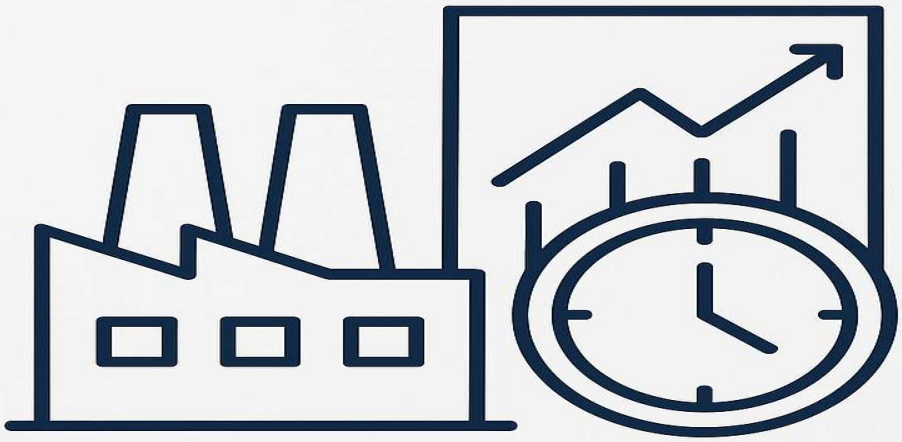
Manufacturing Downtime Analysis



Documentations For Manufacturing Downtime Project

Peter Mohareb

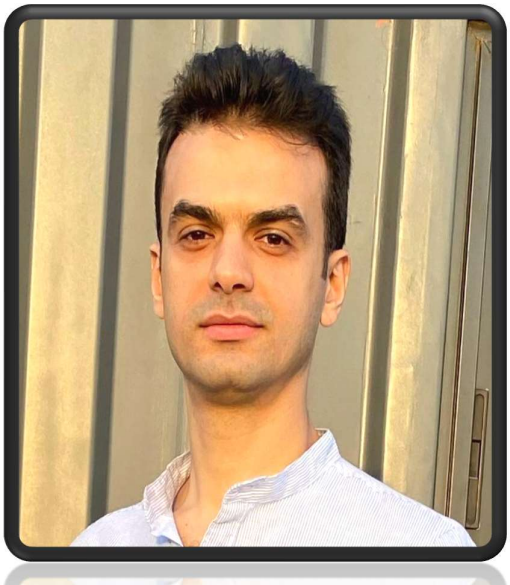
Data Analytics - Data Analyst Specialist

Ministry of Communications and Information Technology

Digital Egypt Pioneers

Team Members:-

Mohamed Gamal



Mohamed Adel

Peter Mohareb

Ministry of Communications and Information Technology

Digital Egypt Pioneers

Supervised By:

Eng/ Ahmed Samir.

Contents

Phase I

Project Overview

. Project Overview.

Objective

The goal of this project is to analyze and forecast manufacturing downtime to identify patterns, reduce unexpected failures, and optimize production efficiency.

Problem Statement

Manufacturing downtime leads to productivity losses and increased operational costs. By leveraging historical data, we aim to predict downtime events and provide actionable insights for process improvement.

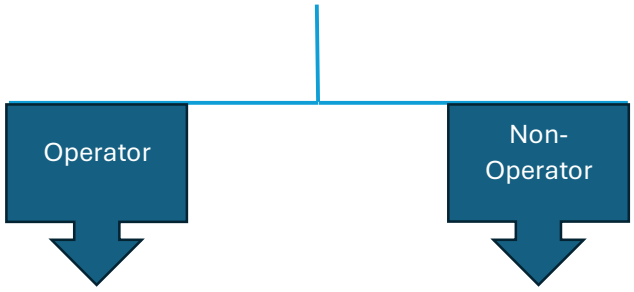
Data Loading

Data Loading.

Raw Data Consits of 4 Sheets

1- **Line productivity.**
2- **Products.**
3- **Downtime factors.**
4- **Line downtime.**

No. Of Batches = 38
No. Of Products = 6
No. Of Factors = 12 Factors Divided to two categories



Dataset Description

- **Source:** (CSV File)
- **Time Period:** (1 Week)
- **Variables:**
  - **Product**
  - **Batch**
  - **Operator**
  - **Flavor**
  - **Size**
  - **Min batch time**
  - **DownTime Factors**

Step 1: Load the Data

Set display options to show all rows & columns

Line productivity Sheet

Date	Product	Batch	Operator	Start Time	End Time
2024-08-29	OR-600	422111	Mac	11:50:00	14:05:00
2024-08-29	LE-600	422112	Mac	14:05:00	15:45:00
2024-08-29	LE-600	422113	Mac	15:45:00	17:35:00
2024-08-29	LE-600	422114	Mac	17:35:00	19:15:00
2024-08-29	LE-600	422115	Charlie	19:15:00	20:39:00
2024-08-29	LE-600	422116	Charlie	20:39:00	21:39:00
2024-08-29	LE-600	422117	Charlie	21:39:00	22:54:00
2024-08-30	CO-600	422118	Dee	04:05:00	06:05:00
2024-08-30	CO-600	422119	Dee	06:05:00	07:30:00
2024-08-30	CO-600	422120	Dee	07:30:00	09:22:00
2024-08-30	CO-600	422121	Dennis	09:22:00	10:37:00
2024-08-30	CO-600	422122	Dennis	10:37:00	12:02:00
2024-08-30	CO-600	422123	Dennis	12:02:00	14:15:00
2024-08-30	CO-600	422124	Dennis	14:15:00	15:55:00
2024-08-30	CO-600	422125	Charlie	15:55:00	17:15:00
2024-08-30	CO-600	422126	Charlie	17:15:00	18:59:00
2024-08-30	CO-600	422127	Charlie	18:59:00	20:22:00
2024-08-30	CO-600	422128	Charlie	20:22:00	22:14:00
2024-08-30	CO-600	422129	Charlie	22:14:00	23:29:00
2024-08-31	CO-600	422130	Dee	07:45:00	09:05:00
2024-08-31	CO-600	422131	Dee	09:05:00	10:35:00
2024-08-31	CO-600	422132	Dee	10:35:00	11:35:00
2024-08-31	DC-600	422133	Dee	11:35:00	12:55:00
2024-08-31	DC-600	422134	Mac	12:55:00	14:45:00
2024-08-31	DC-600	422135	Mac	14:45:00	16:30:00
2024-08-31	DC-600	422136	Mac	16:30:00	17:30:00
2024-09-02	RB-600	422137	Dee	01:00:00	02:45:00
2024-09-02	RB-600	422138	Dee	02:45:00	04:05:00
2024-09-02	RB-600	422139	Dee	04:05:00	05:40:00
2024-09-02	RB-600	422140	Dee	05:40:00	07:43:00
2024-09-02	RB-600	422141	Dennis	07:43:00	08:50:00
2024-09-02	RB-600	422142	Dennis	08:50:00	10:20:00
2024-09-02	RB-600	422143	Dennis	10:20:00	12:18:00
2024-09-02	CO-2L	422144	Dennis	12:18:00	14:50:00
2024-09-02	CO-2L	422145	Charlie	14:50:00	16:50:00
2024-09-02	CO-2L	422146	Charlie	16:50:00	19:30:00
2024-09-02	CO-2L	422147	Charlie	19:30:00	22:55:00
2024-09-03	CO-2L	422148	Mac	00:05:00	02:15:00

Products Sheet

Product	Flavor	Size	Min batch time
OR-600	Orange	600 ml	60
LE-600	Lemon lime	600 ml	60
CO-600	Cola	600 ml	60
DC-600	Diet Cola	600 ml	60
RB-600	Root Berry	600 ml	60
CO-2L	Cola	2 L	98

Downtime factors

Factor	Description	Operator Error
1	Emergency stop	No
2	Batch change	Yes
3	Labeling error	No
4	Inventory shortage	No
5	Product spill	Yes
6	Machine adjustment	Yes
7	Machine failure	No
8	Batch coding error	Yes
9	Conveyor belt jam	No
10	Calibration error	Yes
11	Label switch	Yes
12	Other	No

Line downtime

	Downtime factor										
Batch	1	2	3	4	5	6	7	8	9	10	11	12
422111		60					15					
422112		20						20				
422113		50										
422114				25		15						
422115										24		
422116												
422117		10				5						
422118						14	16				10	20
422119				25								
422120				20	15				17			
422121							15					
422122							25					
422123				43			30					
422124					20	20						
422125											10	10
422126								44				
422127						23						
422128					22		30					
422129												15
422130		20										
422131				20						10		
422132												
422133							20					
422134							30	20				
422135				30								15
422136												
422137								30		15		
422138			20									
422139				20		15						
422140						50					13	
422141												7
422142						30						
422143						40	18					
422144						30		24				
422145			22									
422146						30	25					7
422147				17		60	30					
422148				25				7				

SQL:

```
create database Manufacturing_Downtime

create table Line_productivity
        (Date date not null,
        Product varchar(50) not null,
        Batch int not null,
        Operator varchar(50) not null,
        Start_Time time not null,
        End_Time time not null)

create table Products
        (Product varchar(50) not null,
         Flavor varchar(50) not null,
         Size varchar(50) not null,
         Min_batch_time int not null)

create table Line_downtime_factor1
        (Batch int not null,
         Factor int not null,
         Batch_downtime_by_factor int)

create table Downtime_factors
       (Factor int not null,
        Description varchar(100) not null,
        Operator_Error varchar(50) not null)
```

```
create table Line_downtime_factor
        (Batch int not null,
         one int,
         two int,
         three int,
         four int,
         five int,
         six int,
         seven int,
         eight int,
         nine int,
         ten int,
         eleven int,
         twelve int)
```

Bulk Insert

```
USE Manufacturing_Downtime;
GO
BULK INSERT Line_productivity
FROM 'D:\Data Analyst\Project DEPI\SQL\Line productivity.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
);

BULK INSERT Products
FROM 'D:\Data Analyst\Project DEPI\SQL\Products.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
);

BULK INSERT Downtime_factors
FROM 'D:\Data Analyst\Project DEPI\SQL\Downtime factors.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
);

BULK INSERT Line_downtime_factor
FROM 'D:\Data Analyst\Project DEPI\SQL\Line downtime.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
);

BULK INSERT Line_downtime_factor1
FROM 'D:\Data Analyst\Project DEPI\SQL\Line downtime 1.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
);
```

Python:

```
#Set Up Your Environment
#Make sure you have the required libraries installed:
#!pip install pandas matplotlib openpyxl statsmodels scikit-learn
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from openpyxl import load_workbook
from sklearn.linear_model import LinearRegression
```

```
#Phase 1: Build Data Model, Data Cleaning and Preprocessing
```

```
#Step 1: Load the Data

file_path = r"F:\Rawad Misr\Data Analysis\Projects\Final Projects\Final Project\Manufacturing_Line_Productivity.xlsx"
sheets_dict = pd.read_excel(file_path, sheet_name=None)  # Read all sheets

# Check the keys (sheet names)
print(sheets_dict.keys())

Line_downtime = sheets_dict["Line downtime"]
df_productivity = sheets_dict["Line productivity"]
df_Products = sheets_dict["Products"]
Downtime_factors = sheets_dict["Downtime factors"]
```

```
dict_keys(['Line productivity', 'Products', 'Downtime factors', 'Line downtime'])
```

```
# Set display options to show all rows & columns
pd.set_option("display.max_rows", None)  # Show all rows
pd.set_option("display.max_columns", None)  # Show all columns
pd.set_option("display.expand_frame_repr", False)  # Prevent column wrapping

# Loop through each sheet and display its full content
for sheet_name, df in sheets_dict.items():
    print(f"\n🔹 Sheet: {sheet_name}\n")

    # Display DataFrame info before full content
    print("\n🔍 Data Info:")
    df.info()

    print("=" * 100)  # Separator for readability
    # Display full DataFrame
    print(df)
```

🔹 Sheet: Line productivity

🔍 Data Info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38 entries, 0 to 37
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Date        38 non-null     datetime64[ns]
 1   Product     38 non-null     object
 2   Batch       38 non-null     int64
 3   Operator    38 non-null     object
 4   Start Time  38 non-null     object
 5   End Time    38 non-null     object
dtypes: datetime64[ns](1), int64(1), object(4)
memory usage: 1.9+ KB
```

	Date	Product	Batch	Operator	Start Time	End Time
0	2024-08-29	OR-600	422111	Mac	11:50:00	14:05:00
1	2024-08-29	LE-600	422112	Mac	14:05:00	15:45:00
2	2024-08-29	LE-600	422113	Mac	15:45:00	17:35:00
3	2024-08-29	LE-600	422114	Mac	17:35:00	19:15:00
4	2024-08-29	LE-600	422115	Charlie	19:15:00	20:39:00
5	2024-08-29	LE-600	422116	Charlie	20:39:00	21:39:00
6	2024-08-29	LE-600	422117	Charlie	21:39:00	22:54:00
7	2024-08-30	CO-600	422118	Dee	04:05:00	06:05:00
8	2024-08-30	CO-600	422119	Dee	06:05:00	07:30:00
9	2024-08-30	CO-600	422120	Dee	07:30:00	09:22:00
10	2024-08-30	CO-600	422121	Dennis	09:22:00	10:37:00
11	2024-08-30	CO-600	422122	Dennis	10:37:00	12:02:00
12	2024-08-30	CO-600	422123	Dennis	12:02:00	14:15:00
13	2024-08-30	CO-600	422124	Dennis	14:15:00	15:55:00
14	2024-08-30	CO-600	422125	Charlie	15:55:00	17:15:00
15	2024-08-30	CO-600	422126	Charlie	17:15:00	18:59:00
16	2024-08-30	CO-600	422127	Charlie	18:59:00	20:22:00
17	2024-08-30	CO-600	422128	Charlie	20:22:00	22:14:00
18	2024-08-30	CO-600	422129	Charlie	22:14:00	23:29:00
19	2024-08-31	CO-600	422130	Dee	07:45:00	09:05:00
20	2024-08-31	CO-600	422131	Dee	09:05:00	10:35:00
21	2024-08-31	CO-600	422132	Dee	10:35:00	11:35:00
22	2024-08-31	DC-600	422133	Dee	11:35:00	12:55:00
23	2024-08-31	DC-600	422134	Mac	12:55:00	14:45:00
24	2024-08-31	DC-600	422135	Mac	14:45:00	16:30:00
25	2024-08-31	DC-600	422136	Mac	16:30:00	17:30:00
26	2024-09-02	RB-600	422137	Dee	01:00:00	02:45:00
27	2024-09-02	RB-600	422138	Dee	02:45:00	04:05:00
28	2024-09-02	RB-600	422139	Dee	04:05:00	05:40:00
29	2024-09-02	RB-600	422140	Dee	05:40:00	07:43:00
30	2024-09-02	RB-600	422141	Dennis	07:43:00	08:50:00
31	2024-09-02	RB-600	422142	Dennis	08:50:00	10:20:00
32	2024-09-02	RB-600	422143	Dennis	10:20:00	12:18:00
33	2024-09-02	CO-2L	422144	Dennis	12:18:00	14:50:00
34	2024-09-02	CO-2L	422145	Charlie	14:50:00	16:50:00
35	2024-09-02	CO-2L	422146	Charlie	16:50:00	19:30:00
36	2024-09-02	CO-2L	422147	Charlie	19:30:00	22:55:00
37	2024-09-03	CO-2L	422148	Mac	00:05:00	02:15:00

```
🔹 Sheet: Products

🔍 Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Product          6 non-null      object
 1   Flavor           6 non-null      object
 2   Size             6 non-null      object
 3   Min batch time   6 non-null      int64
dtypes: int64(1), object(3)
memory usage: 324.0+ bytes
==================================================
  Product       Flavor    Size  Min batch time
0  OR-600       Orange  600 ml              60
1  LE-600  Lemon lime  600 ml              60
2  CO-600         Cola  600 ml              60
3  DC-600    Diet Cola  600 ml              60
4  RB-600  Root Berry  600 ml              60
5   CO-2L         Cola     2 L              98
```

```
🔹 Sheet: Downtime factors

🔍 Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Factor           12 non-null     int64
 1   Description      12 non-null     object
 2   Operator Error   12 non-null     object
dtypes: int64(1), object(2)
memory usage: 420.0+ bytes
==================================================
    Factor          Description Operator Error
0        1       Emergency stop             No
1        2         Batch change            Yes
2        3       Labeling error             No
3        4  Inventory shortage              No
4        5        Product spill            Yes
5        6  Machine adjustment             Yes
6        7     Machine failure              No
7        8  Batch coding error             Yes
8        9   Conveyor belt jam              No
9       10   Calibration error             Yes
10      11         Label switch            Yes
11      12                Other             No
```

```
🔹 Sheet: Line downtime


🔍 Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39 entries, 0 to 38
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Unnamed: 0       39 non-null     object
 1   Downtime factor  1 non-null      float64
 2   Unnamed: 2       6 non-null      float64
 3   Unnamed: 3       3 non-null      float64
 4   Unnamed: 4       10 non-null     float64
 5   Unnamed: 5       4 non-null      float64
 6   Unnamed: 6       13 non-null     float64
 7   Unnamed: 7       12 non-null     float64
 8   Unnamed: 8       7 non-null      float64
 9   Unnamed: 9       2 non-null      float64
 10  Unnamed: 10      4 non-null      float64
 11  Unnamed: 11      4 non-null      float64
 12  Unnamed: 12      7 non-null      float64
dtypes: float64(12), object(1)
memory usage: 4.1+ KB
================================================================================================
   Unnamed: 0  Downtime factor  Unnamed: 2  Unnamed: 3  Unnamed: 4  Unnamed: 5  Unnamed: 6  Unnamed: 7  Unnamed: 8  Unnamed: 9  Unnamed: 10  Unname
d: 11  Unnamed: 12
0       Batch             1.0         2.0         3.0         4.0         5.0         6.0         7.0         8.0         9.0         10.0
11.0         12.0
1      422111             NaN        60.0         NaN         NaN         NaN         NaN        15.0         NaN         NaN          NaN
NaN          NaN
2      422112             NaN        20.0         NaN         NaN         NaN         NaN         NaN        20.0         NaN          NaN
NaN          NaN
3      422113             NaN        50.0         NaN         NaN         NaN         NaN         NaN         NaN         NaN          NaN
NaN          NaN
4      422114             NaN         NaN         NaN        25.0         NaN        15.0         NaN         NaN         NaN          NaN
NaN          NaN
5      422115             NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN         24.0
NaN          NaN
```

											NaN
6	422116	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN										
7	422117	NaN	10.0	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN
NaN	NaN										
8	422118	NaN	NaN	NaN	NaN	NaN	14.0	16.0	NaN	NaN	NaN
10.0	20.0										
9	422119	NaN	NaN	NaN	25.0	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN										
10	422120	NaN	NaN	NaN	20.0	15.0	NaN	NaN	NaN	17.0	NaN
NaN	NaN										
11	422121	NaN	NaN	NaN	NaN	NaN	NaN	15.0	NaN	NaN	NaN
NaN	NaN										
12	422122	NaN	NaN	NaN	NaN	NaN	NaN	25.0	NaN	NaN	NaN
NaN	NaN										
13	422123	NaN	NaN	NaN	43.0	NaN	NaN	30.0	NaN	NaN	NaN
NaN	NaN										
14	422124	NaN	NaN	NaN	NaN	20.0	20.0	NaN	NaN	NaN	NaN
NaN	NaN										
15	422125	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10.0	10.0										
16	422126	NaN	NaN	NaN	NaN	NaN	NaN	NaN	44.0	NaN	NaN
NaN	NaN										
17	422127	NaN	NaN	NaN	NaN	NaN	23.0	NaN	NaN	NaN	NaN
NaN	NaN										
18	422128	NaN	NaN	NaN	NaN	22.0	NaN	30.0	NaN	NaN	NaN
NaN	NaN										
19	422129	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	15.0										
20	422130	NaN	20.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN										
21	422131	NaN	NaN	NaN	20.0	NaN	NaN	NaN	NaN	NaN	10.0
NaN	NaN										
22	422132	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN										
23	422133	NaN	NaN	NaN	NaN	NaN	NaN	20.0	NaN	NaN	NaN
NaN	NaN										

24	422134	NaN	NaN	NaN	NaN	NaN	NaN	30.0	20.0	NaN	NaN
NaN	NaN										
25	422135	NaN	NaN	NaN	30.0	NaN	NaN	NaN	NaN	NaN	NaN
NaN	15.0										
26	422136	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN										
27	422137	NaN	NaN	NaN	NaN	NaN	NaN	NaN	30.0	NaN	15.0
NaN	NaN										
28	422138	NaN	NaN	20.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN										
29	422139	NaN	NaN	NaN	20.0	NaN	15.0	NaN	NaN	NaN	NaN
NaN	NaN										
30	422140	NaN	NaN	NaN	NaN	NaN	50.0	NaN	NaN	NaN	NaN
13.0	NaN										
31	422141	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	7.0										
32	422142	NaN	NaN	NaN	NaN	NaN	30.0	NaN	NaN	NaN	NaN
NaN	NaN										
33	422143	NaN	NaN	NaN	NaN	NaN	40.0	18.0	NaN	NaN	NaN
NaN	NaN										
34	422144	NaN	NaN	NaN	NaN	NaN	30.0	NaN	24.0	NaN	NaN
NaN	NaN										
35	422145	NaN	NaN	22.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN										
36	422146	NaN	NaN	NaN	NaN	NaN	30.0	25.0	NaN	NaN	NaN
NaN	7.0										
37	422147	NaN	NaN	NaN	17.0	NaN	60.0	30.0	NaN	NaN	NaN
NaN	NaN										
38	422148	NaN	NaN	NaN	25.0	NaN	NaN	NaN	7.0	NaN	NaN
NaN	NaN										

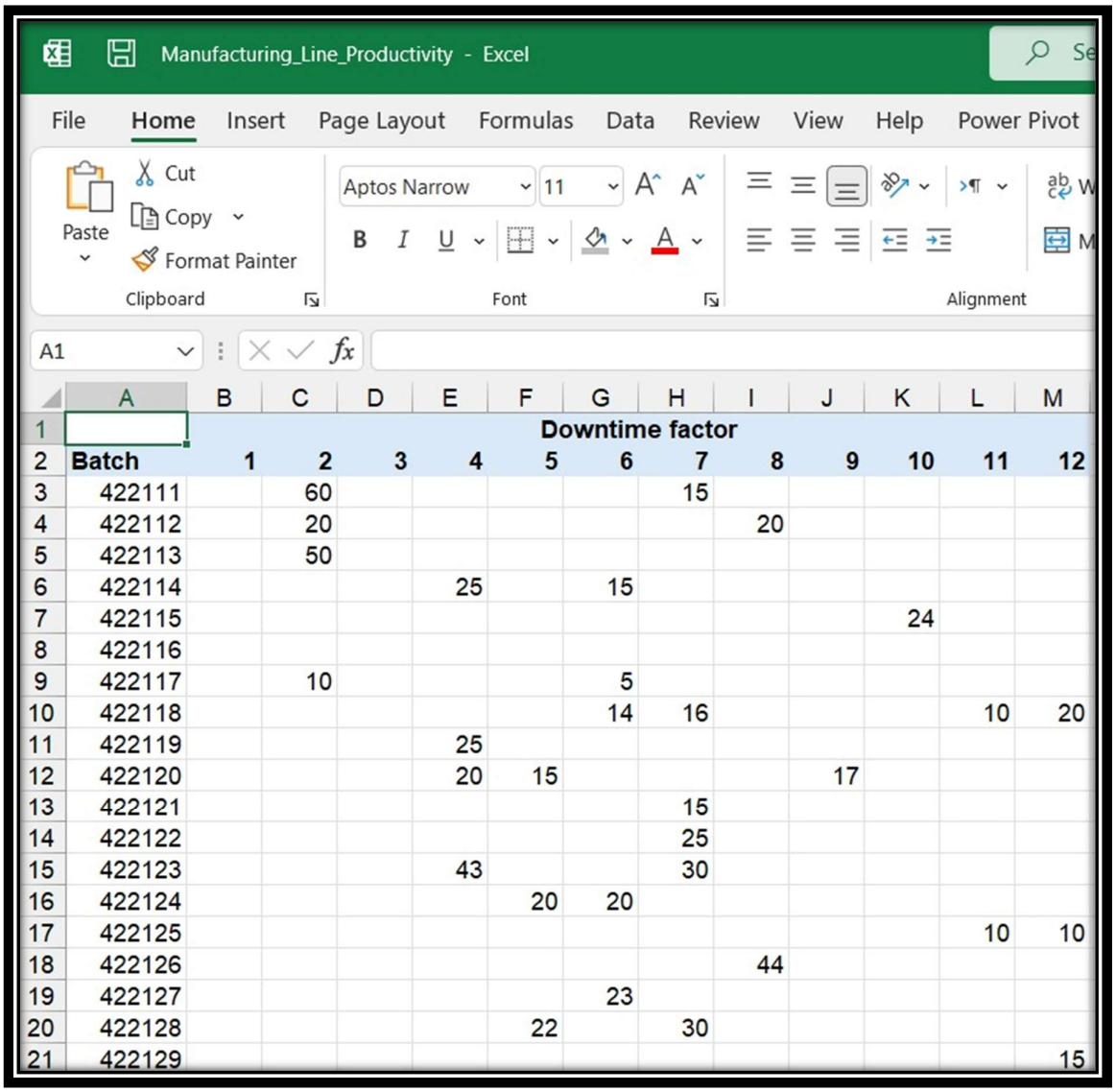
Data Cleaning & Preprocessing

Data Cleaning & Preprocessing

Handling Missing Values

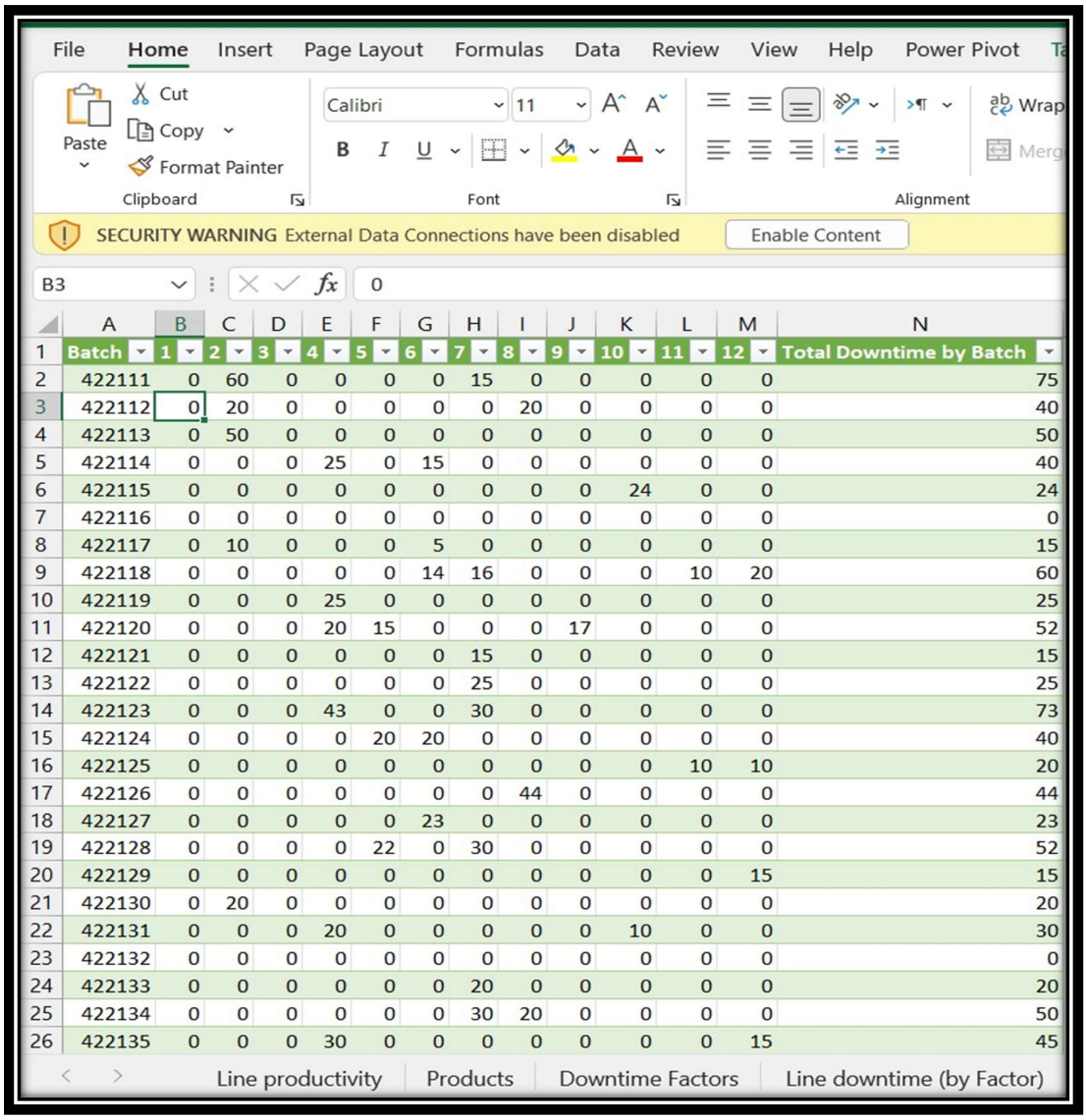
- **Identify missing data**
- **Impute missing values using appropriate techniques (e.g., mean, median, interpolation)**

Excel:



	A	B	C	D	E	F	G	H	I	J	K	L	M
1		Downtime factor											
2	Batch	1	2	3	4	5	6	7	8	9	10	11	12
3	422111		60					15					
4	422112		20						20				
5	422113		50										
6	422114				25		15						
7	422115										24		
8	422116												
9	422117		10				5						
10	422118						14	16				10	20
11	422119				25								
12	422120				20	15				17			
13	422121							15					
14	422122							25					
15	422123				43			30					
16	422124					20	20						
17	422125											10	10
18	422126								44				
19	422127						23						
20	422128					22		30					
21	422129												15

Replace NaN values with 0



File Home Insert Page Layout Formulas Data Review View Help Power Pivot

Cut Copy Format Painter Paste Clipboard

Calibri 11 Font

Alignment

SECURITY WARNING External Data Connections have been disabled Enable Content

B3 0

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Batch	1	2	3	4	5	6	7	8	9	10	11	12	Total Downtime by Batch
2	422111	0	60	0	0	0	0	15	0	0	0	0	0	75
3	422112	0	20	0	0	0	0	0	20	0	0	0	0	40
4	422113	0	50	0	0	0	0	0	0	0	0	0	0	50
5	422114	0	0	0	25	0	15	0	0	0	0	0	0	40
6	422115	0	0	0	0	0	0	0	0	0	24	0	0	24
7	422116	0	0	0	0	0	0	0	0	0	0	0	0	0
8	422117	0	10	0	0	0	5	0	0	0	0	0	0	15
9	422118	0	0	0	0	0	14	16	0	0	0	10	20	60
10	422119	0	0	0	25	0	0	0	0	0	0	0	0	25
11	422120	0	0	0	20	15	0	0	0	17	0	0	0	52
12	422121	0	0	0	0	0	0	15	0	0	0	0	0	15
13	422122	0	0	0	0	0	0	25	0	0	0	0	0	25
14	422123	0	0	0	43	0	0	30	0	0	0	0	0	73
15	422124	0	0	0	0	20	20	0	0	0	0	0	0	40
16	422125	0	0	0	0	0	0	0	0	0	0	10	10	20
17	422126	0	0	0	0	0	0	0	44	0	0	0	0	44
18	422127	0	0	0	0	0	23	0	0	0	0	0	0	23
19	422128	0	0	0	0	22	0	30	0	0	0	0	0	52
20	422129	0	0	0	0	0	0	0	0	0	0	0	15	15
21	422130	0	20	0	0	0	0	0	0	0	0	0	0	20
22	422131	0	0	0	20	0	0	0	0	0	10	0	0	30
23	422132	0	0	0	0	0	0	0	0	0	0	0	0	0
24	422133	0	0	0	0	0	0	20	0	0	0	0	0	20
25	422134	0	0	0	0	0	0	30	20	0	0	0	0	50
26	422135	0	0	0	30	0	0	0	0	0	0	0	15	45

Line productivity | Products | Downtime Factors | Line downtime (by Factor)

SQL:

