFIN: 1,375,911.0
Total BFOUT: 1,243,652
```

```
# Group and sum
bfin_per_order = (
    df_261_filtered
    .groupby('MANUFACTURINGORDER', as_index=False)['BFIN']
    .sum()
```

```
    .sort_values('BFIN', ascending=False)
)

# Display result
display(bfin_per_order)
```

	MANUFACTURINGORDER	BFIN
11	1098326	237908.00
8	1094958	193332.00
0	1086218	128104.00
4	1090897	107553.00
9	1096084	102268.00
1	1087027	101958.00
10	1097259	101024.00
5	1092387	100577.00
6	1092935	99641.00
7	1094006	87046.00
2	1089302	81695.00
3	1090215	34805.00

Next steps: Generate code with `bfin_per_order` New interactive sheet

```
common_orders_df = (
    df_101_filtered[['MANUFACTURINGORDER']]
    .drop_duplicates()
    .merge(
        df_261_filtered[['MANUFACTURINGORDER']].drop_duplicates(),
        on='MANUFACTURINGORDER',
        how='inner'
    )
)

print(f"Common Manufacturing Orders: {common_orders_df.shape[0]}")
display(common_orders_df.head())
```

```
Common Manufacturing Orders: 12
```

	MANUFACTURINGORDER
0	1090897
1	1096084
2	1087027
3	1094958
4	1092935

```
materials_per_order = (
    df_101_filtered[
        df_101_filtered['MANUFACTURINGORDER']
        .isin(common_orders_df['MANUFACTURINGORDER'])
    ]
    .groupby('MANUFACTURINGORDER')['MATERIAL']
    .apply(list)   # use set() if you want unique only
    .reset_index(name='Materials')
)

display(materials_per_order.head())
```

MANUFACTURINGORDER Materials

```
kd_materials_per_order = (
    df_101_filtered
    .groupby('MANUFACTURINGORDER')['MATERIAL']
    .apply(lambda x: list(dict.fromkeys(x)))  # UNIQUE, order preserved
    .reset_index(name='KD_MATERIALS')
)

final_materials_table = (
    ks_material_per_order
    .merge(kd_materials_per_order, on='MANUFACTURINGORDER', how='left')
)

final_materials_table['Materials'] = final_materials_table.apply(
    lambda row: [row['KS_MATERIAL']] + (row['KD_MATERIALS'] or []),
    axis=1
)

final_materials_table = final_materials_table[
    ['MANUFACTURINGORDER', 'Materials']
]

display(final_materials_table.head())
```

1 to 5 of 5 entries Filter

index	MANUFACTURINGORDER	Materials
0	1086218	4PO3BKS,4POCGKD,4PO2CKD,4POPRKD,4POPRKD12W,4POPRST89KD,4POSELKD6
1	1087027	4PO3BKS,4POCGKD,4PO2CKD,4POPRKD,4POPRST89KD,4POPRKD12W,4POSELKD7,4POSELKD6
2	1089302	4PO3BKS,4POCGKD,4PO2CKD,4POPRKD12W,4POPRST89KD,4POPRKD,4POSELKD7,4POSELKD6
3	1090215	4PO3BKS,4POCGKD,4PO2CKD,4POPRKD12W,4POPRKD,4POSELKD6,4POPRST89KD
4	1090897	4PO3BKS,4POCGKD,4PO2CKD,4POPRKD12W,4POPRST8KD,4POPRST6KD,4POPRST89KD,4POPRKD,4POSELKD7,4POS

Show 100 per page
Like what you see? Visit the data table notebook to learn more about interactive tables.

```
order_id = 1086218   # change this if needed
```

```
ks_row = (
    df_261_filtered[df_261_filtered['MANUFACTURINGORDER'] == order_id]
    .iloc[0]
)

ks_material = ks_row['MATERIAL']

print("KS Material:", ks_material)
```

```
KS Material: 4PO3BKS
```

```
ks_bfin = (
    df_261_filtered[
        (df_261_filtered['MANUFACTURINGORDER'] == order_id) &
        (df_261_filtered['MATERIAL'] == ks_material)
    ]['BFIN']
    .sum()
)

print("KS BFIN:", ks_bfin)
```

```
KS BFIN: 128104.0
```

```
total_bfout = (
    df_101_filtered[
        df_101_filtered['MANUFACTURINGORDER'] == order_id
    ]['BFOUT']
    .sum()
)

print("Total BFOUT:", total_bfout)
```

```
Total BFOUT: 115478
```

```
yield_percent = round((ks_bfin / total_bfout) * 100, 2) if total_bfout > 0 else None

print("Yield %:", yield_percent)
```

```
Yield %: 110.93
```

```
summary_df = pd.DataFrame([{
    'MANUFACTURINGORDER': order_id,
    'KS_MATERIAL': ks_material,
    'KS_BFIN': ks_bfin,
    'TOTAL_BFOUT': total_bfout,
    'YIELD_PERCENTAGE': yield_percent
}])

display(summary_df)
```

	MANUFACTURINGORDER	KS_MATERIAL	KS_BFIN	TOTAL_BFOUT	YIELD_PERCENTAGE
0	1086218	4PO3BKS	128104.00	115478	110.93

```
order_101 = df_101_filtered[
    df_101_filtered['MANUFACTURINGORDER'] == order_id
]
```

```
material_input = (
    order_101
    .groupby('MATERIAL')['BFOUT']
    .sum()
    .reset_index(name='TOTAL_BFOUT')
)
```

```
total_input = material_input['TOTAL_BFOUT'].sum()

material_input['INPUT_PERCENTAGE'] = (
    material_input['TOTAL_BFOUT'] / total_input * 100
).round(2)

display(material_input)
```

	MATERIAL	TOTAL_BFOUT	INPUT_PERCENTAGE
0	4PO2CKD	7865	6.81
1	4POCGKD	18316	15.86
2	4POPRKD	77635	67.23
3	4POPRKD12W	7838	6.79
4	4POPRST89KD	2368	2.05
5	4POSELKD6	1456	1.26

Next steps: Generate code with `material_input` New interactive sheet