```
update Line_downtime_factor
 set                one=coalesce(one , 0 ),
                    two=coalesce(two , 0 ),
                    three=coalesce(three , 0 ),
                    four=coalesce(four , 0 ),
                    five=coalesce(five , 0 ),
                    six=coalesce(six , 0 ),
                    seven=coalesce(seven , 0 ),
                    eight=coalesce(eight , 0 ),
                    nine=coalesce(nine , 0 ),
                    ten=coalesce(ten , 0 ),
                    eleven=coalesce(eleven , 0 ),
                    twelve=coalesce(twelve , 0 )
```

Python:

```
print(Line_downtime)
```

```
   Unnamed: 0  Downtime factor  Unnamed: 2  Unnamed: 3  Unnamed: 4  Unnamed: 5  Unnamed: 6  Unnamed: 7  Unnamed: 8  Unnamed: 9  Unnamed: 10  Unnamed: 11
Unnamed: 12
0       Batch             1.0         2.0         3.0         4.0         5.0         6.0         7.0         8.0         9.0         10.0         11.0
12.0
1      422111             NaN        60.0         NaN         NaN         NaN         NaN        15.0         NaN         NaN          NaN          NaN
NaN
2      422112             NaN        20.0         NaN         NaN         NaN         NaN         NaN        20.0         NaN          NaN          NaN
NaN
3      422113             NaN        50.0         NaN         NaN         NaN         NaN         NaN         NaN         NaN          NaN          NaN
NaN
4      422114             NaN         NaN         NaN        25.0         NaN        15.0         NaN         NaN         NaN          NaN          NaN
NaN
5      422115             NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN         24.0          NaN
NaN
6      422116             NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN         NaN          NaN          NaN
NaN
7      422117             NaN        10.0         NaN         NaN         NaN         5.0         NaN         NaN         NaN          NaN          NaN
NaN
8      422118             NaN         NaN         NaN         NaN         NaN        14.0        16.0         NaN         NaN          NaN         10.0
20.0
9      422119             NaN         NaN         NaN        25.0         NaN         NaN         NaN         NaN         NaN          NaN          NaN
NaN
10     422120             NaN         NaN         NaN        20.0        15.0         NaN         NaN         NaN        17.0          NaN          NaN
NaN
11     422121             NaN         NaN         NaN         NaN         NaN         NaN        15.0         NaN         NaN          NaN          NaN
NaN
12     422122             NaN         NaN         NaN         NaN         NaN         NaN        25.0         NaN         NaN          NaN          NaN
```

```
Line_downtime.head()
```

	Unnamed: 0	Downtime factor	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Unnamed: 10	Unnamed: 11	Unnamed: 12
0	Batch	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0
1	422111	0.0	60.0	0.0	0.0	0.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0
2	422112	0.0	20.0	0.0	0.0	0.0	0.0	0.0	20.0	0.0	0.0	0.0	0.0
3	422113	0.0	50.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	422114	0.0	0.0	0.0	25.0	0.0	15.0	0.0	0.0	0.0	0.0	0.0	0.0

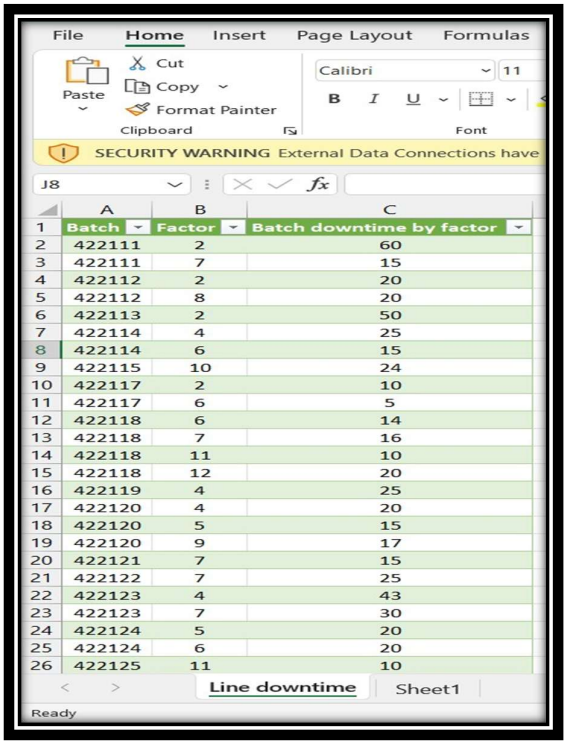
Data Transformation

Data Transformation:

- Convert timestamps to appropriate formats
- Normalize numerical features
- Encode categorical variables

- Remove extra spaces from column names and ensure they are strings
- Rename "Unnamed" downtime factor columns sequentially as "Factor 1" to "Factor 12"
- Identify the batch column (assuming it's the first column)
- Reshape the data using melt
- Reshape the data using melt
- Fill missing "Downtime Factor" cells correctly
- Remove rows where "Minutes" is NaN
- Convert "Batch" column to numeric type and sort from smallest to largest
- Reset index and remove it from display
- Display the transformed data and Save the transformed data to a new sheet in the same Excel file

Excel:



Batch	Factor	Batch downtime by factor
422111	2	60
422111	7	15
422112	2	20
422112	8	20
422113	2	50
422114	4	25
422114	6	15
422115	10	24
422117	2	10
422117	6	5
422118	6	14
422118	7	16
422118	11	10
422118	12	20
422119	4	25
422120	4	20
422120	5	15
422120	9	17
422121	7	15
422122	7	25
422123	4	43
422123	7	30
422124	5	20
422124	6	20
422125	11	10

SQL:

```
create table Line_downtime_factor1
            (Batch int not null,
             Factor int not null,
             Batch_downtime_by_factor int)
```

```
BULK INSERT Line_downtime_factor1
FROM 'D:\Data Analyst\Project DEPI\SQL\Line downtime 1.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
```

Python:

```python
# Read the sheet, skipping the first row
Line_downtime = pd.read_excel(file_path, sheet_name="Line downtime", header=1)

# Remove extra spaces from column names and ensure they are strings
Line_downtime.columns = Line_downtime.columns.astype(str).str.strip()

# Rename "Unnamed" downtime factor columns sequentially as "Factor 1" to "Factor 12"
for i, col in enumerate(Line_downtime.columns[1:], start=1):  # Skipping first column (Batch)
    if "Unnamed" in col or pd.isna(col):  # Handling NaN column names
        Line_downtime.rename(columns={col: f"Factor {i}"}, inplace=True)

# Identify the batch column (assuming it's the first column)
batch_col = Line_downtime.columns[0]  # First column should be "Batch"

# Reshape the data using melt
df_melted = Line_downtime.melt(id_vars=[batch_col], var_name="Downtime Factor", value_name="Minutes")

# Fill missing "Downtime Factor" cells correctly
if df_melted["Downtime Factor"].isna().sum() > 0:
    df_melted["Downtime Factor"] = df_melted["Downtime Factor"].fillna(method="ffill")


# Remove rows where "Minutes" is NaN
df_melted.dropna(subset=["Minutes"], inplace=True)

# Convert "Batch" column to numeric type and sort from smallest to largest
df_melted.dropna(subset=[batch_col], inplace=True)
df_melted.sort_values(by=[batch_col], inplace=True)


# Reset index and remove it from display
df_melted.reset_index(drop=True, inplace=True)

# Display the transformed data
print(df_melted.to_string(index=False))  # Removing index from output
```

Batch	Downtime Factor	Minutes
422111	2	60.0
422111	7	15.0
422112	2	20.0
422112	8	20.0
422113	2	50.0
422114	6	15.0
422114	4	25.0
422115	10	24.0
422117	2	10.0
422117	6	5.0
422118	6	14.0
422118	11	10.0
422118	7	16.0
422118	12	20.0
422119	4	25.0
422120	4	20.0
422120	9	17.0
422120	5	15.0
422121	7	15.0
422122	7	25.0
422123	4	43.0
422123	7	30.0
422124	5	20.0
422124	6	20.0
422125	11	10.0
422125	12	10.0
422126	8	44.0
422127	6	23.0
422128	5	22.0
422128	7	30.0
422129	12	15.0
422130	2	20.0
422131	10	10.0
422131	4	20.0
422133	7	20.0
422134	7	30.0
422134	8	20.0
422135	4	30.0
422135	12	15.0
422137	10	15.0
422137	8	30.0
422138	3	20.0
422139	6	15.0
422139	4	20.0
422140	6	50.0
422140	11	13.0
422141	12	7.0
422142	6	30.0
422143	6	40.0
422143	7	18.0
422144	8	24.0
422144	6	30.0
422145	3	22.0
422146	12	7.0
422146	7	25.0
422146	6	30.0
422147	4	17.0
422147	6	60.0
422147	7	30.0
422148	4	25.0
422148	8	7.0

SQL:

```sql
ALTER TABLE Line_productivity
ADD Duration_min AS
  DATEDIFF(MINUTE,CAST(CONVERT(VARCHAR, Date, 23) + ' ' + CONVERT(VARCHAR, Start_Time, 8) AS DATETIME2),
    CASE
      WHEN End_Time < Start_Time
        THEN CAST(CONVERT(VARCHAR, DATEADD(DAY, 1, Date), 23) + ' ' + CONVERT(VARCHAR, End_Time, 8) AS DATETIME2)
      ELSE CAST(CONVERT(VARCHAR, Date, 23) + ' ' + CONVERT(VARCHAR, End_Time, 8) AS DATETIME2)
    END)
```

- **Add Operator Error & Downtime Classification Columns**
- **Merge based on 'Downtime Factor' to bring in 'Operator Error**
- **Classify downtime as "Operator" or "Non-Operator"**
- **Drop the extra "Factor" column**

Python:

```
#Save the transformed data to a new sheet in the same Excel file
with pd.ExcelWriter(file_path, mode="a", engine="openpyxl", if_sheet_exists="replace") as writer:
    df_melted.to_excel(writer, sheet_name="Transformed Line downtime", index=False)

    print("Transformed data has been saved successfully!")
```

```
Transformed data has been saved successfully!
```

```
# Add Operator Error & Downtime Classification Columns

# Load specific sheets into DataFrames
df_downtime = pd.read_excel(file_path, sheet_name="Transformed Line downtime")
df_factors = pd.read_excel(file_path, sheet_name="Downtime factors")

# Merge based on 'Downtime Factor' to bring in 'Operator Error'
df_downtime = df_downtime.merge(df_factors[['Factor', 'Operator Error']],
                                left_on='Downtime Factor',
                                right_on='Factor',
                                how='left')

# Classify downtime as "Operator" or "Non-Operator"
df_downtime['Downtime Classification'] = df_downtime['Operator Error'].map({'Yes': 'Operator', 'No': 'Non-Operator'})

# Drop the extra "Factor" column
df_downtime.drop(columns=['Factor'], inplace=True)

# Load the original Excel file (to keep all sheets)
with pd.ExcelWriter(file_path, engine='openpyxl', mode='a', if_sheet_exists='replace') as writer:
    df_downtime.to_excel(writer, sheet_name="Transformed Line downtime", index=False)

    print("Sheet updated successfully!")
```

```
Sheet updated successfully!
```

Batch	vntime Fa	Minutes	Operator Error	Downtime Classification
422111	2	60	Yes	Operator
422111	7	15	No	Non-Operator
422112	2	20	Yes	Operator
422112	8	20	Yes	Operator
422113	2	50	Yes	Operator
422114	6	15	Yes	Operator
422114	4	25	No	Non-Operator
422115	10	24	Yes	Operator
422117	2	10	Yes	Operator
422117	6	5	Yes	Operator
422118	6	14	Yes	Operator
422118	11	10	Yes	Operator
422118	7	16	No	Non-Operator
422118	12	20	No	Non-Operator
422119	4	25	No	Non-Operator
422120	4	20	No	Non-Operator
422120	9	17	No	Non-Operator
422120	5	15	Yes	Operator
422121	7	15	No	Non-Operator
422122	7	25	No	Non-Operator
422123	4	43	No	Non-Operator
422123	7	30	No	Non-Operator

- **Convert L to mL.**
- **Read the "Products" sheet.**
- **Function to convert L to ml.**
- **Apply conversion.**

```python
# Convert L to mL.

# Read the "Products" sheet
df = pd.read_excel(file_path, sheet_name="Products")

# Function to convert L to ml
def convert_to_ml(size):
    if isinstance(size, str):
        if "L" in size:
            return float(size.replace("L", "").strip()) * 1000  # Convert Liters to ml
        elif "ml" in size:
            return float(size.replace("ml", "").strip())  # Keep ml values
    return size

# Apply conversion
df["Size"] = df["Size"].astype(str).apply(convert_to_ml)

# Load existing Excel file to preserve other sheets
with pd.ExcelWriter(file_path, engine="openpyxl", mode="a", if_sheet_exists="replace") as writer:
    df.to_excel(writer, sheet_name="Transformed Products", index=False)
```

Product	Flavor	Size	Min batch time
OR-600	Orange	600	60
LE-600	Lemon lime	600	60
CO-600	Cola	600	60
DC-600	Diet Cola	600	60
RB-600	Root Berry	600	60
CO-2L	Cola	2000	98

- **Add Batch Duration Column.**

```
# Convert "Start Time" and "End Time" to datetime format, keeping NaT for missing values
df_productivity["Start Time"] = pd.to_datetime(df_productivity["Start Time"], format="%H:%M:%S", errors="coerce")
df_productivity["End Time"] = pd.to_datetime(df_productivity["End Time"], format="%H:%M:%S", errors="coerce")

# Calculate the time difference (Batch Duration) in minutes
df_productivity["Batch Duration"] = (df_productivity["End Time"] - df_productivity["Start Time"]).dt.total_seconds() / 60

# Handle missing values
df_productivity.loc[df_productivity["End Time"].isna(), "Batch Duration"] = None
df_productivity.loc[df_productivity["Start Time"].isna(), "Batch Duration"] = None

# Handle cases where End Time is before Start Time (e.g., shift past midnight)
df_productivity["Batch Duration"] = df_productivity["Batch Duration"].apply(lambda x: x + 1440 if x < 0 else x)

# Convert "Start Time" and "End Time" back to string format to avoid date display in Excel
df_productivity["Start Time"] = df_productivity["Start Time"].dt.strftime("%H:%M:%S")
df_productivity["End Time"] = df_productivity["End Time"].dt.strftime("%H:%M:%S")

# Ensure "Batch Duration" remains numeric to prevent Excel from treating it as a date
df_productivity["Batch Duration"] = df_productivity["Batch Duration"].astype(float)

# ✅ Write the transformed data to a new sheet without modifying "Line productivity"
with pd.ExcelWriter(file_path, engine="openpyxl", mode="a") as writer:
    df_productivity.to_excel(writer, sheet_name="Transformed Line Productivity", index=False, float_format="%.2f")

# ✅ Display the transformed DataFrame
print(df_productivity)

print("Sheet 'Transformed Line Productivity' created successfully!")
```

- **Convert "Start Time" and "End Time" to datetime format, keeping NaT for missing values.**
- **Calculate the time difference (Batch Duration) in minutes.**
- **Handle missing values.**
- **Handle cases where End Time is before Start Time (e.g., shift past midnight).**
- **Convert "Start Time" and "End Time" back to string format to avoid date display in Excel.**
- **Ensure "Batch Duration" remains numeric to prevent Excel from treating it as a date.**

	Date	Product	Batch	Operator	Start Time	End Time	Batch Duration
0	2024-08-29	OR-600	422111	Mac	11:50:00	14:05:00	135.0
1	2024-08-29	LE-600	422112	Mac	14:05:00	15:45:00	100.0
2	2024-08-29	LE-600	422113	Mac	15:45:00	17:35:00	110.0
3	2024-08-29	LE-600	422114	Mac	17:35:00	19:15:00	100.0
4	2024-08-29	LE-600	422115	Charlie	19:15:00	20:39:00	84.0
5	2024-08-29	LE-600	422116	Charlie	20:39:00	21:39:00	60.0
6	2024-08-29	LE-600	422117	Charlie	21:39:00	22:54:00	75.0
7	2024-08-30	CO-600	422118	Dee	04:05:00	06:05:00	120.0
8	2024-08-30	CO-600	422119	Dee	06:05:00	07:30:00	85.0
9	2024-08-30	CO-600	422120	Dee	07:30:00	09:22:00	112.0
10	2024-08-30	CO-600	422121	Dennis	09:22:00	10:37:00	75.0
11	2024-08-30	CO-600	422122	Dennis	10:37:00	12:02:00	85.0
12	2024-08-30	CO-600	422123	Dennis	12:02:00	14:15:00	133.0
13	2024-08-30	CO-600	422124	Dennis	14:15:00	15:55:00	100.0
14	2024-08-30	CO-600	422125	Charlie	15:55:00	17:15:00	80.0
15	2024-08-30	CO-600	422126	Charlie	17:15:00	18:59:00	104.0
16	2024-08-30	CO-600	422127	Charlie	18:59:00	20:22:00	83.0
17	2024-08-30	CO-600	422128	Charlie	20:22:00	22:14:00	112.0
18	2024-08-30	CO-600	422129	Charlie	22:14:00	23:29:00	75.0
19	2024-08-31	CO-600	422130	Dee	07:45:00	09:05:00	80.0
20	2024-08-31	CO-600	422131	Dee	09:05:00	10:35:00	90.0
21	2024-08-31	CO-600	422132	Dee	10:35:00	11:35:00	60.0
22	2024-08-31	DC-600	422133	Dee	11:35:00	12:55:00	80.0
23	2024-08-31	DC-600	422134	Mac	12:55:00	14:45:00	110.0
24	2024-08-31	DC-600	422135	Mac	14:45:00	16:30:00	105.0
25	2024-08-31	DC-600	422136	Mac	16:30:00	17:30:00	60.0
26	2024-09-02	RB-600	422137	Dee	01:00:00	02:45:00	105.0
27	2024-09-02	RB-600	422138	Dee	02:45:00	04:05:00	80.0
28	2024-09-02	RB-600	422139	Dee	04:05:00	05:40:00	95.0
29	2024-09-02	RB-600	422140	Dee	05:40:00	07:43:00	123.0
30	2024-09-02	RB-600	422141	Dennis	07:43:00	08:50:00	67.0
31	2024-09-02	RB-600	422142	Dennis	08:50:00	10:20:00	90.0
32	2024-09-02	RB-600	422143	Dennis	10:20:00	12:18:00	118.0
33	2024-09-02	CO-2L	422144	Dennis	12:18:00	14:50:00	152.0
34	2024-09-02	CO-2L	422145	Charlie	14:50:00	16:50:00	120.0
35	2024-09-02	CO-2L	422146	Charlie	16:50:00	19:30:00	160.0
36	2024-09-02	CO-2L	422147	Charlie	19:30:00	22:55:00	205.0
37	2024-09-03	CO-2L	422148	Mac	00:05:00	02:15:00	130.0

Sheet 'Transformed Line Productivity' created successfully!

Add Batch Min batch time Column

```python
# Load the necessary sheets
df_productivity = pd.read_excel(file_path, sheet_name="Transformed Line Productivity")
df_products = pd.read_excel(file_path, sheet_name="Transformed Products")

# Ensure necessary columns exist
required_columns_productivity = ["Product"]
required_columns_products = ["Product", "Min batch time"]

for col in required_columns_productivity:
    if col not in df_productivity.columns:
        raise KeyError(f"❌ '{col}' column not found in 'Transformed Line Productivity' sheet.")

for col in required_columns_products:
    if col not in df_products.columns:
        raise KeyError(f"❌ '{col}' column not found in 'Products' sheet.")

# ✅ Merge "Min batch time" from "Transformed Products" into "Transformed Line Productivity"
df_productivity = df_productivity.merge(df_products[["Product", "Min batch time"]], on="Product", how="left")

# ✅ Save the updated sheet back to Excel, replacing the old one
with pd.ExcelWriter(file_path, engine="openpyxl", mode="a", if_sheet_exists="replace") as writer:
    df_productivity.to_excel(writer, sheet_name="Transformed Line Productivity", index=False)

# ✅ Display the updated DataFrame
print(df_productivity)
print("✅ 'Transformed Line Productivity' sheet updated successfully!")
```

Load the necessary sheets Transformed Line Productivity & Transformed Products

Ensure necessary columns exist

✅ **Merge "Min batch time" from "Transformed Products" into "Transformed Line Productivity"**

✅ **Save the updated sheet back to Excel**

	Date	Product	Batch	Operator	Start Time	End Time	Batch Duration	Min batch time
0	2024-08-29	OR-600	422111	Mac	11:50:00	14:05:00	135	60
1	2024-08-29	LE-600	422112	Mac	14:05:00	15:45:00	100	60
2	2024-08-29	LE-600	422113	Mac	15:45:00	17:35:00	110	60
3	2024-08-29	LE-600	422114	Mac	17:35:00	19:15:00	100	60
4	2024-08-29	LE-600	422115	Charlie	19:15:00	20:39:00	84	60
5	2024-08-29	LE-600	422116	Charlie	20:39:00	21:39:00	60	60
6	2024-08-29	LE-600	422117	Charlie	21:39:00	22:54:00	75	60
7	2024-08-30	CO-600	422118	Dee	04:05:00	06:05:00	120	60
8	2024-08-30	CO-600	422119	Dee	06:05:00	07:30:00	85	60
9	2024-08-30	CO-600	422120	Dee	07:30:00	09:22:00	112	60
10	2024-08-30	CO-600	422121	Dennis	09:22:00	10:37:00	75	60
11	2024-08-30	CO-600	422122	Dennis	10:37:00	12:02:00	85	60
12	2024-08-30	CO-600	422123	Dennis	12:02:00	14:15:00	133	60
13	2024-08-30	CO-600	422124	Dennis	14:15:00	15:55:00	100	60
14	2024-08-30	CO-600	422125	Charlie	15:55:00	17:15:00	80	60
15	2024-08-30	CO-600	422126	Charlie	17:15:00	18:59:00	104	60
16	2024-08-30	CO-600	422127	Charlie	18:59:00	20:22:00	83	60
17	2024-08-30	CO-600	422128	Charlie	20:22:00	22:14:00	112	60
18	2024-08-30	CO-600	422129	Charlie	22:14:00	23:29:00	75	60
19	2024-08-31	CO-600	422130	Dee	07:45:00	09:05:00	80	60
20	2024-08-31	CO-600	422131	Dee	09:05:00	10:35:00	90	60
21	2024-08-31	CO-600	422132	Dee	10:35:00	11:35:00	60	60
22	2024-08-31	DC-600	422133	Dee	11:35:00	12:55:00	80	60
23	2024-08-31	DC-600	422134	Mac	12:55:00	14:45:00	110	60
24	2024-08-31	DC-600	422135	Mac	14:45:00	16:30:00	105	60
25	2024-08-31	DC-600	422136	Mac	16:30:00	17:30:00	60	60
26	2024-09-02	RB-600	422137	Dee	01:00:00	02:45:00	105	60
27	2024-09-02	RB-600	422138	Dee	02:45:00	04:05:00	80	60
28	2024-09-02	RB-600	422139	Dee	04:05:00	05:40:00	95	60
29	2024-09-02	RB-600	422140	Dee	05:40:00	07:43:00	123	60
30	2024-09-02	RB-600	422141	Dennis	07:43:00	08:50:00	67	60
31	2024-09-02	RB-600	422142	Dennis	08:50:00	10:20:00	90	60
32	2024-09-02	RB-600	422143	Dennis	10:20:00	12:18:00	118	60
33	2024-09-02	CO-2L	422144	Dennis	12:18:00	14:50:00	152	98
34	2024-09-02	CO-2L	422145	Charlie	14:50:00	16:50:00	120	98
35	2024-09-02	CO-2L	422146	Charlie	16:50:00	19:30:00	160	98
36	2024-09-02	CO-2L	422147	Charlie	19:30:00	22:55:00	205	98
37	2024-09-03	CO-2L	422148	Mac	00:05:00	02:15:00	130	98

✅ 'Transformed Line Productivity' sheet updated successfully!

Add Batch Total Downtime by batch_Column

```python
# Load the "Transformed Line Productivity" sheet
df_productivity = pd.read_excel(file_path, sheet_name="Transformed Line Productivity")

# ✅ Ensure "Batch Duration" and "Min batch time" are numeric
df_productivity["Batch Duration"] = pd.to_numeric(df_productivity["Batch Duration"], errors="coerce")
df_productivity["Min batch time"] = pd.to_numeric(df_productivity["Min batch time"], errors="coerce")

# ✅ Handle missing values before calculations
df_productivity["Batch Duration"] = df_productivity["Batch Duration"].fillna(0)
df_productivity["Min batch time"] = df_productivity["Min batch time"].fillna(0)

# ✅ Calculate "Total Downtime by Batch" (Minutes)
df_productivity["Total Downtime by Batch"] = df_productivity["Batch Duration"] - df_productivity["Min batch time"]

# ✅ Ensure "Total Downtime by Batch" has no NaN values
df_productivity["Total Downtime by Batch"] = df_productivity["Total Downtime by Batch"].fillna(0)

# ✅ Ensure "Batch Duration" is always positive (Handle negative values if needed)
df_productivity["Batch Duration"] = df_productivity["Batch Duration"].apply(lambda x: x + 1440 if x < 0 else x)

# ✅ Save back to "Transformed Line Productivity" without modifying other sheets
with pd.ExcelWriter(file_path, engine="openpyxl", mode="a", if_sheet_exists="replace") as writer:
    df_productivity.to_excel(writer, sheet_name="Transformed Line Productivity", index=False, float_format="%.2f")

# ✅ Display the updated DataFrame
print(df_productivity)
print("✅ 'Transformed Line Productivity' sheet updated successfully!")
```

Load the "Transformed Line Productivity" sheet

- ☑ **Ensure "Batch Duration" and "Min batch time" are numeric**
- ☑ **Handle missing values before calculations**
- ☑ **Calculate "Total Downtime by Batch" (Minutes)**
- ☑ **Ensure "Total Downtime by Batch" has no NaN values**
- ☑ **Ensure "Batch Duration" is always positive (Handle negative values if needed)**
- ☑ **Save back to "Transformed Line Productivity" without modifying other sheets**

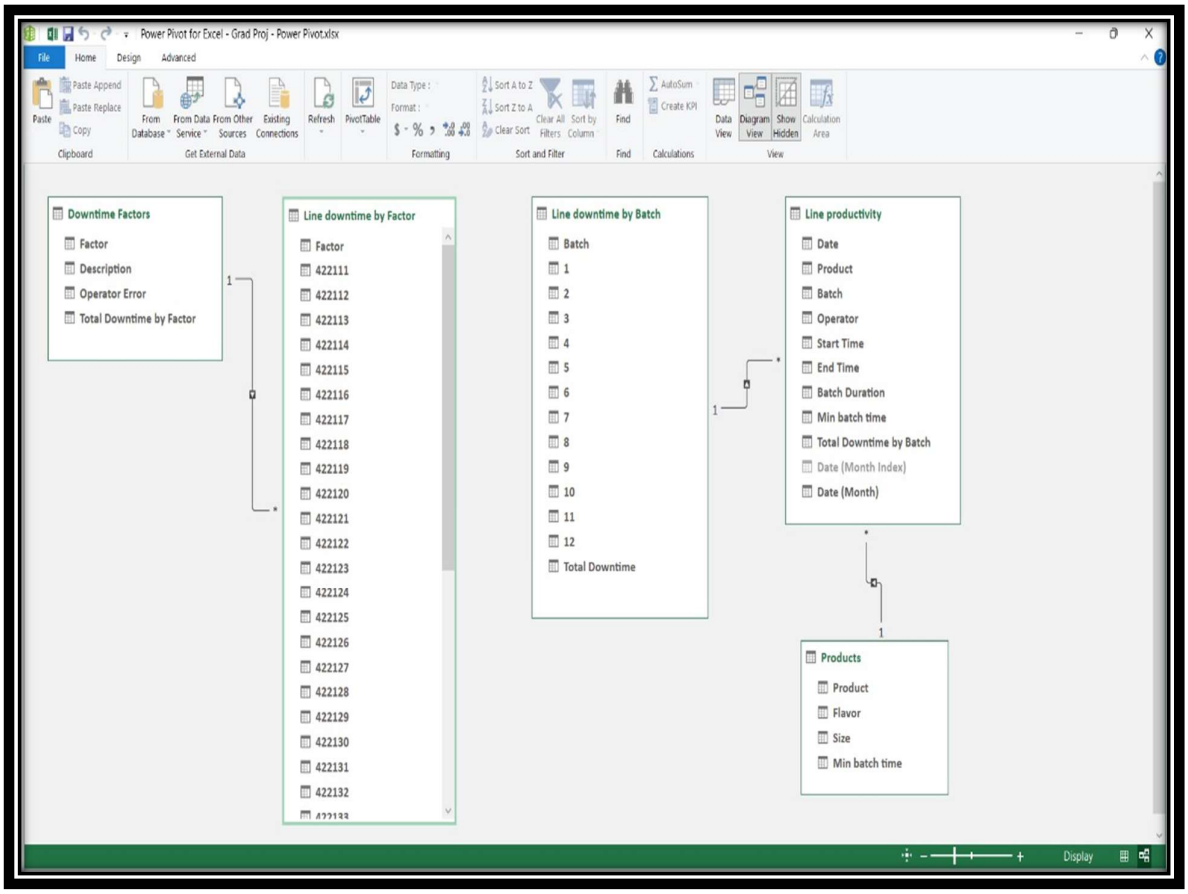
	Date	Product	Batch	Operator	Start Time	End Time	Batch Duration	Min batch time	Total Downtime by Batch
0	2024-08-29	OR-600	422111	Mac	11:50:00	14:05:00	135	60	75
1	2024-08-29	LE-600	422112	Mac	14:05:00	15:45:00	100	60	40
2	2024-08-29	LE-600	422113	Mac	15:45:00	17:35:00	110	60	50
3	2024-08-29	LE-600	422114	Mac	17:35:00	19:15:00	100	60	40
4	2024-08-29	LE-600	422115	Charlie	19:15:00	20:39:00	84	60	24
5	2024-08-29	LE-600	422116	Charlie	20:39:00	21:39:00	60	60	0
6	2024-08-29	LE-600	422117	Charlie	21:39:00	22:54:00	75	60	15
7	2024-08-30	CO-600	422118	Dee	04:05:00	06:05:00	120	60	60
8	2024-08-30	CO-600	422119	Dee	06:05:00	07:30:00	85	60	25
9	2024-08-30	CO-600	422120	Dee	07:30:00	09:22:00	112	60	52
10	2024-08-30	CO-600	422121	Dennis	09:22:00	10:37:00	75	60	15
11	2024-08-30	CO-600	422122	Dennis	10:37:00	12:02:00	85	60	25
12	2024-08-30	CO-600	422123	Dennis	12:02:00	14:15:00	133	60	73
13	2024-08-30	CO-600	422124	Dennis	14:15:00	15:55:00	100	60	40
14	2024-08-30	CO-600	422125	Charlie	15:55:00	17:15:00	80	60	20
15	2024-08-30	CO-600	422126	Charlie	17:15:00	18:59:00	104	60	44
16	2024-08-30	CO-600	422127	Charlie	18:59:00	20:22:00	83	60	23
17	2024-08-30	CO-600	422128	Charlie	20:22:00	22:14:00	112	60	52
18	2024-08-30	CO-600	422129	Charlie	22:14:00	23:29:00	75	60	15
19	2024-08-31	CO-600	422130	Dee	07:45:00	09:05:00	80	60	20
20	2024-08-31	CO-600	422131	Dee	09:05:00	10:35:00	90	60	30
21	2024-08-31	CO-600	422132	Dee	10:35:00	11:35:00	60	60	0
22	2024-08-31	DC-600	422133	Dee	11:35:00	12:55:00	80	60	20
23	2024-08-31	DC-600	422134	Mac	12:55:00	14:45:00	110	60	50
24	2024-08-31	DC-600	422135	Mac	14:45:00	16:30:00	105	60	45
25	2024-08-31	DC-600	422136	Mac	16:30:00	17:30:00	60	60	0
26	2024-09-02	RB-600	422137	Dee	01:00:00	02:45:00	105	60	45
27	2024-09-02	RB-600	422138	Dee	02:45:00	04:05:00	80	60	20
28	2024-09-02	RB-600	422139	Dee	04:05:00	05:40:00	95	60	35
29	2024-09-02	RB-600	422140	Dee	05:40:00	07:43:00	123	60	63
30	2024-09-02	RB-600	422141	Dennis	07:43:00	08:50:00	67	60	7
31	2024-09-02	RB-600	422142	Dennis	08:50:00	10:20:00	90	60	30
32	2024-09-02	RB-600	422143	Dennis	10:20:00	12:18:00	118	60	58
33	2024-09-02	CO-2L	422144	Dennis	12:18:00	14:50:00	152	98	54
34	2024-09-02	CO-2L	422145	Charlie	14:50:00	16:50:00	120	98	22
35	2024-09-02	CO-2L	422146	Charlie	16:50:00	19:30:00	160	98	62
36	2024-09-02	CO-2L	422147	Charlie	19:30:00	22:55:00	205	98	107
37	2024-09-03	CO-2L	422148	Mac	00:05:00	02:15:00	130	98	32

✅ 'Transformed Line Productivity' sheet updated successfully!

Data Modeling

Data Modeling:

Excel:



SQL:

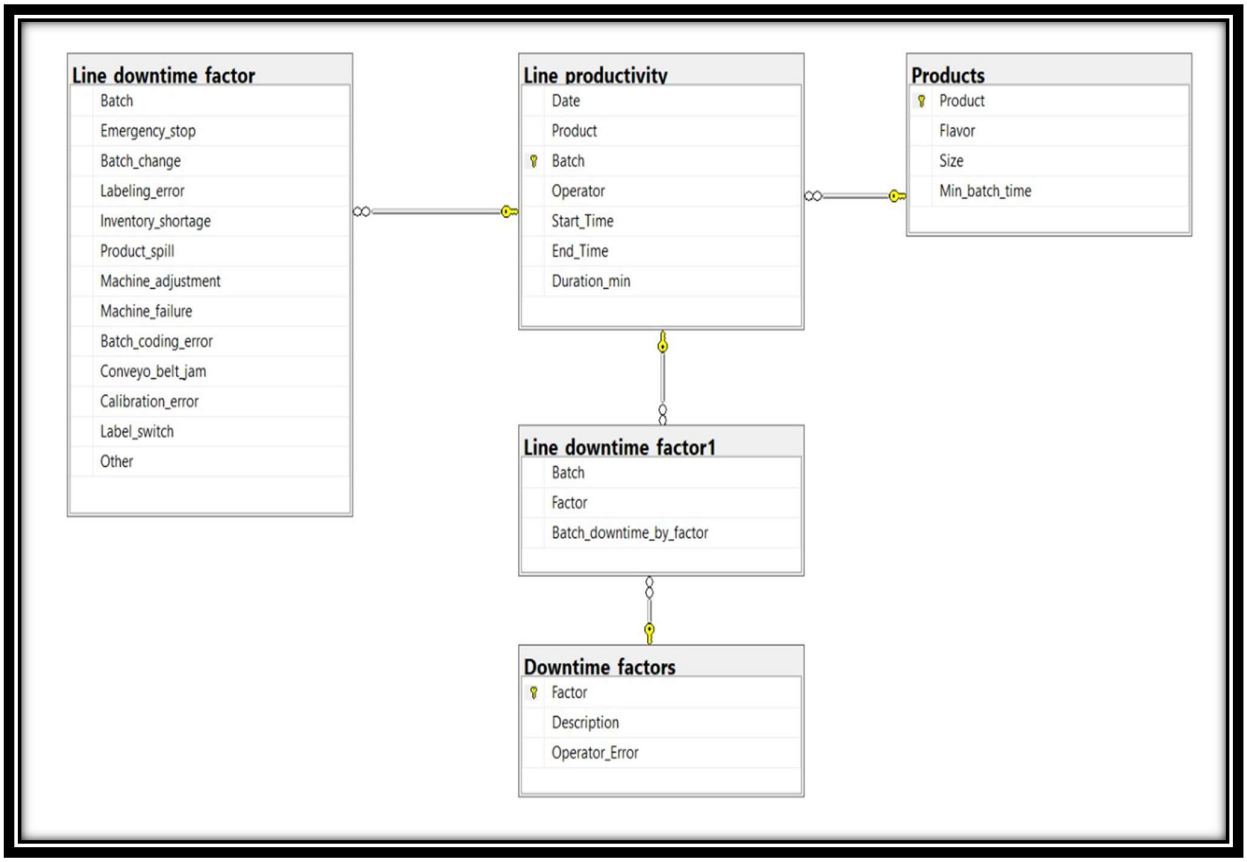
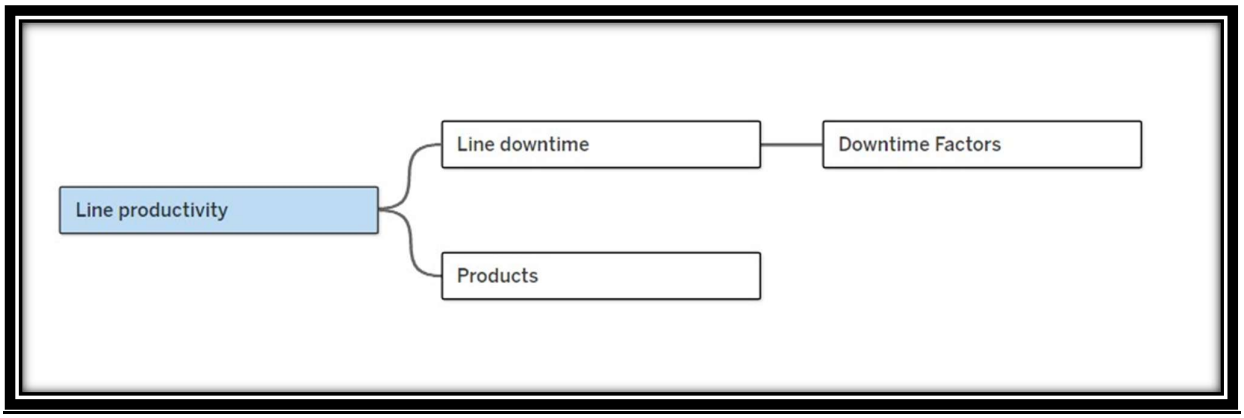

Line downtime factor
Batch
Emergency_stop
Batch_change
Labeling_error
Inventory_shortage
Product_spill
Machine_adjustment
Machine_failure
Batch_coding_error
Conveyo_belt_jam
Calibration_error
Label_switch
Other
Line productivity
Date
Product
Batch
Operator
Start_Time
End_Time
Duration_min
Products
Product
Flavor
Size
Min_batch_time
Line downtime factor1
Batch
Factor
Batch_downtime_by_factor
Downtime factors
Factor
Description
Operator_Error

Tableau:


Line productivity
Line downtime
Downtime Factors
Products

Phase II

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA).

Statistical Summary

- Distribution of downtime durations

Data Visualizations

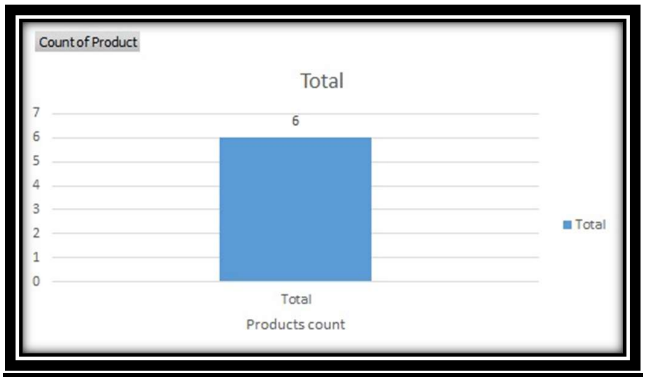
- Time series plots of downtime occurrences
- Downtime frequency by machine and cause

Analysis Questions

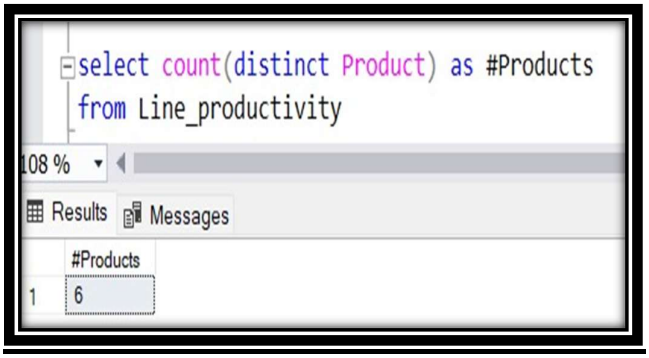
Productivity Analysis

Q1) How many different products were produced?

Excel:



SQL:

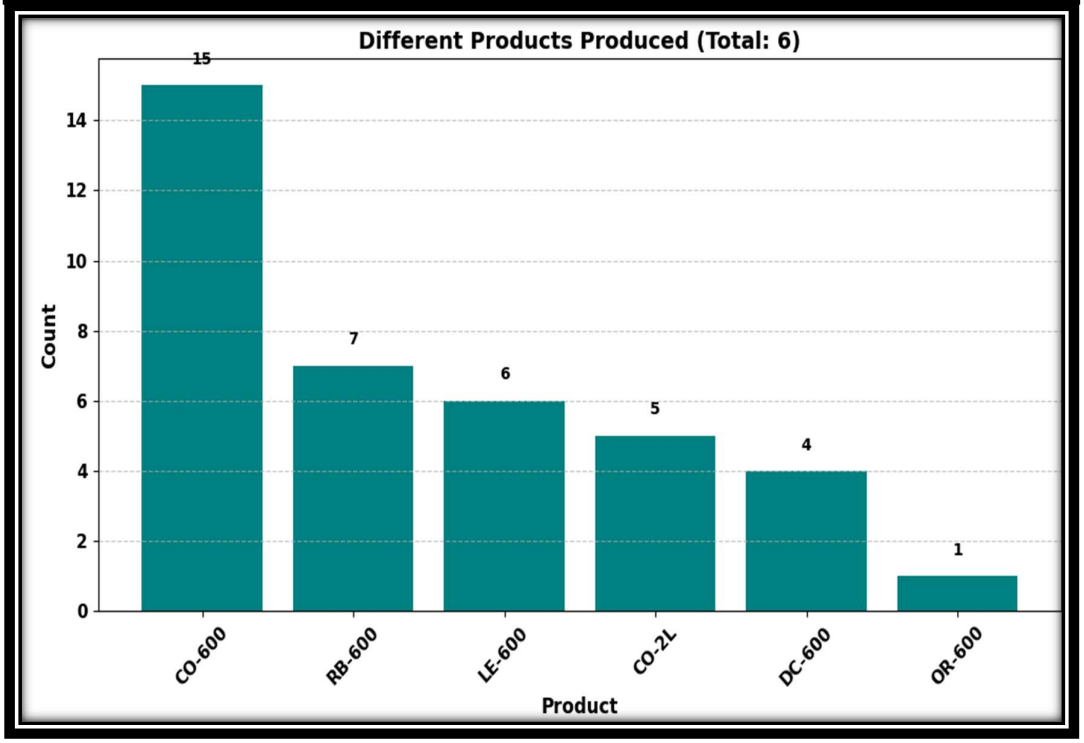


Python

```python
#Q1) How many different products were produced?

df = pd.read_excel(file_path, sheet_name="Transformed Line Productivity")

# Ensure column names are stripped of spaces
df.columns = df.columns.str.strip()

# Count unique products
unique_products = df["Product"].nunique()

# Print result
print(f"Total Number of Different Products Produced: {unique_products}")
```

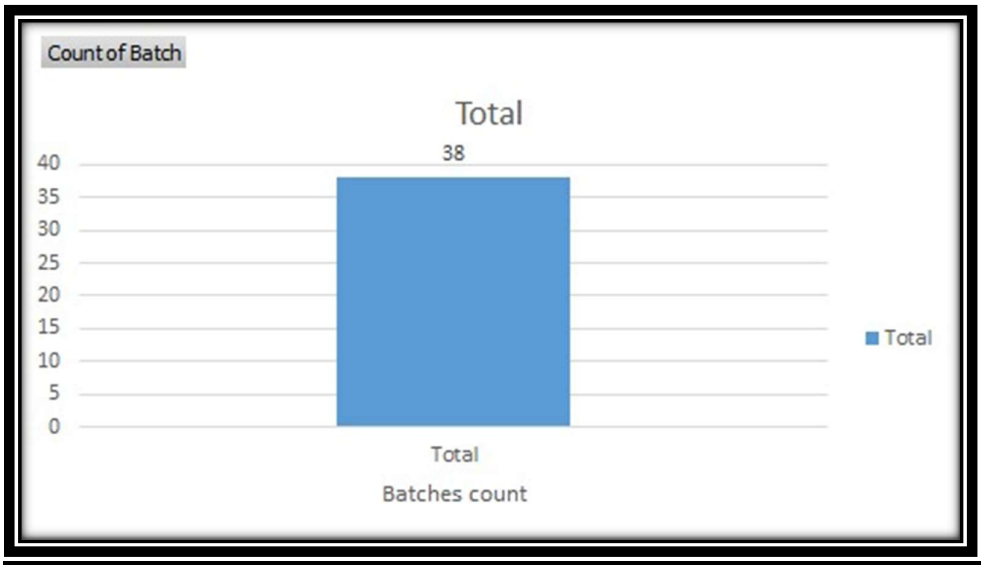
```
Total Number of Different Products Produced: 6
```

```python
#Q1) How many different products were produced?

# Clean column names
df.columns = df.columns.str.strip()

# Count of each product
product_counts = df["Product"].value_counts()

# Plotting
plt.figure(figsize=(10, 6))
bars = plt.bar(product_counts.index, product_counts.values, color='teal')

# Add labels and title
plt.title(f"Different Products Produced (Total: {df['Product'].nunique()})", fontsize=14, fontweight='bold')
plt.xlabel("Product", fontsize=12, fontweight='bold')
plt.ylabel("Count", fontsize=12, fontweight='bold')
plt.xticks(rotation=45, fontsize=11, fontweight='bold')
plt.yticks(fontsize=11, fontweight='bold')
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Show count labels on top of bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.5, int(yval),
             ha='center', va='bottom', fontsize=10, fontweight='bold')

plt.tight_layout()
plt.show()
```

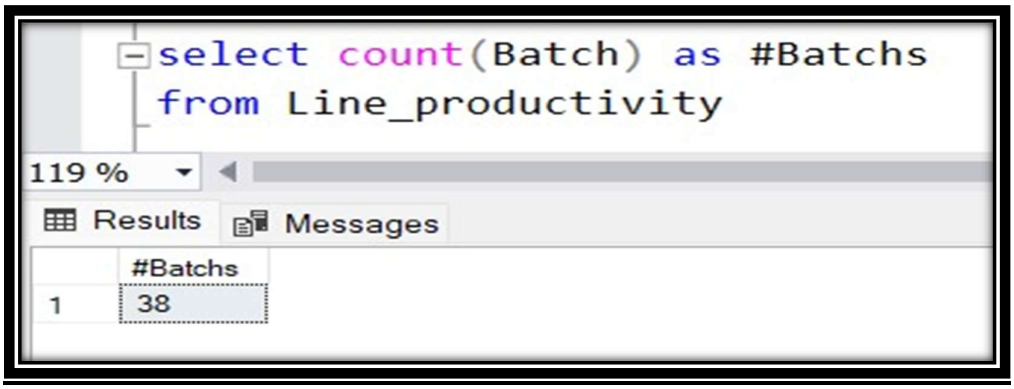


Q2) What is the total number of batches produced?

Excel:

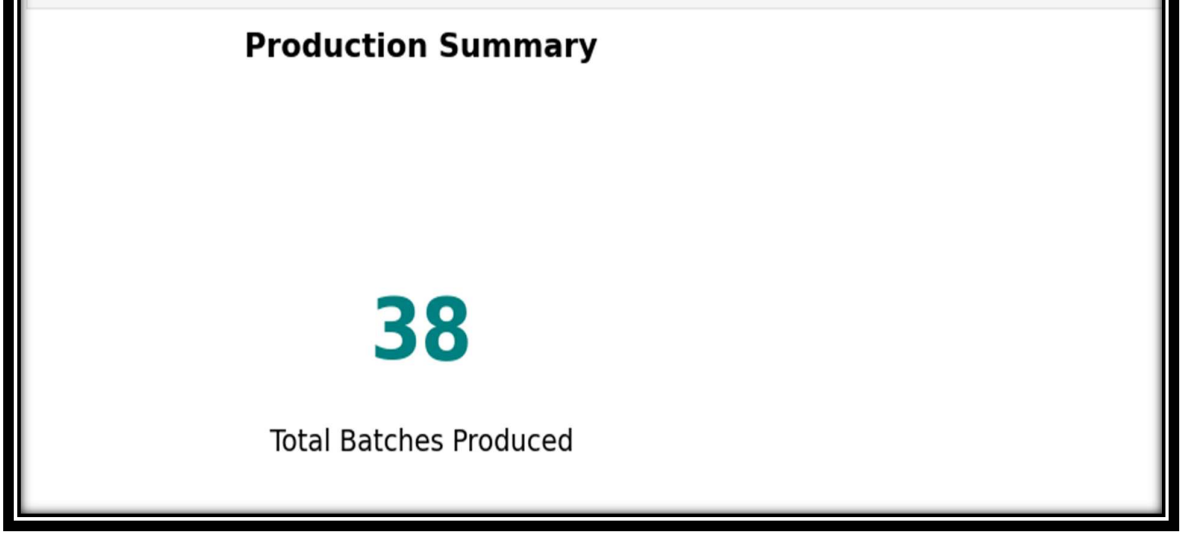


SQL:

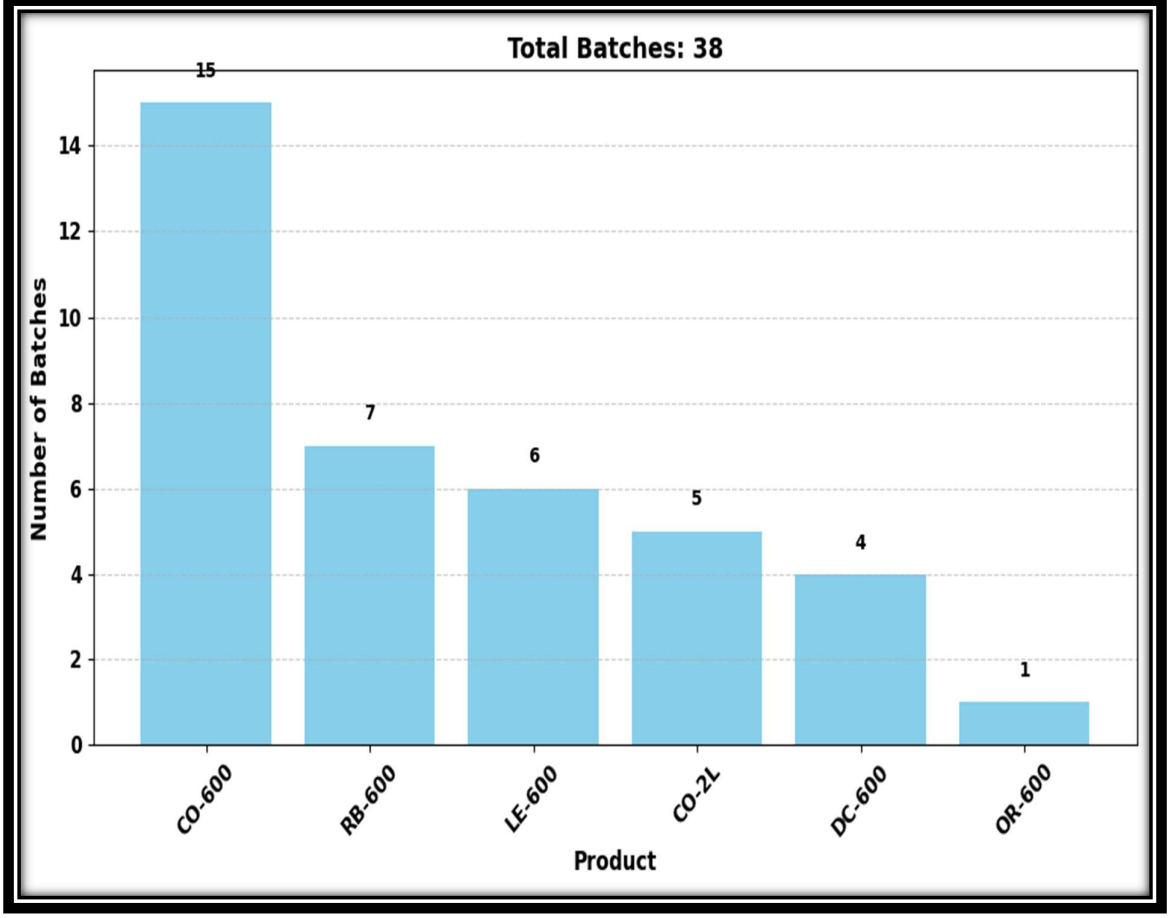


Python:

```python
#Q2) What is the total number of batches produced?

# Total batches produced
total_batches = df["Batch"].count()

# Create a figure
plt.figure(figsize=(6, 4))
plt.text(0.5, 0.5, f"{total_batches:,}", fontsize=40, fontweight="bold", ha="center", va="center", color="teal")
plt.text(0.5, 0.3, "Total Batches Produced", fontsize=14, ha="center", va="center")

# Remove axes
plt.axis("off")
plt.title("Production Summary", fontsize=16, fontweight="bold")
plt.tight_layout()
plt.show()
```

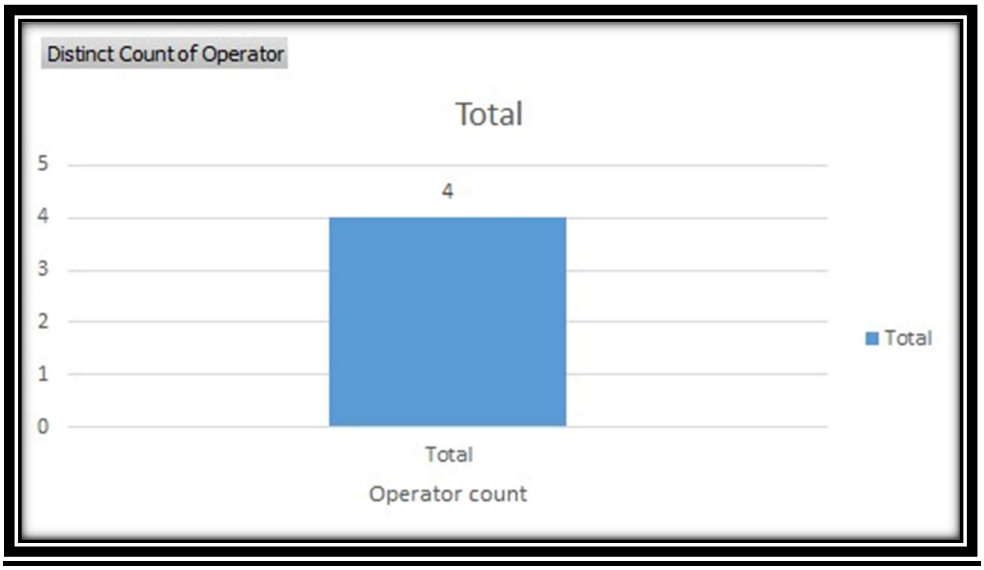


```python
#Q2) What is the total number of batches produced?

# Count batches per product
batches_per_product = df["Product"].value_counts()

# Plot
plt.figure(figsize=(10, 6))
bars = plt.bar(batches_per_product.index, batches_per_product.values, color='skyblue')

# Labels and title
plt.title(f"Total Batches: {total_batches}", fontsize=14, fontweight="bold")
plt.xlabel("Product", fontsize=12, fontweight="bold")
plt.ylabel("Number of Batches", fontsize=12, fontweight="bold")
plt.xticks(rotation=45, fontsize=11, fontweight='bold')
plt.yticks(fontsize=11, fontweight='bold')
plt.grid(axis="y", linestyle="--", alpha=0.6)

# Add value labels on bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.5, int(yval),
             ha='center', va='bottom', fontsize=10, fontweight='bold')

plt.tight_layout()
plt.show()
```

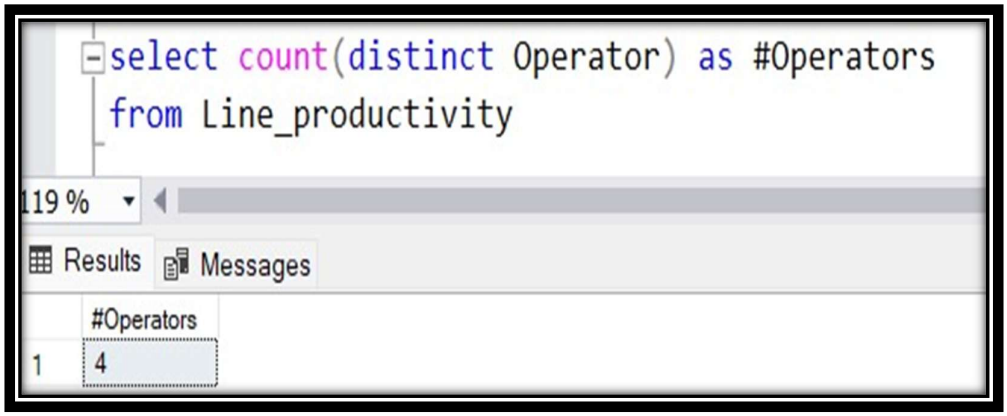


Q3) How many operators were involved in production?

Excel:



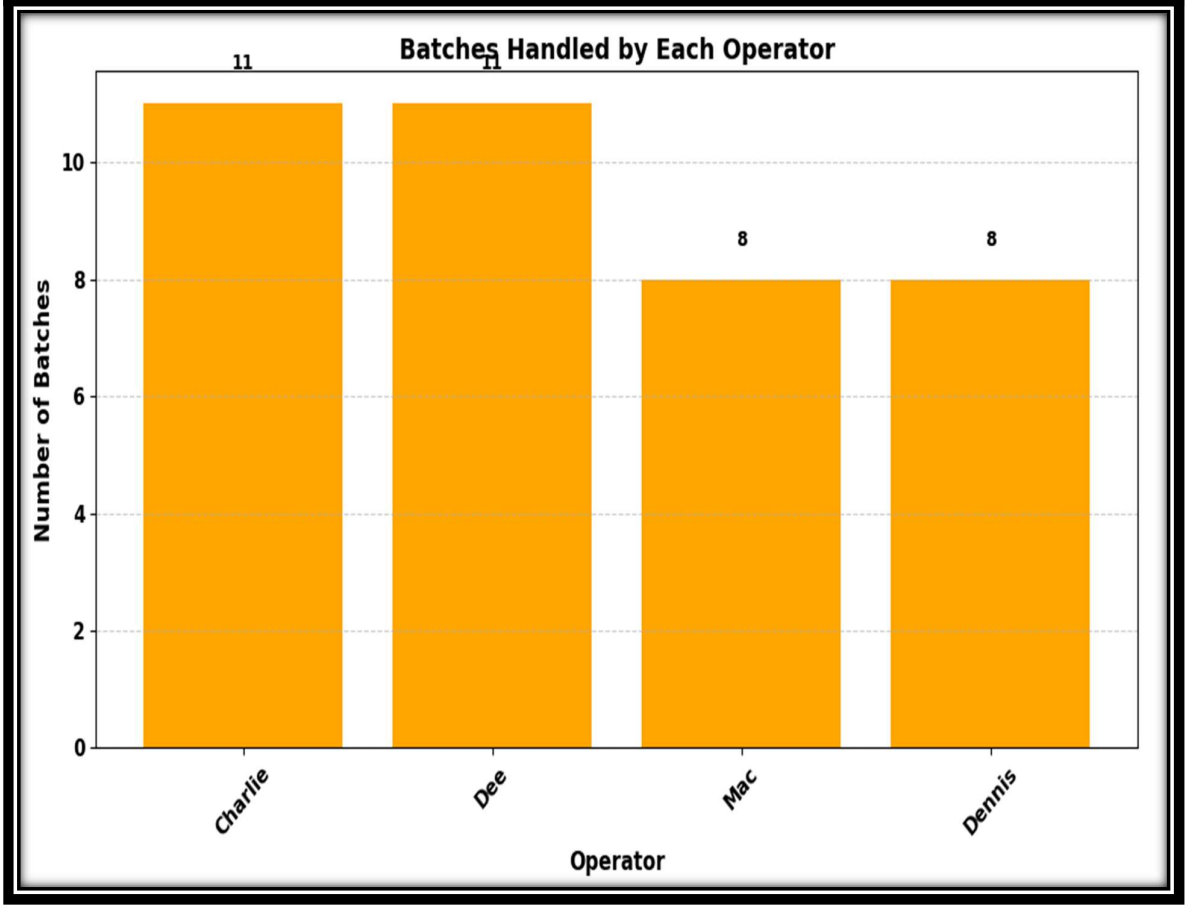
SQL:



Python:

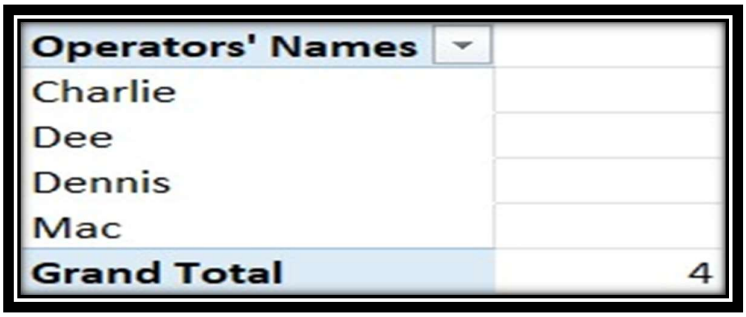
```python
#Q3) How many operators were involved in production?

# Total number of unique operators
num_operators = df["Operator"].nunique()

# Big number chart
plt.figure(figsize=(6, 4))
plt.text(0.5, 0.5, f"{num_operators}", fontsize=40, fontweight="bold", ha="center", va="center", color="darkgreen")
plt.text(0.5, 0.3, "Unique Operators Involved", fontsize=14, ha="center", va="center")

plt.axis("off")
plt.title("Operator Participation Summary", fontsize=16, fontweight="bold")
plt.tight_layout()
plt.show()
```

```python
# Count batches per operator
batches_per_operator = df["Operator"].value_counts()

# Plot bar chart
plt.figure(figsize=(10, 6))
bars = plt.bar(batches_per_operator.index, batches_per_operator.values, color='orange')

# Labels and formatting
plt.title("Batches Handled by Each Operator", fontsize=14, fontweight="bold")
plt.xlabel("Operator", fontsize=12, fontweight="bold")
plt.ylabel("Number of Batches", fontsize=12, fontweight="bold")
plt.xticks(rotation=45, fontsize=11, fontweight='bold')
plt.yticks(fontsize=11, fontweight='bold')
plt.grid(axis="y", linestyle="--", alpha=0.6)

# Add data labels on bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.5, int(yval),
             ha='center', va='bottom', fontsize=10, fontweight='bold')

plt.tight_layout()
plt.show()
```



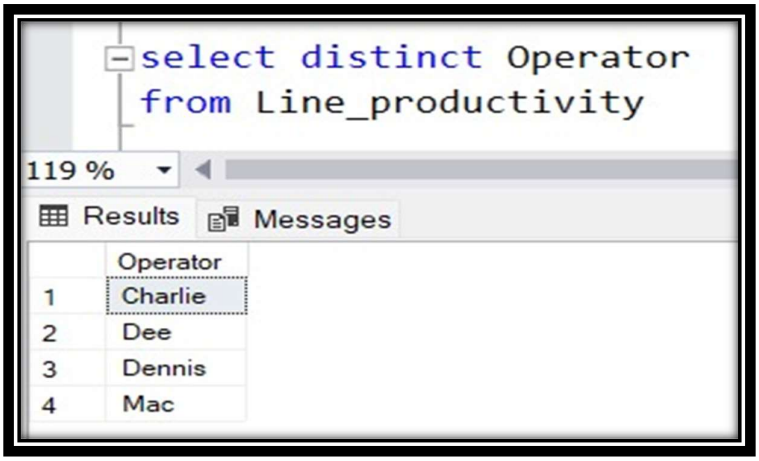
Q4) Which operators participated in the production process?

Excel:



Operators' Names	
Charlie	
Dee	
Dennis	
Mac	
Grand Total	4

SQL:



```
select distinct Operator
from Line_productivity
```

119 %

Results | Messages

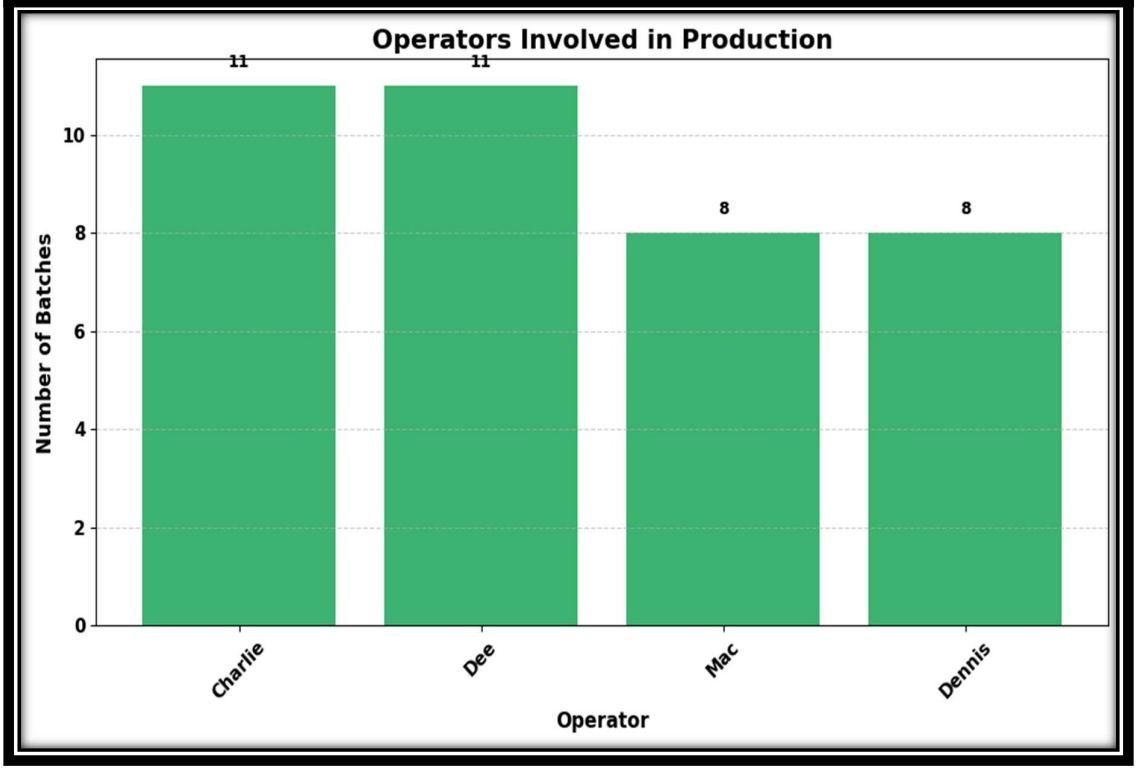
	Operator
1	Charlie
2	Dee
3	Dennis
4	Mac

Python:

```
#Q4) Which operators participated in the production process?

df = pd.read_excel(file_path, sheet_name="Transformed Line Productivity")

# Ensure column names are stripped of spaces
df.columns = df.columns.str.strip()

# Get the list of unique operators
unique_operators = df["Operator"].dropna().unique()

# Print result
print("Operators Involved in Production:")
print(unique_operators)
```

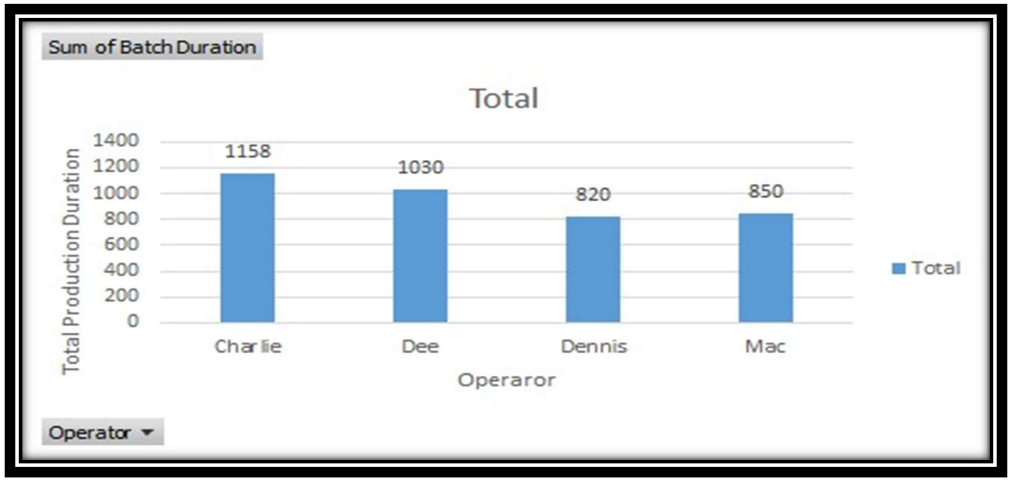
```
Operators Involved in Production:
['Mac' 'Charlie' 'Dee' 'Dennis']
```

```python
#Q4) Which operators participated in the production process?

# Count how many times each operator appears (participation level)
operator_counts = df["Operator"].value_counts()

# Plot bar chart
plt.figure(figsize=(10, 6))
bars = plt.bar(operator_counts.index.astype(str), operator_counts.values, color="mediumseagreen")

# Add labels and formatting
plt.title("Operators Involved in Production", fontsize=16, fontweight="bold")
plt.xlabel("Operator", fontsize=12, fontweight="bold")
plt.ylabel("Number of Batches", fontsize=12, fontweight="bold")
plt.xticks(rotation=45, fontsize=11, fontweight='bold')
plt.yticks(fontsize=11, fontweight='bold')
plt.grid(axis="y", linestyle="--", alpha=0.6)

# Add value labels
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.3, int(yval),
             ha='center', va='bottom', fontsize=10, fontweight='bold')

plt.tight_layout()
plt.show()
```

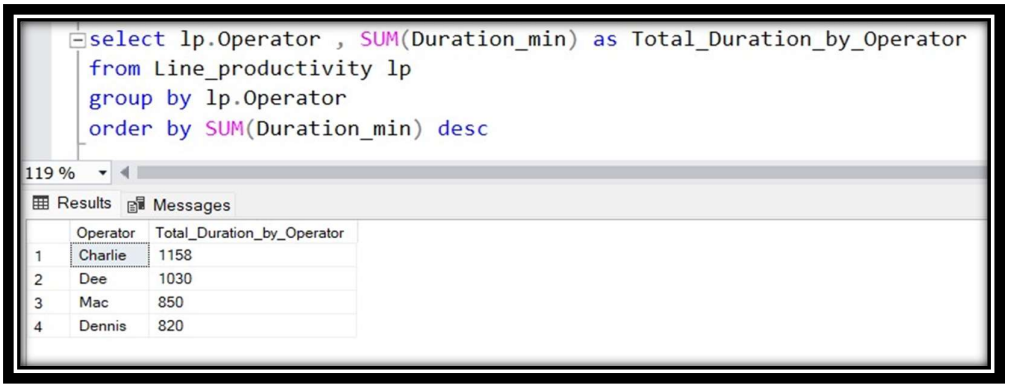


Q5) What is the total production duration for each operator?

Excel:



SQL:



```
select lp.Operator , SUM(Duration_min) as Total_Duration_by_Operator
from Line_productivity lp
group by lp.Operator
order by SUM(Duration_min) desc
```

119 %

Results | Messages

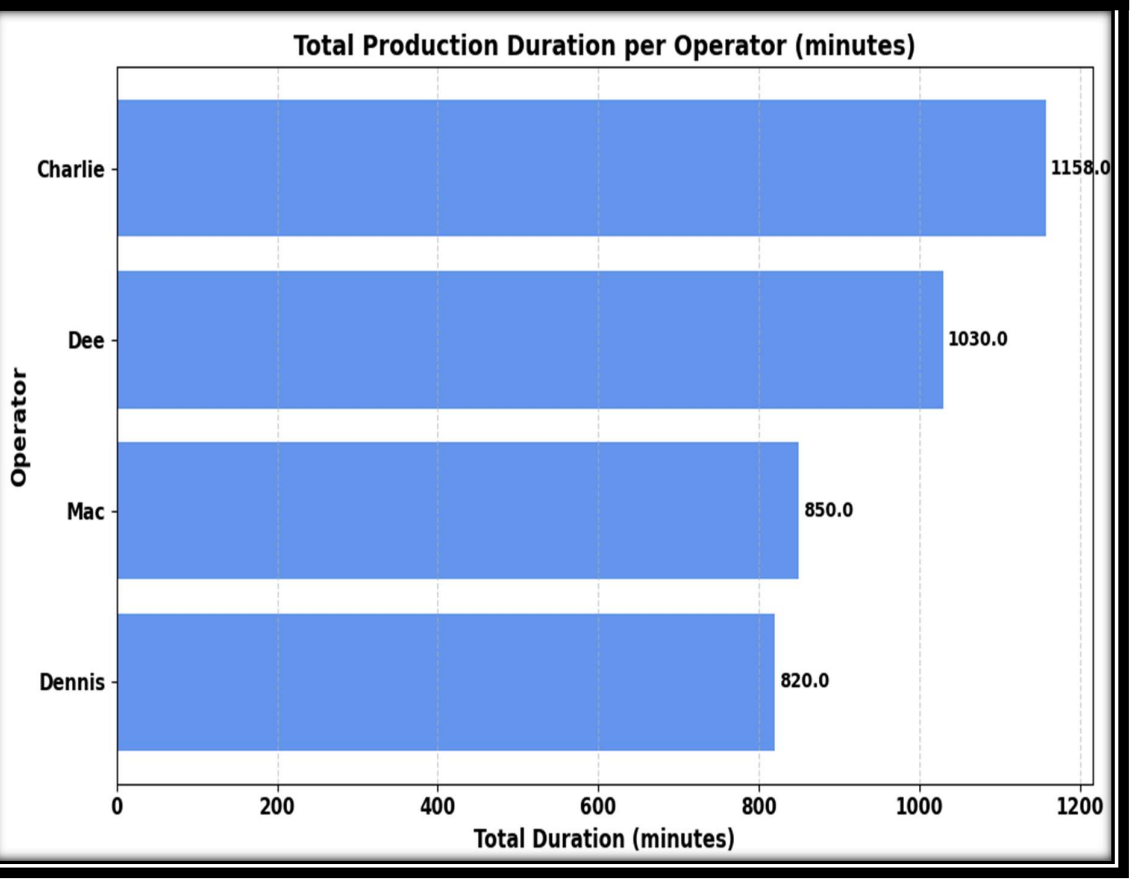
	Operator	Total_Duration_by_Operator
1	Charlie	1158
2	Dee	1030
3	Mac	850
4	Dennis	820

Python:

```python
#Q5) What is the total production duration for each operator?

import warnings

# Suppress pandas warnings
warnings.simplefilter(action="ignore", category=UserWarning)


# Ensure 'Start Time' and 'End Time' are in datetime format
df["Start Time"] = pd.to_datetime(df["Start Time"])
df["End Time"] = pd.to_datetime(df["End Time"])

# Calculate batch duration in minutes
df["Batch Duration"] = (df["End Time"] - df["Start Time"]).dt.total_seconds() / 60

# Calculate total production duration for each operator
operator_durations = df.groupby("Operator")["Batch Duration"].sum().reset_index()

# Display production duration per operator
print(operator_durations)
```

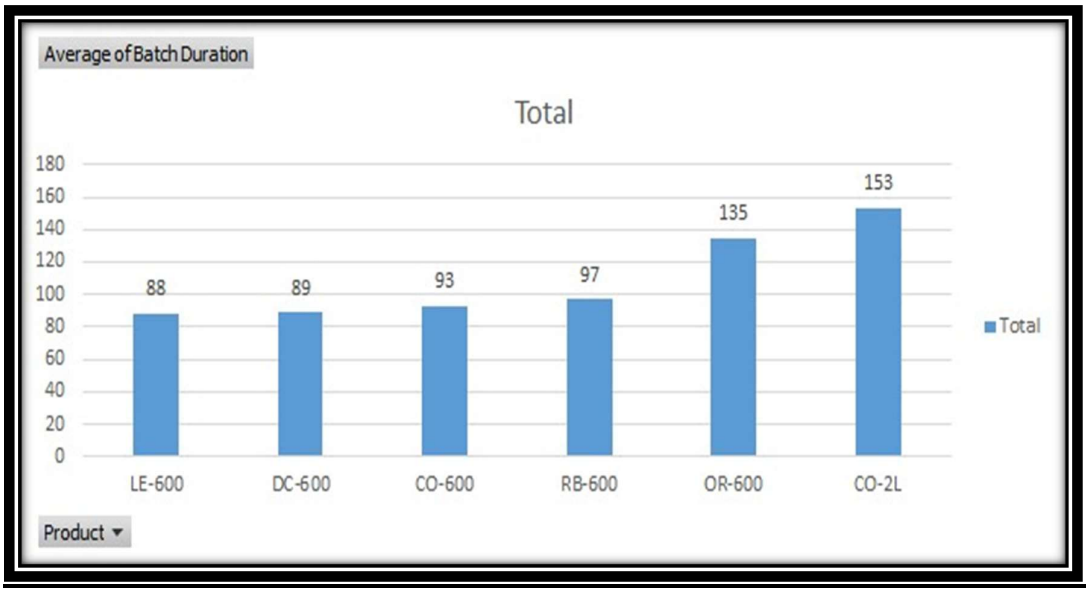
```
  Operator  Batch Duration
0  Charlie          1158.0
1      Dee          1030.0
2   Dennis           820.0
3      Mac           850.0
```

```python
#Q5) What is the total production duration for each operator?

# Sort operators by total duration (descending)
operator_durations = operator_durations.sort_values(by="Batch Duration", ascending=True)

# Plot horizontal bar chart
plt.figure(figsize=(10, 6))
bars = plt.barh(operator_durations["Operator"].astype(str),
                operator_durations["Batch Duration"], color="cornflowerblue")

# Labels and formatting
plt.title("Total Production Duration per Operator (minutes)", fontsize=14, fontweight="bold")
plt.xlabel("Total Duration (minutes)", fontsize=12, fontweight="bold")
plt.ylabel("Operator", fontsize=12, fontweight="bold")
plt.xticks(fontsize=11, fontweight="bold")
plt.yticks(fontsize=11, fontweight="bold")
plt.grid(axis="x", linestyle="--", alpha=0.6)

# Add duration labels to each bar
for bar in bars:
    width = bar.get_width()
    plt.text(width + 5, bar.get_y() + bar.get_height()/2,
             f"{width:.1f}", va='center', fontsize=10, fontweight="bold")

plt.tight_layout()
plt.show()
```



Q6) Which product has the highest average production time?

Excel:



SQL:

```
select lp.Product , avg(Duration_min) as Average_Production_Duration
from Line_productivity lp
group by lp.Product
order by avg(Duration_min) desc
```

119 %

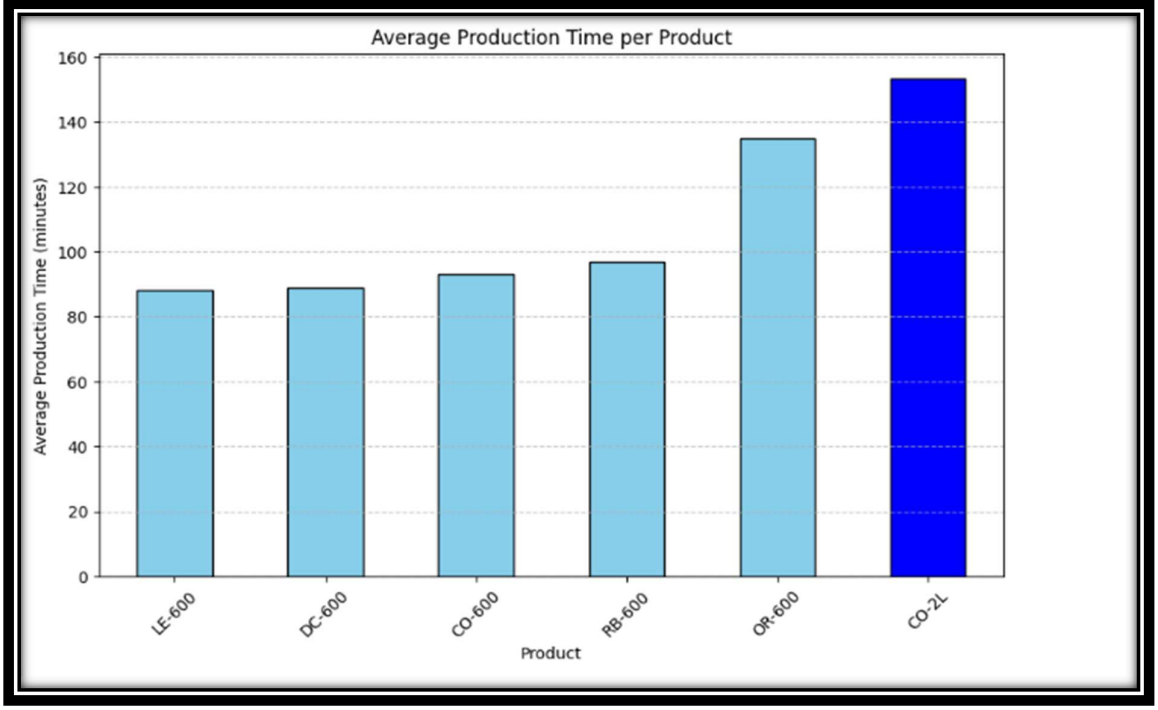
Results | Messages

	Product	Average_Production_Duration
1	CO-2L	153
2	OR-600	135
3	RB-600	96
4	CO-600	92
5	DC-600	88
6	LE-600	88

Python:

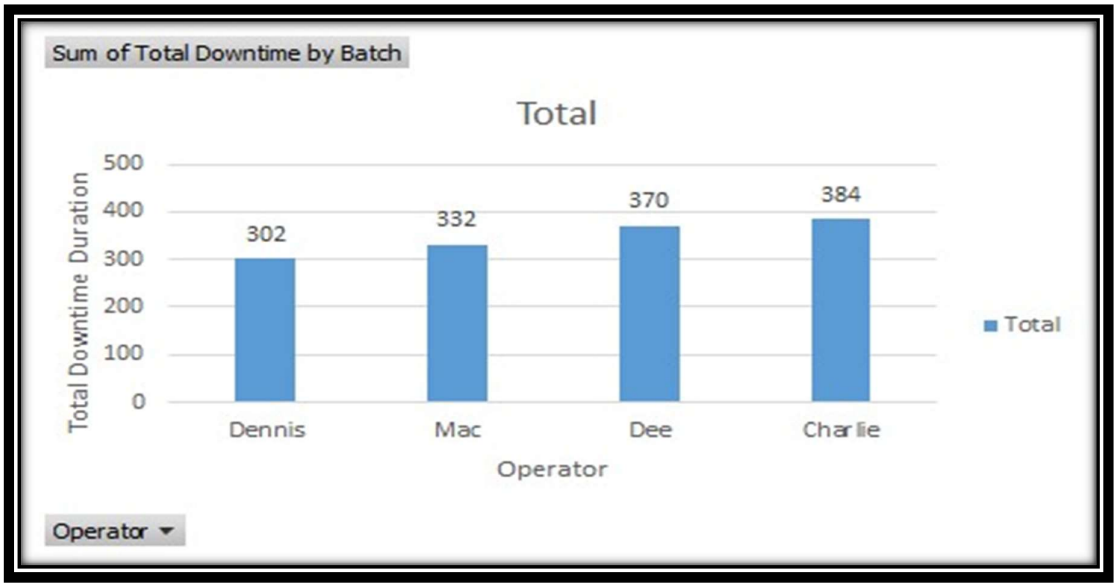
```python
#Q6) Which product has the highest average production time?

# Load data
df = pd.read_excel(file_path, sheet_name="Transformed Line Productivity")

# Calculate the average production time per product
avg_production_time = df.groupby("Product")["Batch Duration"].mean()

# Identify the product with the highest average production time
highest_avg_product = avg_production_time.idxmax()
highest_avg_time = avg_production_time.max()

print(f"The product with the highest average production time is {highest_avg_product} with {highest_avg_time:.2f} minutes.")

# Sort the values for better visualization
avg_production_time = avg_production_time.sort_values()

# Set bar colors: default skyblue, highlight the highest one in blue
colors = ["skyblue" if product != highest_avg_product else "blue" for product in avg_production_time.index]

# Plot the average production time per product
plt.figure(figsize=(10, 6))
avg_production_time.plot(kind="bar", color=colors, edgecolor="black")

# Add labels and title
plt.xlabel("Product")
plt.ylabel("Average Production Time (minutes)")
plt.title("Average Production Time per Product")
plt.xticks(rotation=45)
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Show the plot
plt.show()
```

```
The product with the highest average production time is CO-2L with 153.40 minutes.
```



Q7) Which operator is the most efficient in handling batches?

Excel:



SQL:

```sql
select lp.Operator , sum(ld.Batch_downtime_by_factor) as Total_Downtime
from Line_productivity lp join Line_downtime_factor1 ld
on lp.Batch = ld.Batch
group by lp.Operator
order by SUM(ld.Batch_downtime_by_factor) asc
```

119 %

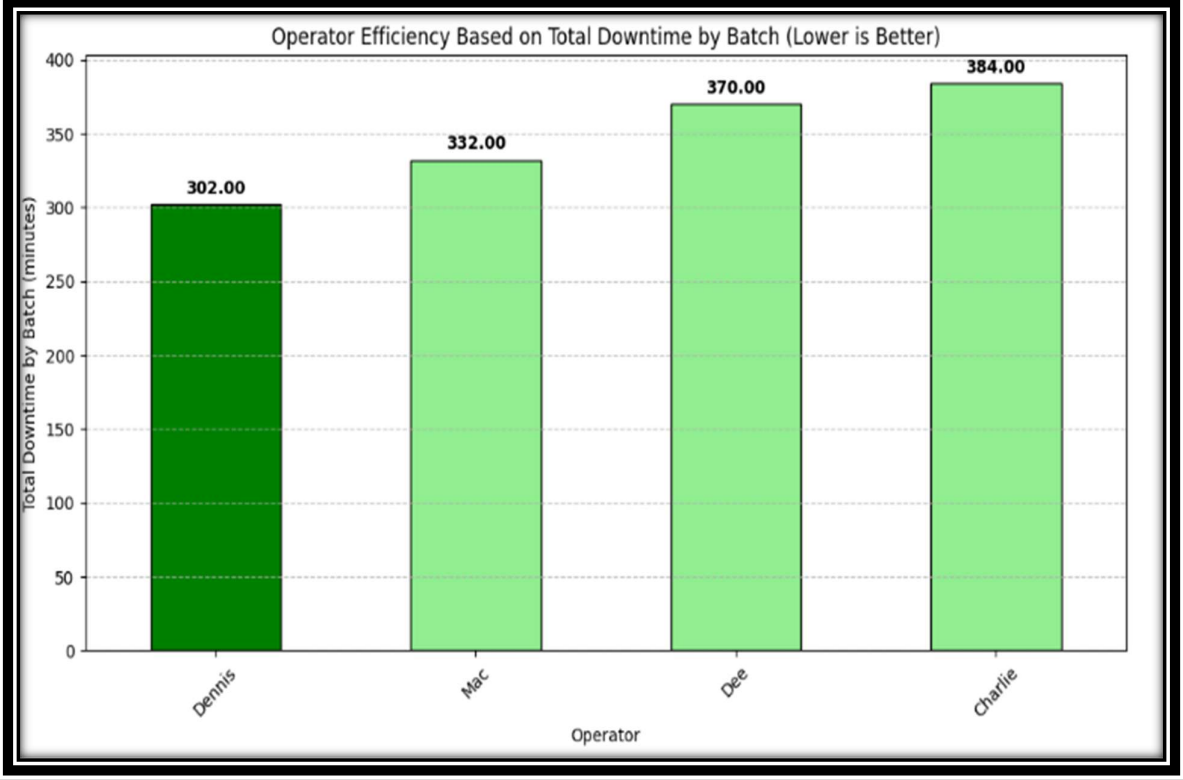
Results | Messages

	Operator	Total_Downtime
1	Dennis	302
2	Mac	332
3	Dee	370
4	Charlie	384

Python:

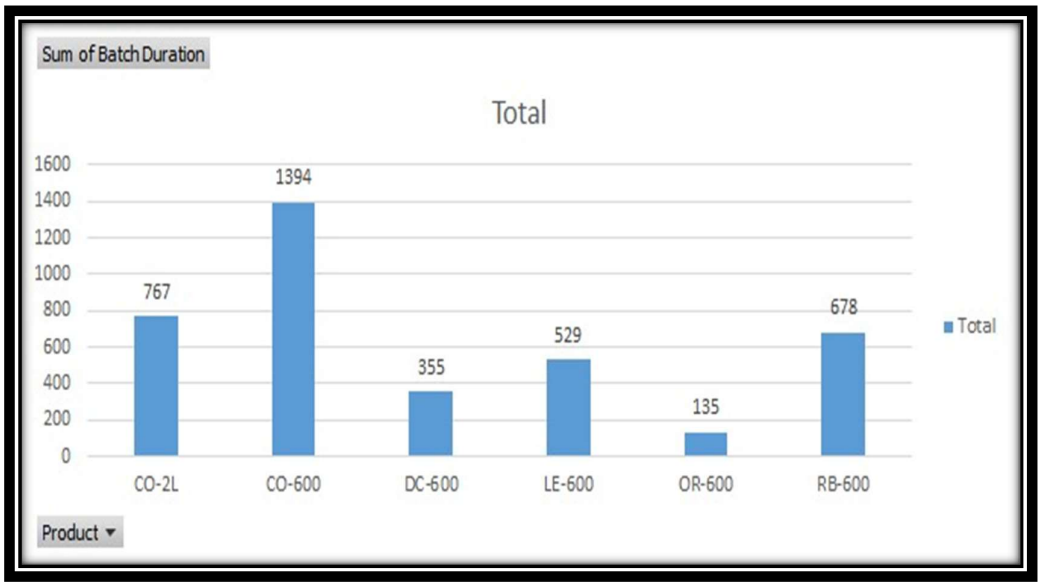
```python
#Q7) Which operator is the most efficient in handling batches?


# Load the dataset
df = pd.read_excel(file_path, sheet_name="Transformed Line Productivity")

# Ensure names are stripped of spaces
df["Operator"] = df["Operator"].astype(str).str.strip()

# Calculate the total downtime for each operator
operator_total_downtime = df.groupby("Operator")["Total Downtime by Batch"].sum()

# Sort operators by lowest total downtime (smallest first)
operator_total_downtime = operator_total_downtime.sort_values(ascending=True)

# Identify the most efficient operator (lowest total downtime)
most_efficient_operator = operator_total_downtime.idxmin()
lowest_total_downtime = operator_total_downtime.min()

print(f"The most efficient operator based on total downtime by batch is {most_efficient_operator} with a total downtime of {lowest_total_downtime:.2f} minutes.")

# Assign colors (highlight only the most efficient operator)
colors = ["lightgreen" if operator != most_efficient_operator else "green" for operator in operator_total_downtime.index]

# Plot total downtime per operator
plt.figure(figsize=(12, 6))
bars = operator_total_downtime.plot(kind="bar", color=colors, edgecolor="black")

# Add text labels on top of each bar
for bar in bars.patches:
    plt.text(bar.get_x() + bar.get_width() / 2,  # X position (center of the bar)
             bar.get_height() + 5,  # Y position (slightly above the bar)
             f"{bar.get_height():.2f}",  # Text (total downtime)
             ha="center", va="bottom", fontsize=10, fontweight="bold", color="black")

# Add labels and title
plt.xlabel("Operator")
plt.ylabel("Total Downtime by Batch (minutes)")
plt.title("Operator Efficiency Based on Total Downtime by Batch (Lower is Better)")
plt.xticks(rotation=45)
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Show the plot
plt.show()
```

```
The most efficient operator based on total downtime by batch is Dennis with a total downtime of 302.00 minutes.
```



Q8) What is the production duration for each product?

Excel:



SQL:

```
select lp.Product , sum(Duration_min) as Total_Production_Duration
from Line_productivity lp
group by lp.Product
order by sum(Duration_min) desc
```

119 %

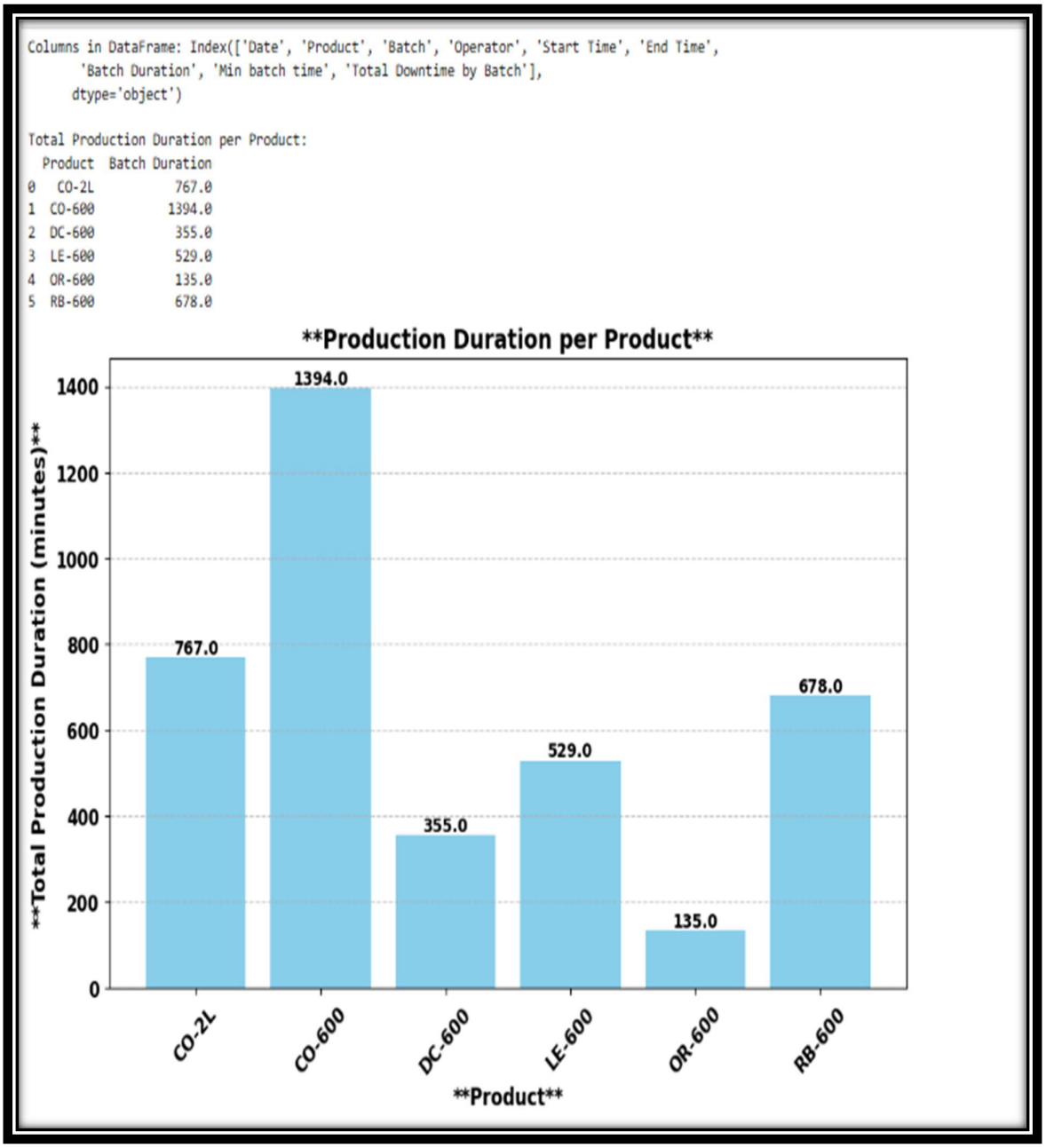
Results | Messages

	Product	Total_Production_Duration
1	CO-600	1394
2	CO-2L	767
3	RB-600	678
4	LE-600	529
5	DC-600	355
6	OR-600	135

Python:

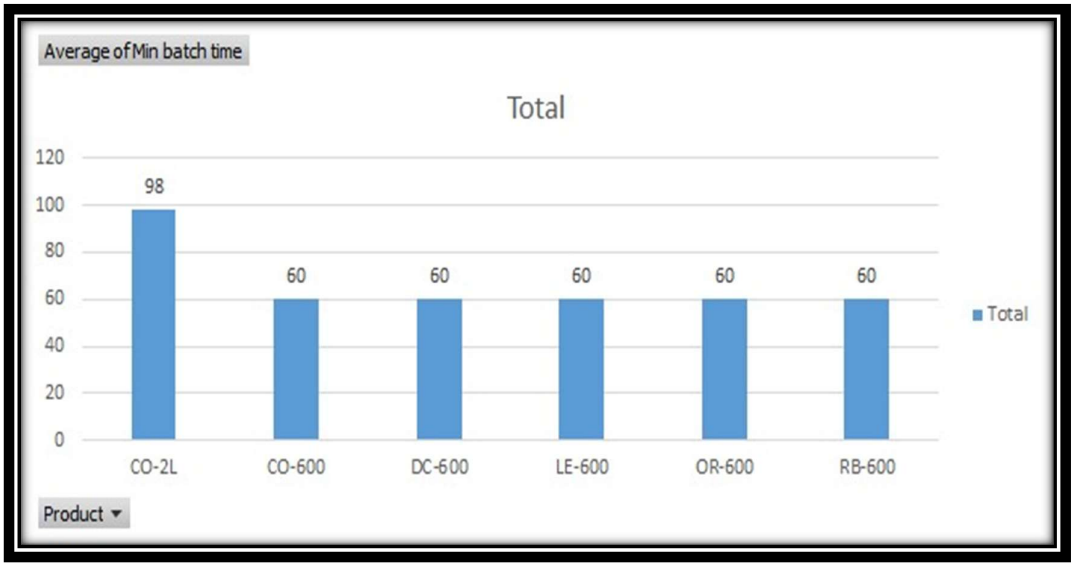
```python
# Q8) **What is the production duration for each product?**


# **Suppress pandas warnings**
warnings.simplefilter(action="ignore", category=UserWarning)

# **Display column names (for debugging)**
print("Columns in DataFrame:", df.columns)

# **Convert 'Start Time' and 'End Time' to datetime format safely**
df["Start Time"] = pd.to_datetime(df["Start Time"], errors="coerce", infer_datetime_format=True)
df["End Time"] = pd.to_datetime(df["End Time"], errors="coerce", infer_datetime_format=True)

# **Calculate batch duration in minutes**
df["Batch Duration"] = (df["End Time"] - df["Start Time"]).dt.total_seconds() / 60

# **Calculate total production duration per product**
product_durations = df.groupby("Product")["Batch Duration"].sum().reset_index()

# **Display results**
print("\nTotal Production Duration per Product:")
print(product_durations)

# **Visualize the results with bold labels and value labels**
plt.figure(figsize=(10, 6))
bars = plt.bar(product_durations["Product"], product_durations["Batch Duration"], color="skyblue")

# **Bold labels and title**
plt.xlabel("**Product**", fontsize=12, fontweight="bold")
plt.ylabel("**Total Production Duration (minutes)**", fontsize=12, fontweight="bold")
plt.title("**Production Duration per Product**", fontsize=14, fontweight="bold")
plt.xticks(rotation=45, fontsize=11, fontweight="bold")  # Rotate and bold x-axis labels
plt.yticks(fontsize=11, fontweight="bold")  # Bold y-axis labels
plt.grid(axis="y", linestyle="--", alpha=0.7)

# **Add labels on top of bars**
for bar in bars:
    height = bar.get_height()
    plt.text(
        bar.get_x() + bar.get_width() / 2,  # X-coordinate
        height,  # Y-coordinate (bar height)
        f"{height:.1f}",  # Display value with 1 decimal place
        ha="center",  # Center alignment
        va="bottom",  # Position slightly above the bar
        fontsize=10,
        fontweight="bold",
        color="black",
    )

# **Show the plot**
plt.show()
```

```
Columns in DataFrame: Index(['Date', 'Product', 'Batch', 'Operator', 'Start Time', 'End Time',
       'Batch Duration', 'Min batch time', 'Total Downtime by Batch'],
      dtype='object')

Total Production Duration per Product:
  Product  Batch Duration
0   CO-2L           767.0
1  CO-600          1394.0
2  DC-600           355.0
3  LE-600           529.0
4  OR-600           135.0
5  RB-600           678.0
```

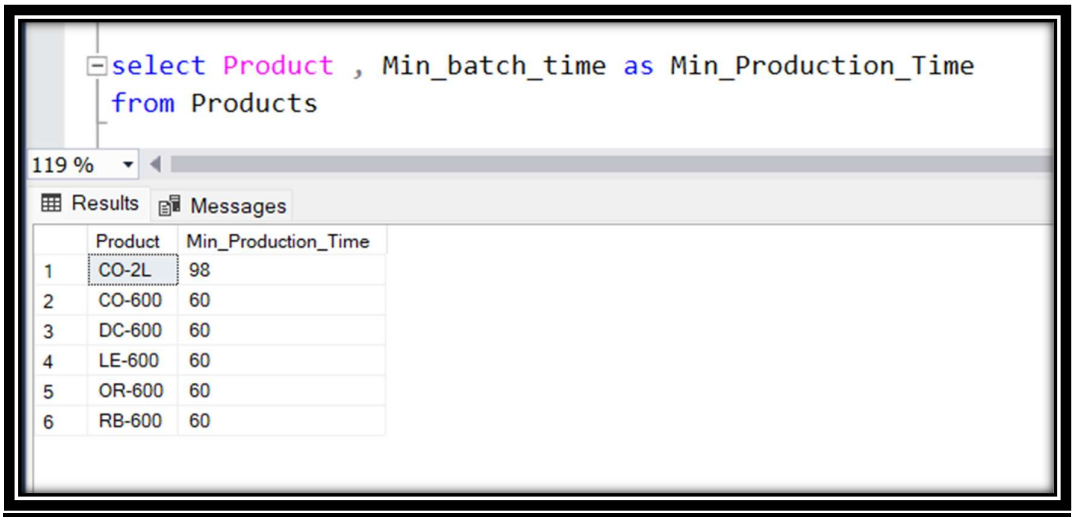


Q9) What is the minimum batch duration for each product?

Excel:



SQL:



```
select Product , Min_batch_time as Min_Production_Time
from Products
```

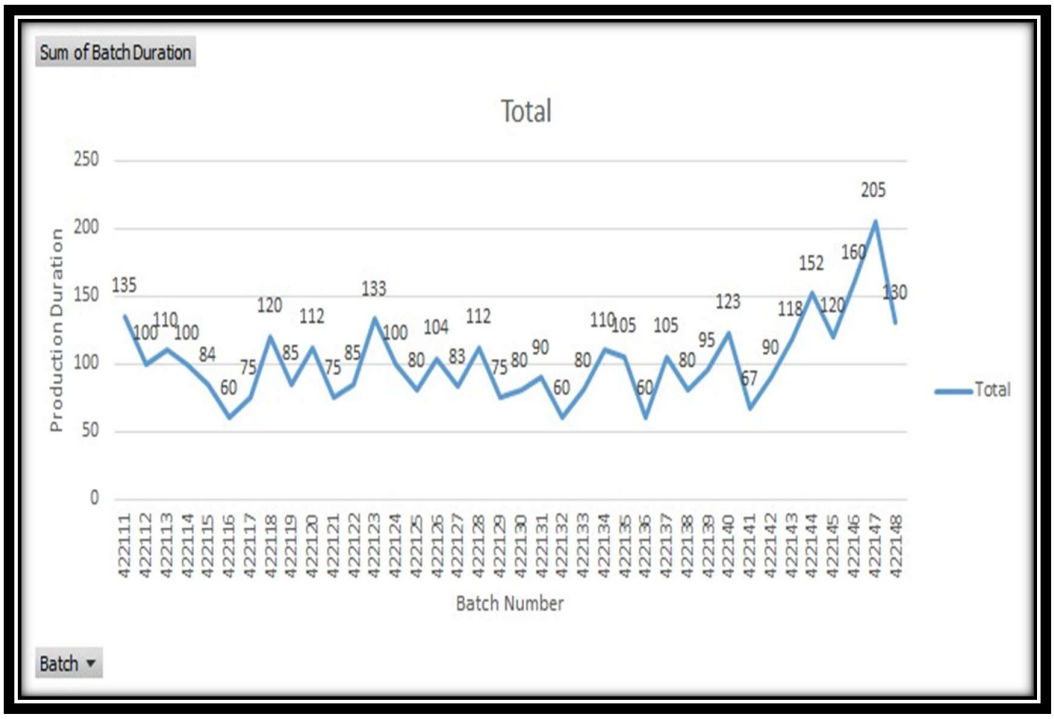
119 %

Results | Messages

	Product	Min_Production_Time
1	CO-2L	98
2	CO-600	60
3	DC-600	60
4	LE-600	60
5	OR-600	60
6	RB-600	60

Q10) What is the duration of each batch?

Excel:



SQL:

```
select lp.Batch , Duration_min
from Line_productivity lp
```

131 %

Results | Messages

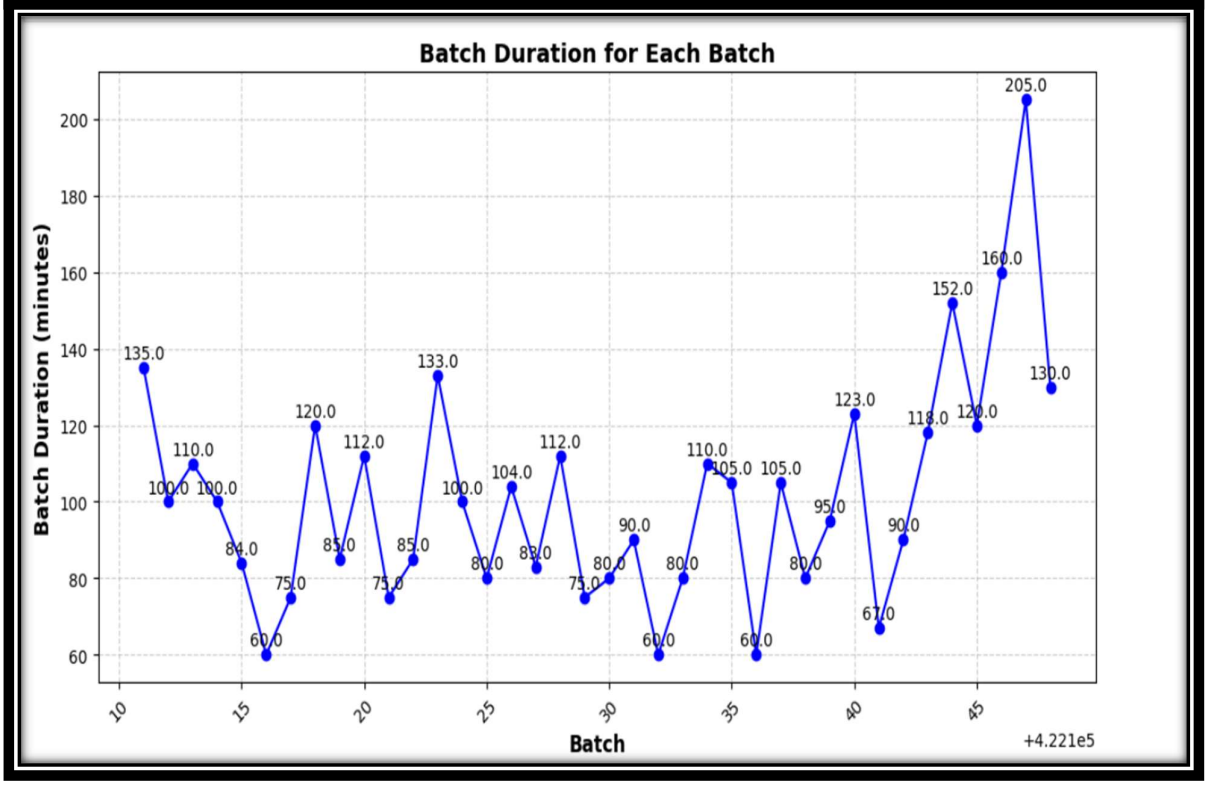
	Batch	Duration_min
1	422111	135
2	422112	100
3	422113	110
4	422114	100
5	422115	84
6	422116	60
7	422117	75
8	422118	120
9	422119	85
10	422120	112
11	422121	75
12	422122	85
13	422123	133
14	422124	100
15	422125	80
16	422126	104
17	422127	83
18	422128	112
19	422129	75
20	422130	80
21	422131	90
22	422132	60
23	422133	80
24	422134	110
25	422135	105
26	422136	60
27	422137	105
28	422138	80
29	422139	95
30	422140	123
31	422141	67
32	422142	90
33	422143	118
34	422144	152
35	422145	120
36	422146	160
37	422147	205
38	422148	130

Python:

```python
#Q10) What is the duration of each batch?

# Convert "Start Time" and "End Time" to datetime (without date)
df["Start Time"] = pd.to_datetime(df["Start Time"], format="%H:%M:%S", errors="coerce")
df["End Time"] = pd.to_datetime(df["End Time"], format="%H:%M:%S", errors="coerce")

# Ensure no NaT values exist
df = df.dropna(subset=["Start Time", "End Time"])

# Assign a fixed date (September 2, 2024)
base_date = pd.Timestamp("2024-09-02")

# Add time component as a timedelta
df["Start Time"] = base_date + df["Start Time"].dt.hour.astype("timedelta64[h]") + df["Start Time"].dt.minute.astype("timedelta64[m]") + df["Start Tim
df["End Time"] = base_date + df["End Time"].dt.hour.astype("timedelta64[h]") + df["End Time"].dt.minute.astype("timedelta64[m]") + df["End Time"].dt.s

# Fix incorrect Start Time and End Time for Batch 422148
df.loc[df["Batch"] == 422148, "Start Time"] = pd.Timestamp("2024-09-02 22:55:00")
df.loc[df["Batch"] == 422148, "End Time"] = pd.Timestamp("2024-09-03 01:05:00")

# Recalculate Batch Duration in minutes
df["Batch Duration"] = (df["End Time"] - df["Start Time"]).dt.total_seconds() / 60

# Save the updated file
df.to_excel("Updated_Manufacturing.xlsx", index=False)

# Display results
batch_durations = df[["Batch", "Batch Duration"]]
print(batch_durations)
```

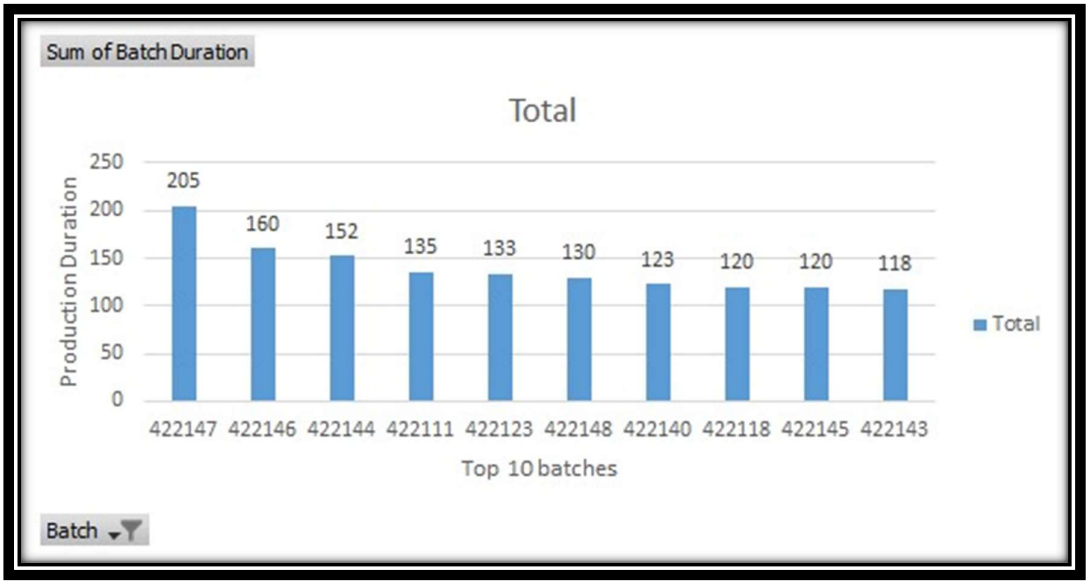
	Batch	Batch Duration
0	422111	135.0
1	422112	100.0
2	422113	110.0
3	422114	100.0
4	422115	84.0
5	422116	60.0
6	422117	75.0
7	422118	120.0
8	422119	85.0
9	422120	112.0
10	422121	75.0
11	422122	85.0
12	422123	133.0
13	422124	100.0
14	422125	80.0
15	422126	104.0
16	422127	83.0
17	422128	112.0
18	422129	75.0
19	422130	80.0
20	422131	90.0
21	422132	60.0
22	422133	80.0
23	422134	110.0
24	422135	105.0
25	422136	60.0
26	422137	105.0
27	422138	80.0
28	422139	95.0
29	422140	123.0
30	422141	67.0
31	422142	90.0
32	422143	118.0
33	422144	152.0
34	422145	120.0
35	422146	160.0
36	422147	205.0
37	422148	130.0

```python
#Q10) What is the duration of each batch?

# Convert "Start Time" and "End Time" to datetime format
df["Start Time"] = pd.to_datetime(df["Start Time"], errors="coerce")
df["End Time"] = pd.to_datetime(df["End Time"], errors="coerce")

# Calculate Batch Duration in minutes
df["Batch Duration"] = (df["End Time"] - df["Start Time"]).dt.total_seconds() / 60

# Sort data by Batch for visualization
df_sorted = df.sort_values(by="Batch")

# Plot line chart
plt.figure(figsize=(12, 6))
plt.plot(df_sorted["Batch"], df_sorted["Batch Duration"], marker="o", linestyle="-", color="b")

# Labeling
plt.xlabel("Batch", fontsize=12, fontweight="bold")
plt.ylabel("Batch Duration (minutes)", fontsize=12, fontweight="bold")
plt.title("Batch Duration for Each Batch", fontsize=14, fontweight="bold")

# Display labels for each point
for i, txt in enumerate(df_sorted["Batch Duration"]):
    plt.annotate(f"{txt:.1f}", (df_sorted["Batch"].iloc[i], df_sorted["Batch Duration"].iloc[i]),
                 textcoords="offset points", xytext=(0,5), ha="center", fontsize=10)

plt.xticks(rotation=45)
plt.grid(True, linestyle="--", alpha=0.6)
plt.show()
```



Q11) Which Top 10 Batches with Most Batch Duration ?

Excel:



SQL:

```
select top 10 lp.Batch , Duration_min
from Line_productivity lp
order by Duration_min desc
```

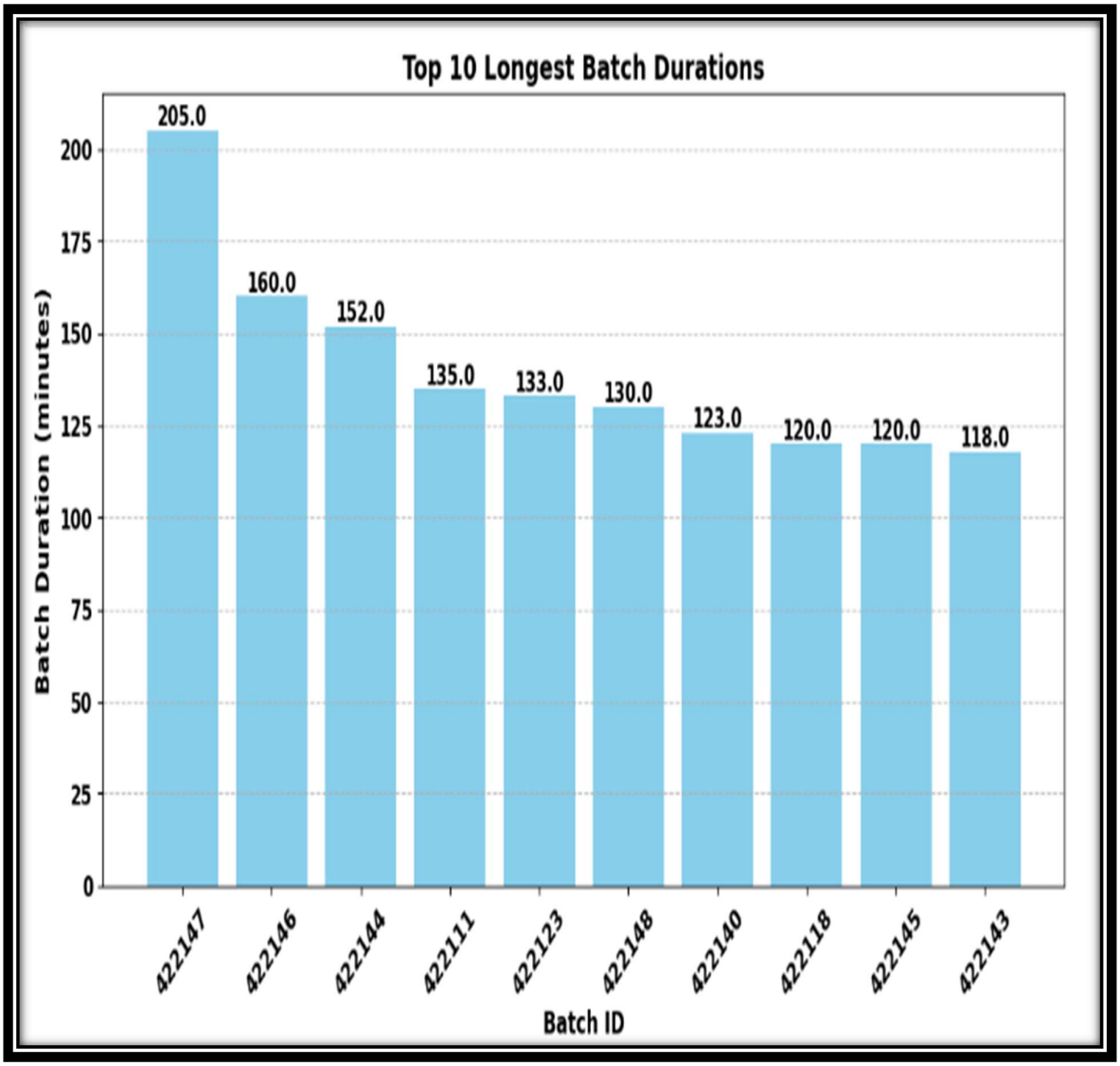
119 %

Results | Messages

	Batch	Duration_min
1	422147	205
2	422146	160
3	422144	152
4	422111	135
5	422123	133
6	422148	130
7	422140	123
8	422118	120
9	422145	120
10	422143	118

Python:

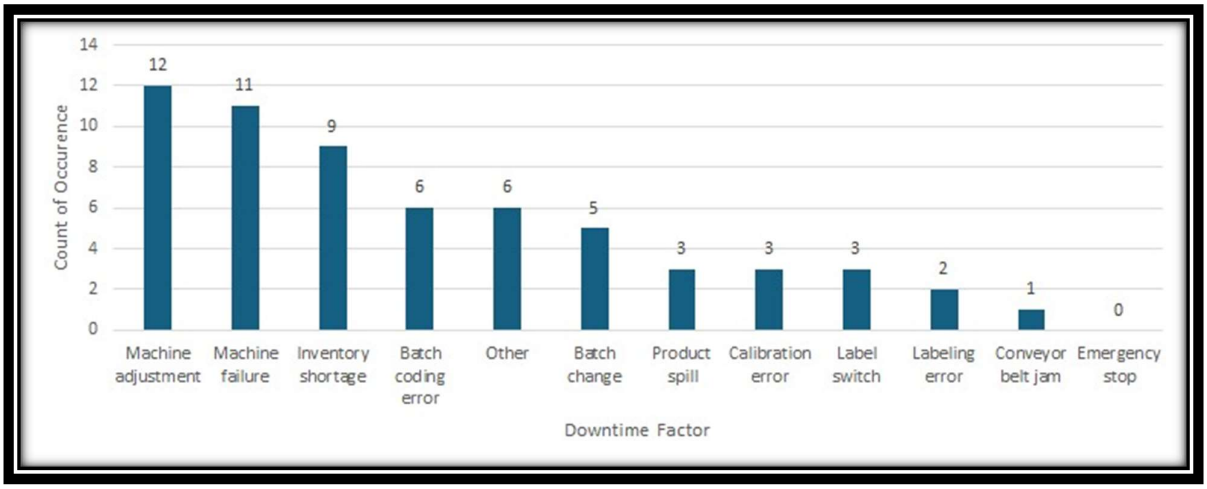
```python
#Q11) Which Top 10 Batches with Most Batch Duration ?

# Convert "Start Time" and "End Time" to datetime format
df["Start Time"] = pd.to_datetime(df["Start Time"], format="%H:%M:%S", errors="coerce")
df["End Time"] = pd.to_datetime(df["End Time"], format="%H:%M:%S", errors="coerce")

# Ensure no NaT values exist
df = df.dropna(subset=["Start Time", "End Time"])

# Assign a fixed date (September 2, 2024)
base_date = pd.Timestamp("2024-09-02")

# Add time component as a timedelta
df["Start Time"] = base_date + df["Start Time"].dt.hour.astype("timedelta64[h]") + \
                   df["Start Time"].dt.minute.astype("timedelta64[m]") + \
                   df["Start Time"].dt.second.astype("timedelta64[s]")

df["End Time"] = base_date + df["End Time"].dt.hour.astype("timedelta64[h]") + \
                 df["End Time"].dt.minute.astype("timedelta64[m]") + \
                 df["End Time"].dt.second.astype("timedelta64[s]")

# Fix incorrect Start Time and End Time for Batch 422148
df.loc[df["Batch"] == 422148, "Start Time"] = pd.Timestamp("2024-09-02 22:55:00")
df.loc[df["Batch"] == 422148, "End Time"] = pd.Timestamp("2024-09-03 01:05:00")

# Recalculate Batch Duration in minutes
df["Batch Duration"] = (df["End Time"] - df["Start Time"]).dt.total_seconds() / 60

# Select top 10 longest batch durations for visualization
top_batches = df.nlargest(10, "Batch Duration")

# Plot bar chart
plt.figure(figsize=(12, 6))
bars = plt.bar(top_batches["Batch"].astype(str), top_batches["Batch Duration"], color="skyblue")

# Labels and title
plt.xlabel("Batch ID", fontsize=12, fontweight="bold")
plt.ylabel("Batch Duration (minutes)", fontsize=12, fontweight="bold")
plt.title("Top 10 Longest Batch Durations", fontsize=14, fontweight="bold")
plt.xticks(rotation=45, fontsize=11, fontweight="bold")
plt.yticks(fontsize=11, fontweight="bold")

# Display values on top of bars
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height, f"{height:.1f}",
             ha="center", va="bottom", fontsize=11, fontweight="bold")

plt.grid(axis="y", linestyle="--", alpha=0.7)

# Show plot
plt.show()
```


Top 10 Longest Batch Durations
Batch Duration (minutes)
0
25
50
75
100
125
150
175
200
205.0
160.0
152.0
135.0
133.0
130.0
123.0
120.0
120.0
118.0
422147
422146
422144
422111
422123
422148
422140
422118
422145
422143
Batch ID

Downtime Analysis

Q12) What are the most common downtime factors?

Excel:



SQL:

```
select df.Description  ,count( ld.Factor) as most_common_downtime_factor
from Line_downtime_factor1 ld join Downtime_factors df
on ld.Factor=df.Factor
group by df.Description
order by most_common_downtime_factor desc
```

119 %

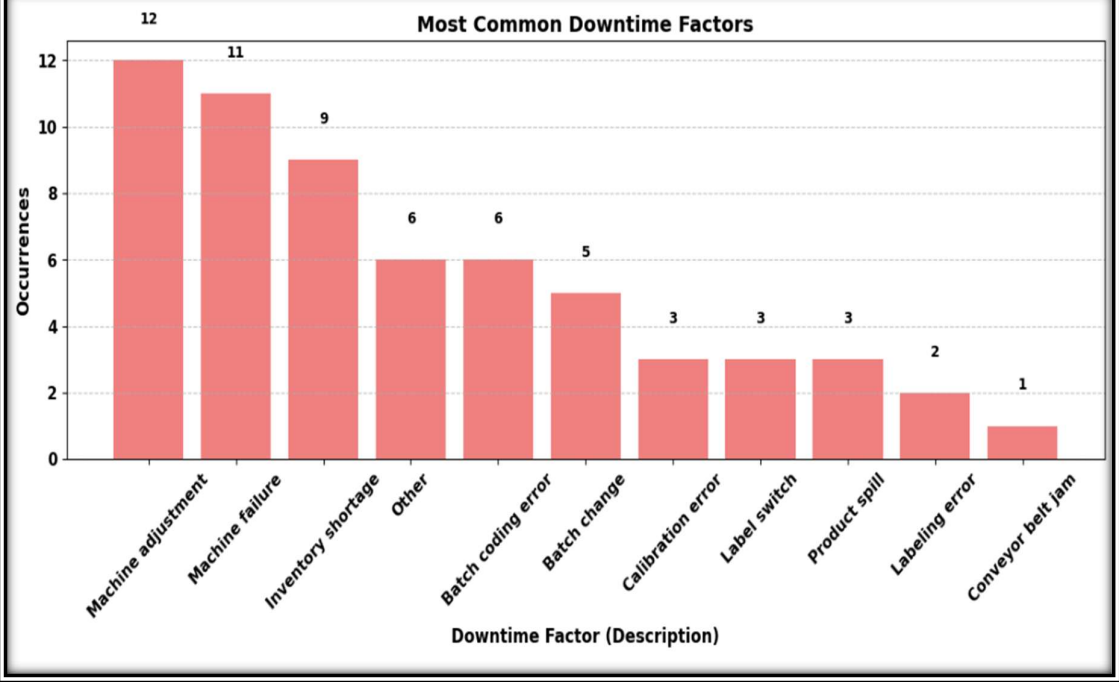
Results | Messages

	Description	most_common_downtime_factor
1	Machine adjustment	12
2	Machine failure	11
3	Inventory shortage	9
4	Batch coding error	6
5	Other	6
6	Batch change	5
7	Product spill	3
8	Calibration error	3
9	Label switch	3
10	Labeling error	2
11	Conveyor belt jam	1

```python
#Q12) What are the most common downtime factors?

df = pd.read_excel(file_path, sheet_name="Transformed Line downtime")

# Count occurrences of each downtime factor
common_downtime_factors = df["Downtime Factor"].value_counts()

# Get the most common downtime factor(s)
most_common_factor = common_downtime_factors.idxmax()
most_common_count = common_downtime_factors.max()

print(f"The most common downtime factor is '{most_common_factor}' with {most_common_count} occurrences.")
```

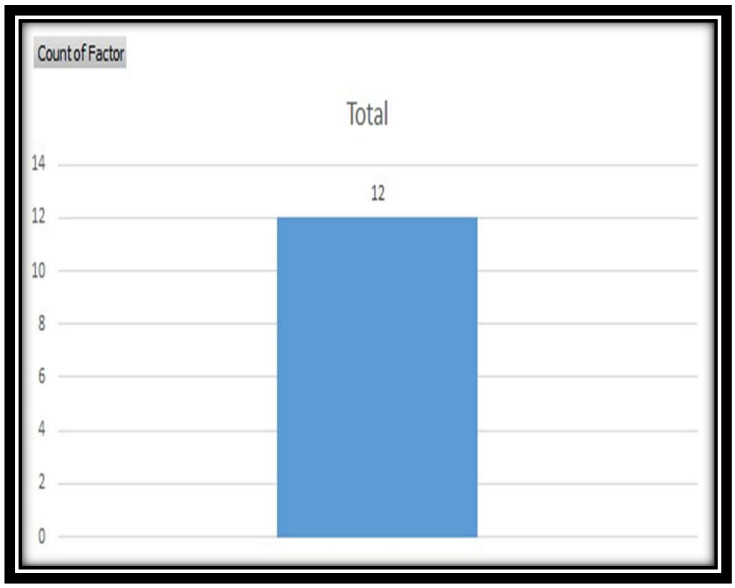
```
The most common downtime factor is '6' with 12 occurrences.
```

```python
# Q12) **What are the most common downtime factors?**

# Visualize the most common downtime factors
plt.figure(figsize=(12, 6))
bars = plt.bar(common_downtime_factors.index, common_downtime_factors.values, color="lightcoral")

# Bold labels and title
plt.xlabel("Downtime Factor (Description)", fontsize=12, fontweight="bold")
plt.ylabel("Occurrences", fontsize=12, fontweight="bold")
plt.title("Most Common Downtime Factors", fontsize=14, fontweight="bold")
plt.xticks(rotation=45, fontsize=11, fontweight="bold")  # Rotate and bold x-axis labels
plt.yticks(fontsize=11, fontweight="bold")  # Bold y-axis labels
plt.grid(axis="y", linestyle="--", alpha=0.7)

# ✅ Add only the count as label (remove description text)
for bar in bars:
    height = bar.get_height()
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        height + 1,
        f"{int(height)}",  # Only show the count
        ha="center",
        va="bottom",
        fontsize=10,
        fontweight="bold",
        color="black"
    )

# Show the plot
plt.tight_layout()
plt.show()
```

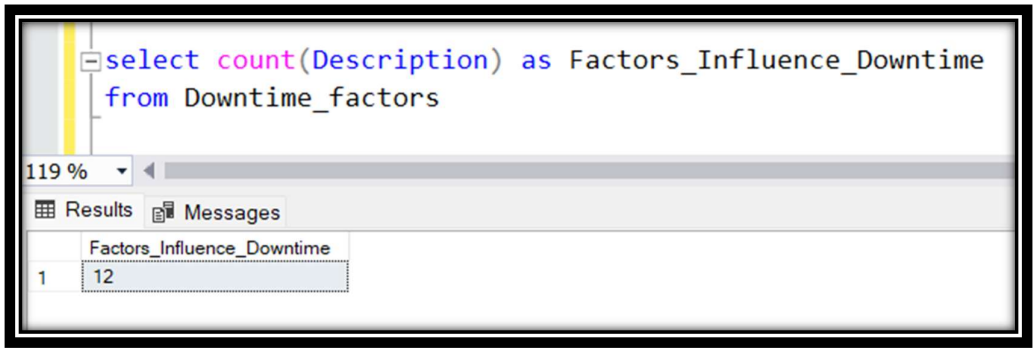


Q13) How many factors influence downtime?

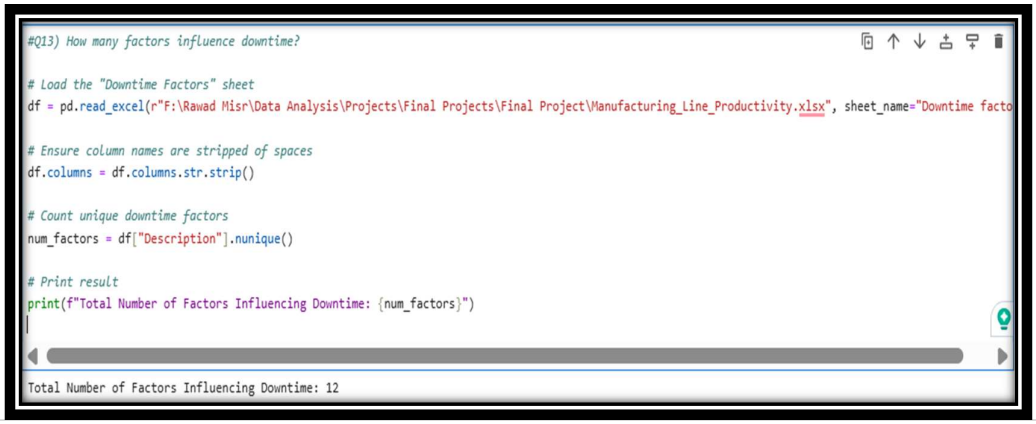
Excel:



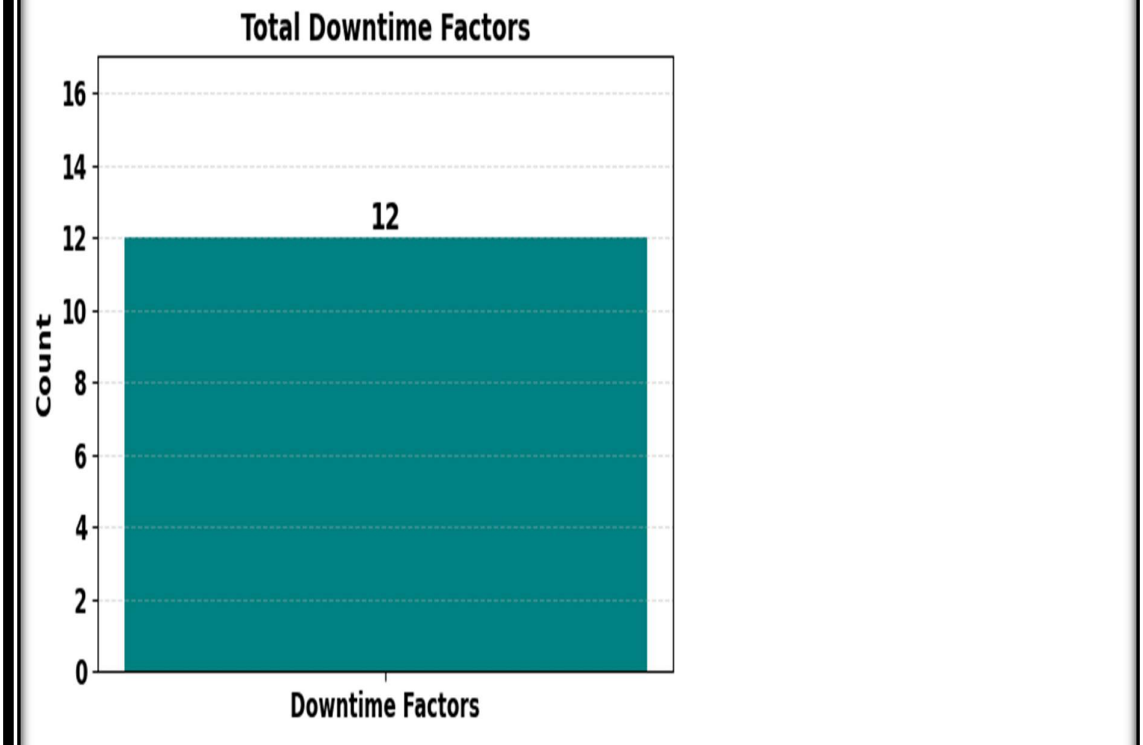
SQL:



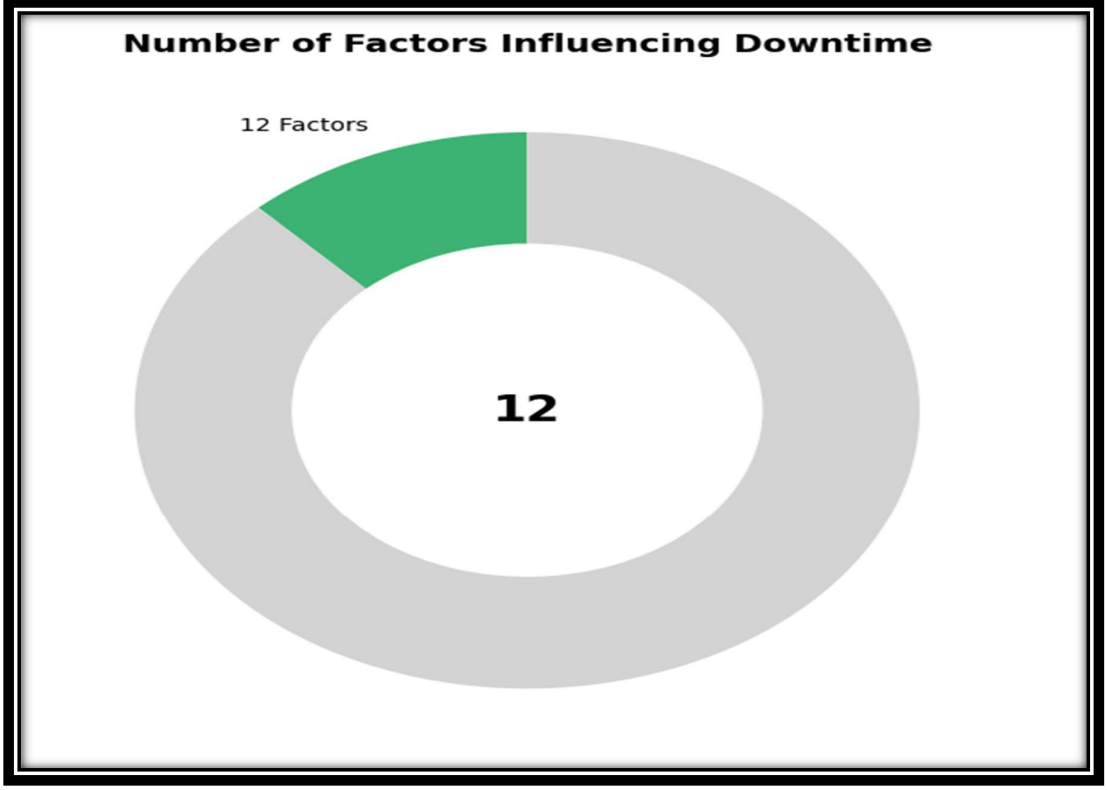
Python:



```python
#Q13) How many factors influence downtime?

# Load data
df = pd.read_excel(r"F:\Rawad Misr\Data Analysis\Projects\Final Projects\Final Project\Manufacturing_Line_Productivity.xlsx", sheet_name="Downtime facto
df.columns = df.columns.str.strip()

# Count unique factors
num_factors = df["Description"].nunique()

# Bar Chart
plt.figure(figsize=(6, 4))
plt.bar(["Downtime Factors"], [num_factors], color="teal")
plt.title("Total Downtime Factors", fontsize=14, fontweight="bold")
plt.ylabel("Count", fontsize=12, fontweight="bold")
plt.xticks(fontsize=12, fontweight="bold")
plt.yticks(fontsize=12, fontweight="bold")

# Annotate the bar with the count
plt.text(0, num_factors + 0.2, str(num_factors), ha="center", fontsize=14, fontweight="bold", color="black")

plt.ylim(0, num_factors + 5)
plt.grid(axis="y", linestyle="--", alpha=0.4)
plt.tight_layout()
plt.show()
```



```python
#Q13) How many factors influence downtime?

# Donut chart
plt.figure(figsize=(6, 6))
plt.pie([num_factors, 100 - num_factors], labels=[f"{num_factors} Factors", ""], startangle=90,
        colors=["mediumseagreen", "lightgrey"], wedgeprops={'width': 0.4})

plt.title("Number of Factors Influencing Downtime", fontsize=14, fontweight="bold")
plt.text(0, 0, str(num_factors), ha='center', va='center', fontsize=20, fontweight="bold")
plt.tight_layout()
plt.show()
```



Q14) How many factors influence downtime, and what are they?

Excel:

Downtime Factors	
Batch change	
Batch coding error	
Calibration error	
Conveyor belt jam	
Emergency stop	
Inventory shortage	
Label switch	
Labeling error	
Machine adjustment	
Machine failure	
Other	
Product spill	
Grand Total	12

SQL:

```
select Description
from Downtime_factors
```

119 %

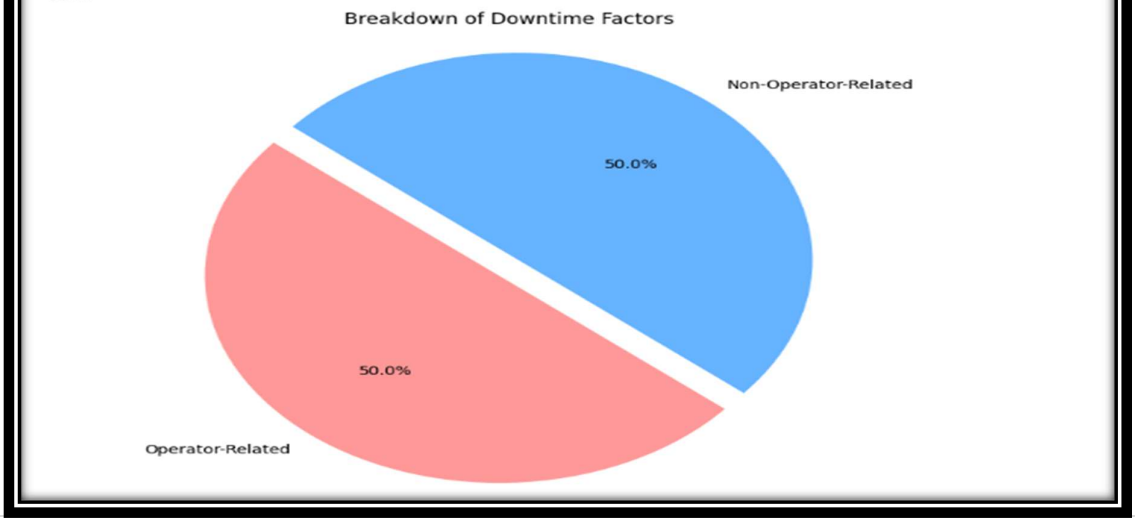
Results | Messages

	Description
1	Emergency stop
2	Batch change
3	Labeling error
4	Inventory shortage
5	Product spill
6	Machine adjustment
7	Machine failure
8	Batch coding error
9	Conveyor belt jam
10	Calibration error
11	Label switch
12	Other

Python:

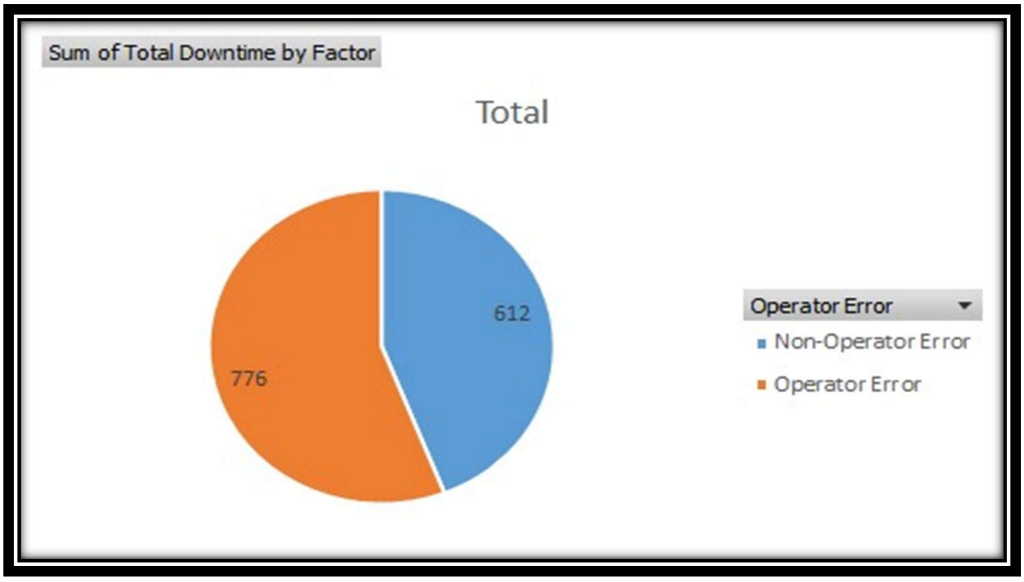
```python
#Q14) How many factors influence downtime, and what are they?

# Ensure column names are stripped of spaces
df.columns = df.columns.str.strip()

# Classify downtime factors
operator_factors = df[df["Operator Error"] == "Yes"]["Description"].unique()
non_operator_factors = df[df["Operator Error"] == "No"]["Description"].unique()

# Count unique factors in each category
num_operator_factors = len(operator_factors)
num_non_operator_factors = len(non_operator_factors)

# Print results
print(f"📌 Total Number of Factors Influencing Downtime: {num_operator_factors + num_non_operator_factors}\n")

print("✅ Operator-Related Downtime Factors:")
print(f"Total: {num_operator_factors}")
for factor in operator_factors:
    print("-", factor)

print("\n❌ Non-Operator-Related Downtime Factors:")
print(f"Total: {num_non_operator_factors}")
for factor in non_operator_factors:
    print("-", factor)

# Visualization
labels = ['Operator-Related', 'Non-Operator-Related']
sizes = [num_operator_factors, num_non_operator_factors]
colors = ['#ff9999', '#66b3ff']
explode = (0.1, 0)  # Highlight the operator-related section
```

```
📌 Total Number of Factors Influencing Downtime: 12

✅ Operator-Related Downtime Factors:
Total: 6
- Batch change
- Product spill
- Machine adjustment
- Batch coding error
- Calibration error
- Label switch

❌ Non-Operator-Related Downtime Factors:
Total: 6
- Emergency stop
- Labeling error
- Inventory shortage
- Machine failure
- Conveyor belt jam
- Other
```



Q15) How much downtime is caused by operator errors vs. other causes in Minutes ?

Excel:



SQL:

```
select 'Total Downtime' as " " ,sum(Batch_change+Product_spill+Machine_adjustment+Batch_coding_error+
                                    Calibration_error+Label_switch) as Operator
from Line_downtime_factor

select 'Total Downtime' as " " ,sum(Emergency_stop+Labeling_error+Inventory_shortage+Machine_failure+
                                    Conveyo_belt_jam+Other) as Non_operator
from Line_downtime_factor
```

119 %

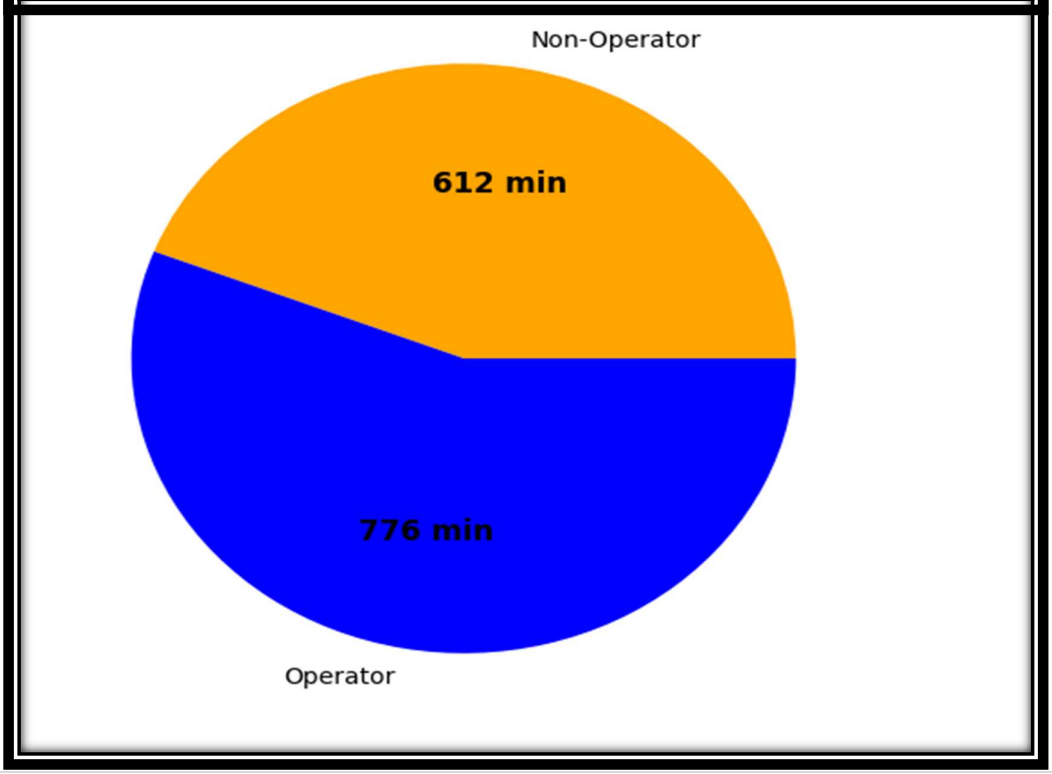
Results | Messages

		Operator
1	Total Downtime	776

		Non_operator
1	Total Downtime	612

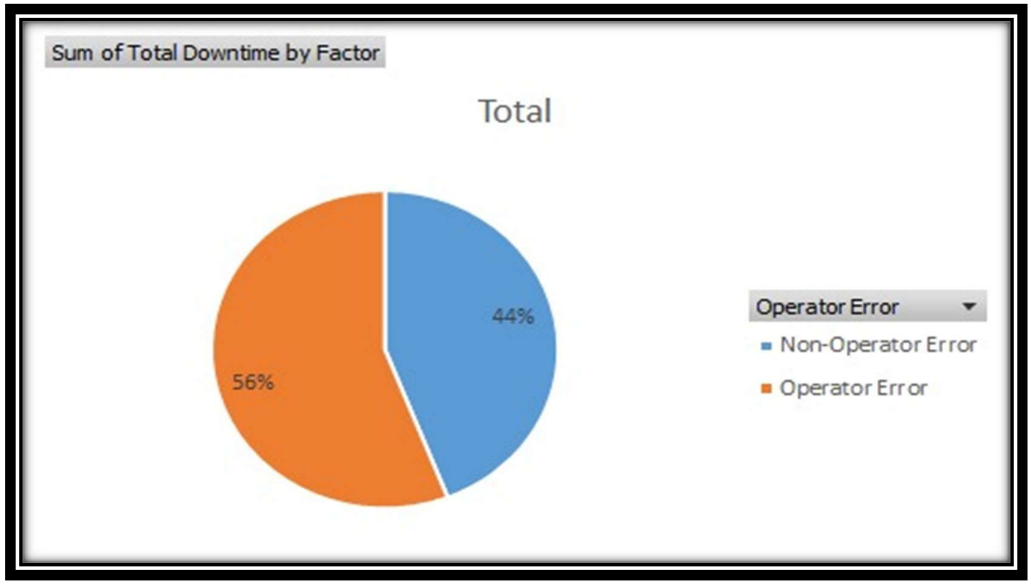
Python:

```python
#Q15) How much downtime is caused by operator errors vs. other causes in Minutes ?

# Aggregate total downtime minutes by classification
downtime_summary = df_downtime.groupby("Downtime Classification")["Minutes"].sum()

# Function to display absolute values instead of percentages
def absolute_values(values):
    return [f"{v:.0f} min" for v in values]

# Plot pie chart
fig, ax = plt.subplots(figsize=(8, 6))
wedges, texts, autotexts = ax.pie(
    downtime_summary, labels=downtime_summary.index,
    colors=["orange", "blue"], autopct=lambda p: f"{p * sum(downtime_summary) / 100:.0f} min"
)

# Format text size
for text in autotexts:
    text.set_fontsize(12)
    text.set_fontweight("bold")

# Add title
plt.title("Total Downtime Breakdown by Operator & Non-Operator")

# Show plot
plt.show()
```

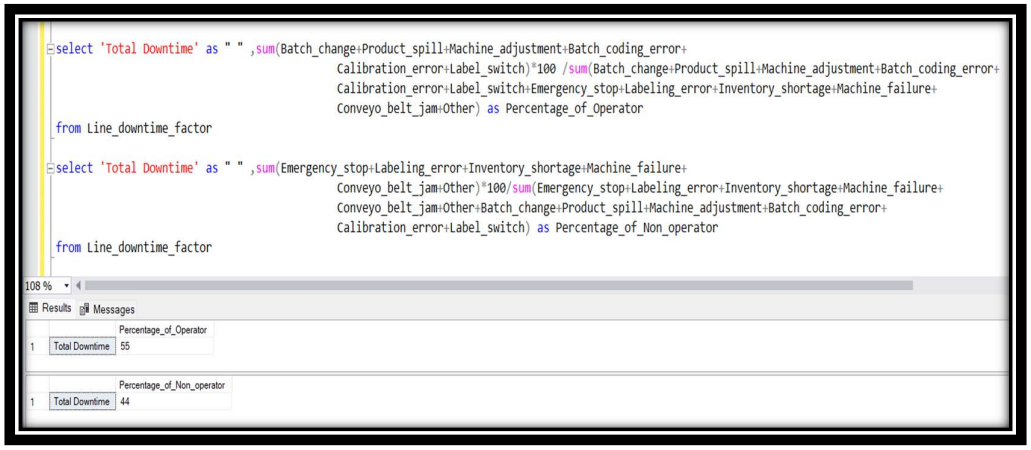


Q16) How much downtime is caused by operator errors vs. other causes in Percentage ?

Excel:



SQL:



```
select 'Total Downtime' as " " ,sum(Batch_change+Product_spill+Machine_adjustment+Batch_coding_error+
                                    Calibration_error+Label_switch)*100 /sum(Batch_change+Product_spill+Machine_adjustment+Batch_coding_error+
                                    Calibration_error+Label_switch+Emergency_stop+Labeling_error+Inventory_shortage+Machine_failure+
                                    Conveyo_belt_jam+Other) as Percentage_of_Operator
from Line_downtime_factor

select 'Total Downtime' as " " ,sum(Emergency_stop+Labeling_error+Inventory_shortage+Machine_failure+
                                    Conveyo_belt_jam+Other)*100/sum(Emergency_stop+Labeling_error+Inventory_shortage+Machine_failure+
                                    Conveyo_belt_jam+Other+Batch_change+Product_spill+Machine_adjustment+Batch_coding_error+
                                    Calibration_error+Label_switch) as Percentage_of_Non_operator
from Line_downtime_factor
```

108 %

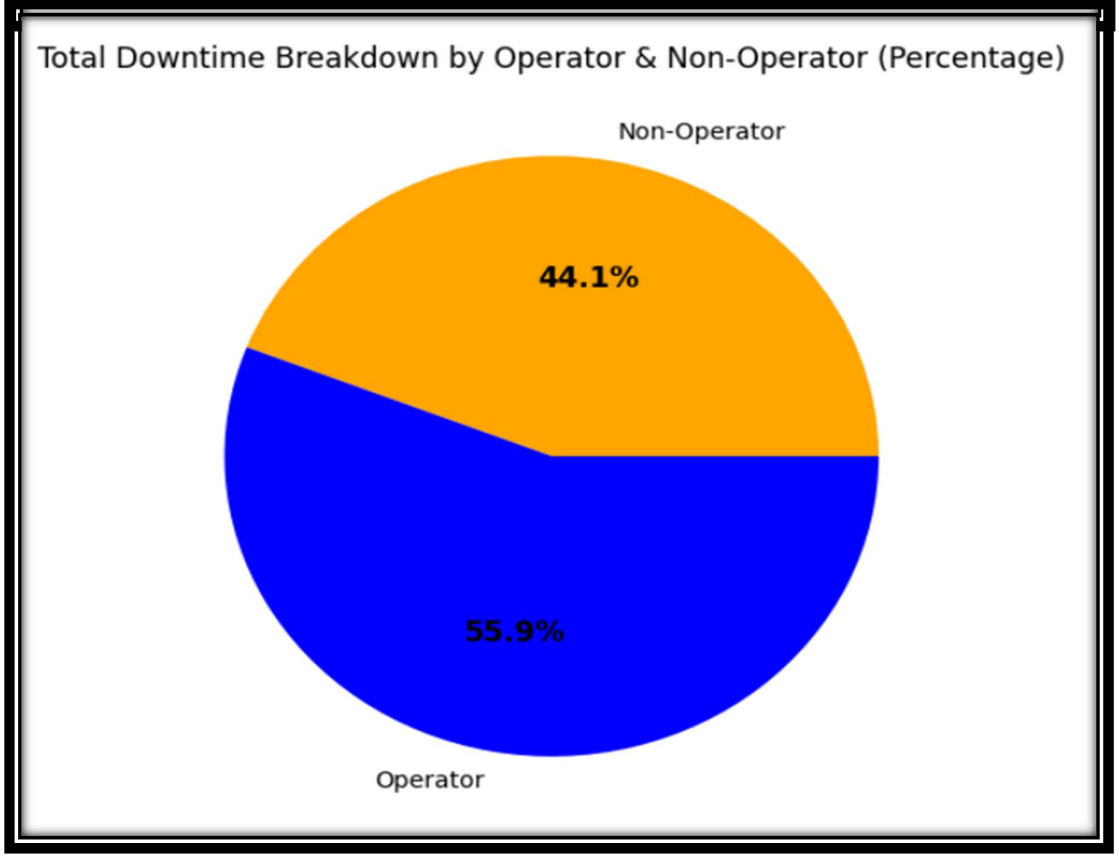
Results | Messages

		Percentage_of_Operator
1	Total Downtime	55

		Percentage_of_Non_operator
1	Total Downtime	44

Python:

```python
#Q16) How much downtime is caused by operator errors vs. other causes in Percentage ?

# Aggregate total downtime minutes by classification
downtime_summary = df_downtime.groupby("Downtime Classification")["Minutes"].sum()

# Plot pie chart
fig, ax = plt.subplots(figsize=(8, 6))
wedges, texts, autotexts = ax.pie(
    downtime_summary, labels=downtime_summary.index,
    colors=["orange", "blue"], autopct='%1.1f%%'  # Display percentage
)

# Format text size
for text in autotexts:
    text.set_fontsize(12)
    text.set_fontweight("bold")

# Add title
plt.title("Total Downtime Breakdown by Operator & Non-Operator (Percentage)")

# Show plot
plt.show()
```



Q17) What is the total downtime recorded?

Excel:

Sum of Total Downtime
1388

SQL:

```
select sum(Batch_downtime_by_factor) as Total_Downtime
from Line_downtime_factor1
```

119 %

Results | Messages

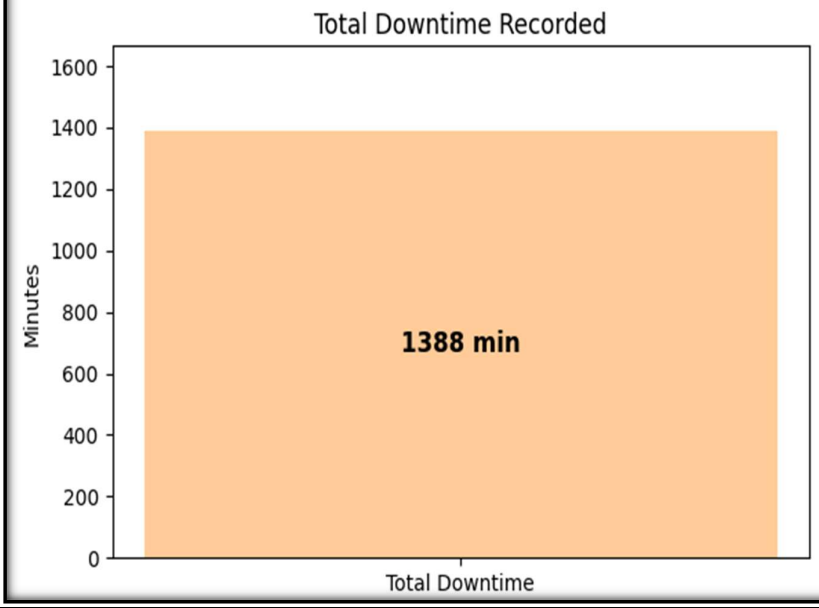
	Total_Downtime
1	1388

Python:

```
#Q17) What is the total downtime recorded?


df = pd.read_excel(file_path, sheet_name="Transformed Line Productivity")

# Ensure column names are stripped of spaces
df.columns = df.columns.str.strip()

# Calculate total downtime
total_downtime = df["Total Downtime by Batch"].sum()

# Print result
print(f"Total Downtime Recorded: {total_downtime} minutes")
```

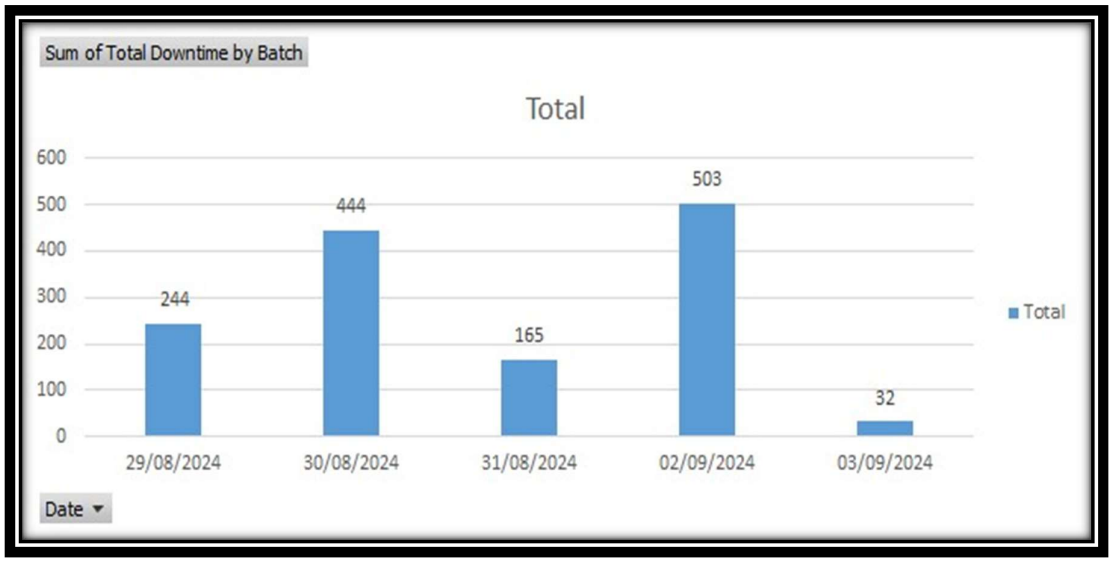
```
Total Downtime Recorded: 1388 minutes
```

```python
# Ensure column names are stripped of spaces
df_productivity.columns = df_productivity.columns.str.strip()

# Calculate total downtime
total_downtime = df_productivity["Total Downtime by Batch"].sum()

# Visualization: Total Downtime as a single metric bar
plt.figure(figsize=(6, 4))
plt.bar(["Total Downtime"], [total_downtime], color='#ffcc99')
plt.ylabel("Minutes")
plt.title("Total Downtime Recorded")
plt.text(0, total_downtime/2, f"{total_downtime:.0f} min", ha='center', va='center', fontsize=12, weight='bold')
plt.ylim(0, total_downtime * 1.2)
plt.tight_layout()
plt.show()
```



Q18) What is the total downtime for each day?

Excel:



SQL:

```
select lp.Date , sum(ld.Batch_downtime_by_factor) as Total_Downtime
from Line_productivity lp left outer join Line_downtime_factor1 ld
on lp.Batch = ld.Batch
group by lp.Date
order by Total_Downtime desc
```

119 %

Results | Messages

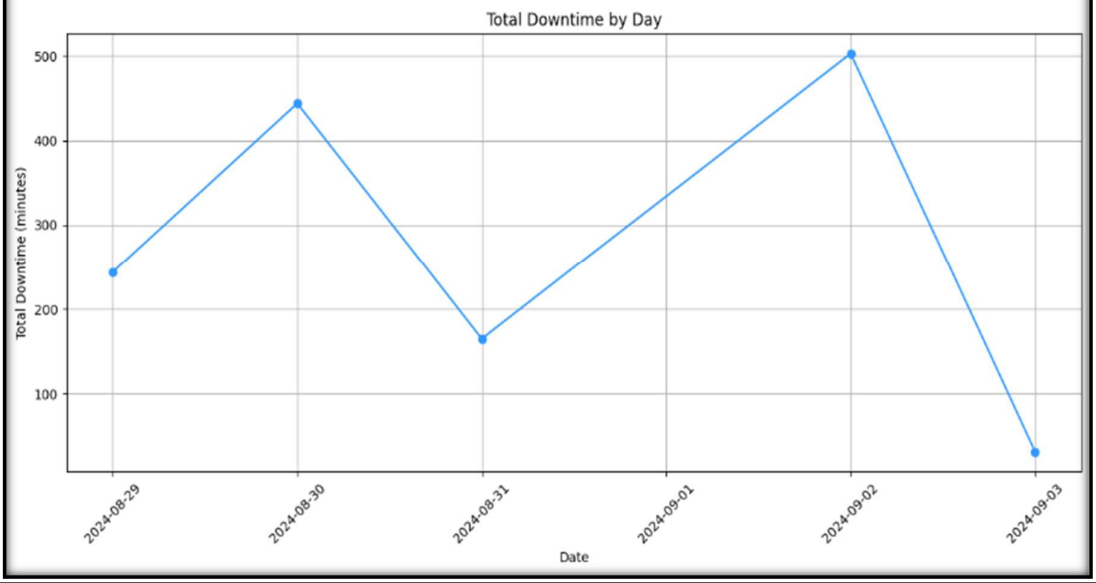
	Date	Total_Downtime
1	2024-09-02	503
2	2024-08-30	444
3	2024-08-29	244
4	2024-08-31	165
5	2024-09-03	32

Python:

```python
#Q18) What is the total downtime for each day?


# Load the "Transformed Line Productivity" sheet
df = pd.read_excel(r"F:\Rawad Misr\Data Analysis\Projects\Final Projects\Final Project\Manufacturing_Line_Productivity.xlsx", sheet_name="Transformed Li

# Ensure column names are stripped of spaces
df.columns = df.columns.str.strip()

# Convert "Date" column to datetime format
df["Date"] = pd.to_datetime(df["Date"])

# Group by Date and sum the total downtime
daily_downtime = df.groupby(df["Date"].dt.date)["Total Downtime by Batch"].sum().reset_index()

# Rename columns for clarity
daily_downtime.columns = ["Date", "Total Downtime by Batch"]

# Print results
print(daily_downtime)
```

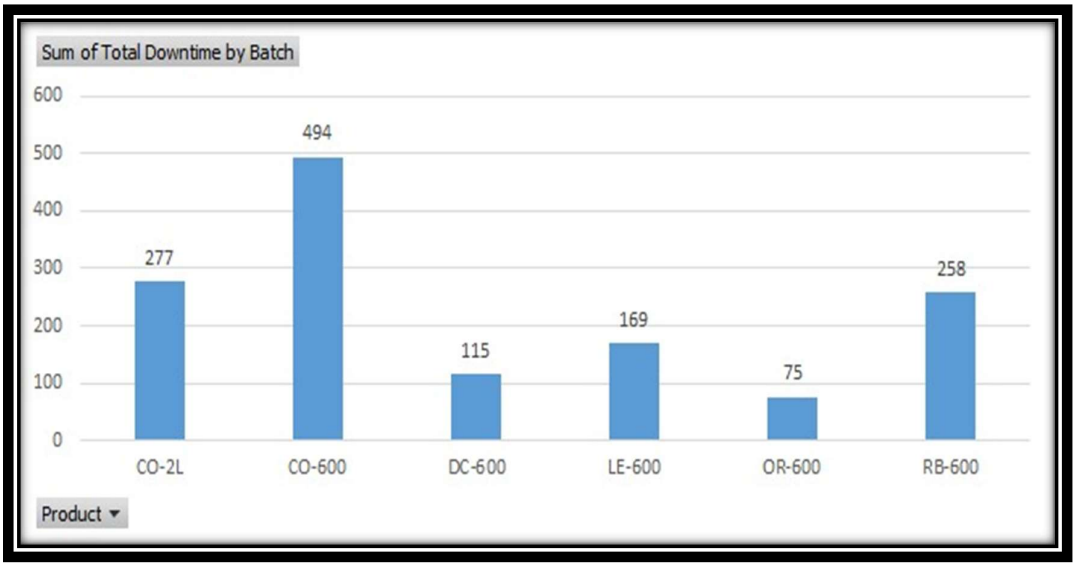
```
         Date  Total Downtime by Batch
0  2024-08-29                      244
1  2024-08-30                      444
2  2024-08-31                      165
3  2024-09-02                      503
4  2024-09-03                       32
```

```python
# Convert "Date" column to datetime format
df_productivity["Date"] = pd.to_datetime(df_productivity["Date"])

# Group by Date and sum the total downtime
daily_downtime = df_productivity.groupby(df_productivity["Date"].dt.date)["Total Downtime by Batch"].sum().reset_index()
daily_downtime.columns = ["Date", "Total Downtime by Batch"]

# Visualization: Line plot of total downtime per day
plt.figure(figsize=(12, 6))
plt.plot(daily_downtime["Date"], daily_downtime["Total Downtime by Batch"], marker='o', color='#3399ff')
plt.title("Total Downtime by Day")
plt.xlabel("Date")
plt.ylabel("Total Downtime (minutes)")
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



19) Which Product experienced the most downtime?

Excel:



SQL:

```
select lp.Product, SUM(Emergency_stop+Batch_change
                       +Labeling_error+Inventory_shortage+Product_spill
                       +Machine_adjustment+Machine_failure+Batch_coding_error+Conveyo_belt_jam
                       +Calibration_error+Label_switch+Other) AS Total_Downtime_Duration
from Line_productivity lp JOIN Line_downtime_factor ldf
on lp.Batch = ldf.Batch
group by lp.Product
order by Total_Downtime_Duration desc
```

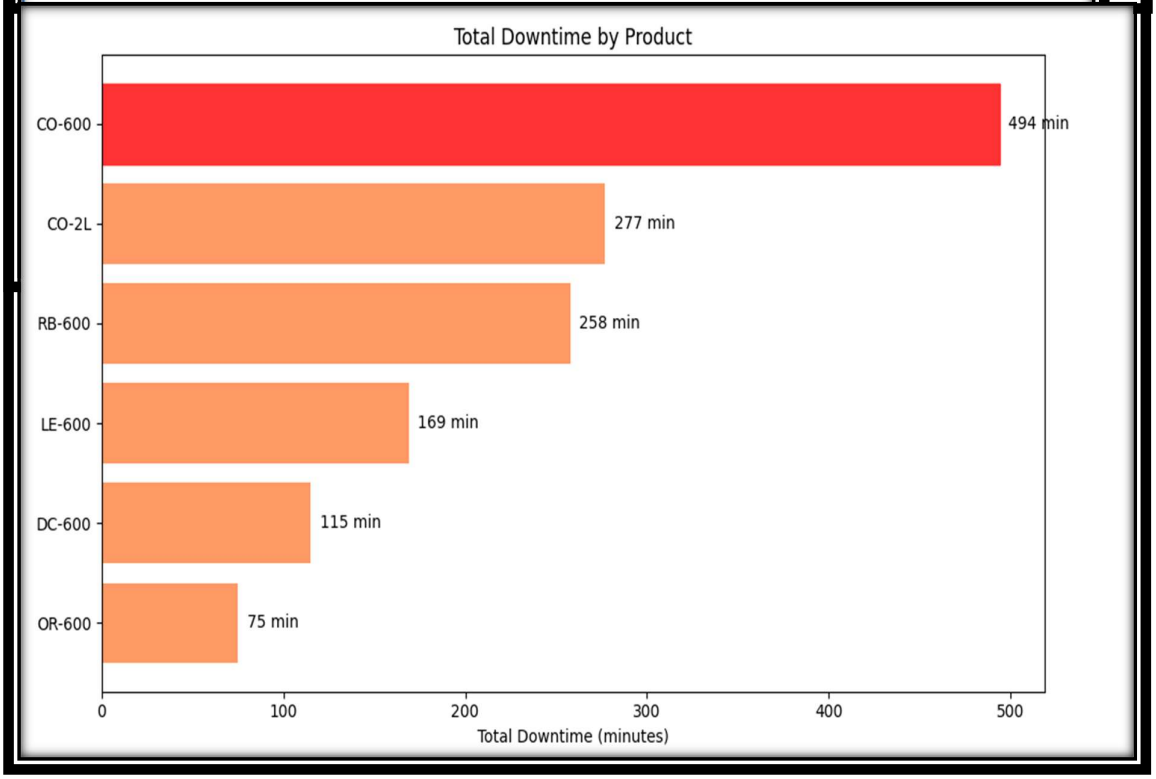
119 %

Results | Messages

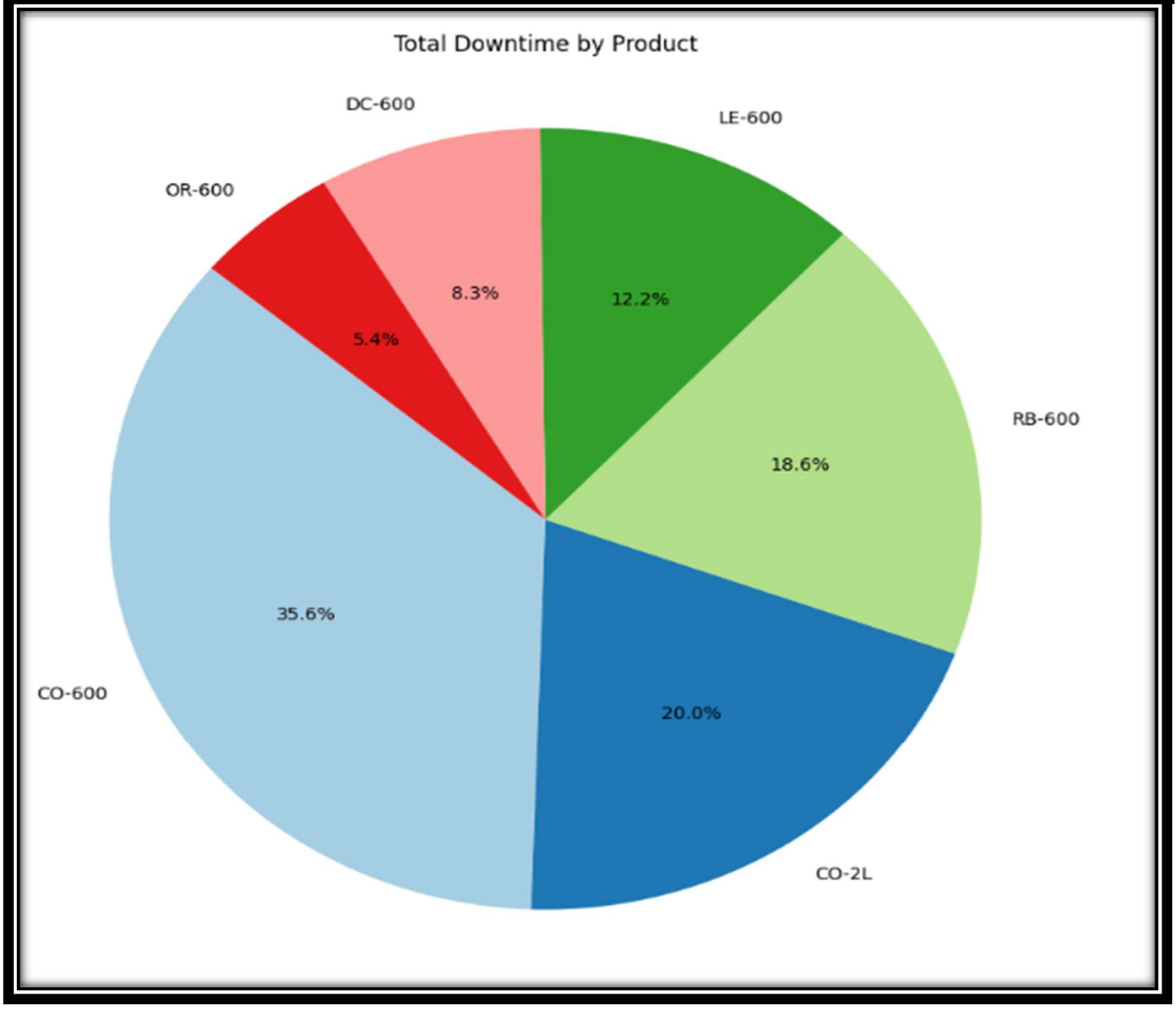
	Product	Total_Downtime_Duration
1	CO-600	494
2	CO-2L	277
3	RB-600	258
4	LE-600	169
5	DC-600	115
6	OR-600	75

Python:

```python
#19) Which Product experienced the most downtime?

# Group by Product and sum total downtime
product_downtime = df_productivity.groupby("Product")["Total Downtime by Batch"].sum().sort_values(ascending=False)

# Identify the product with the most downtime
most_downtime_product = product_downtime.idxmax()
max_downtime = product_downtime.max()

# Visualization: Horizontal bar chart with labels
plt.figure(figsize=(10, 6))
bars = plt.barh(product_downtime.index, product_downtime.values, color='#ff9966')
plt.xlabel("Total Downtime (minutes)")
plt.title("Total Downtime by Product")

# Highlight the product with the most downtime
bars[0].set_color('#ff3333')

# Add labels to each bar
for bar in bars:
    width = bar.get_width()
    plt.text(width + 5, bar.get_y() + bar.get_height()/2,
             f'{width:.0f} min', va='center', fontsize=10)

plt.gca().invert_yaxis()  # Highest bar at the top
plt.tight_layout()
plt.show()
```

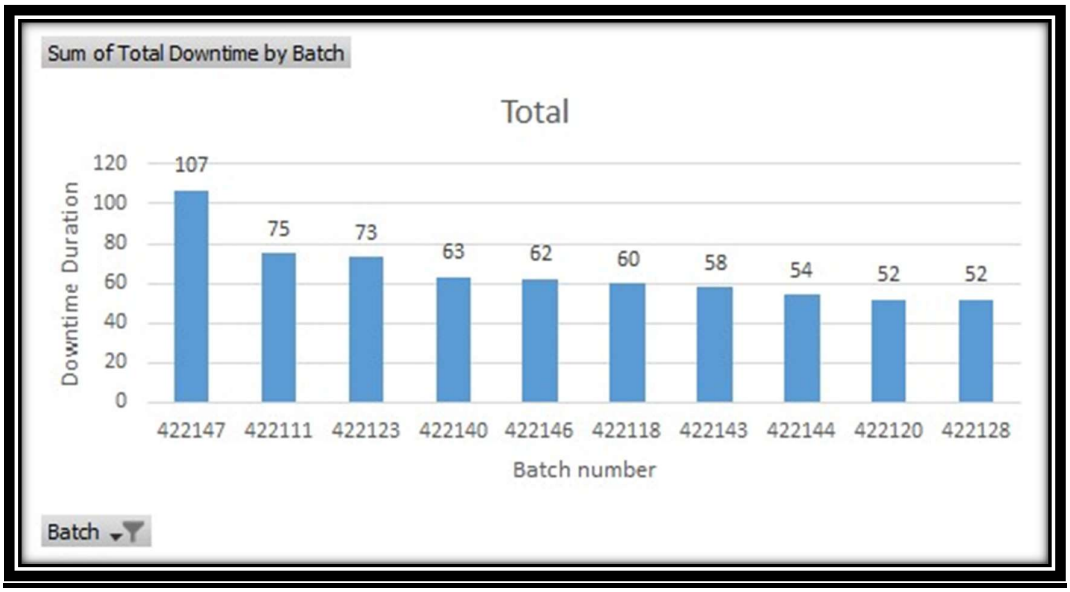


```python
#19) Which Product experienced the most downtime?

# Group by Product and sum total downtime
product_downtime = df_productivity.groupby("Product")["Total Downtime by Batch"].sum().sort_values(ascending=False)

# Identify the product with the most downtime
most_downtime_product = product_downtime.idxmax()
max_downtime = product_downtime.max()

# Plot as pie chart
plt.figure(figsize=(8, 8))
plt.pie(product_downtime, labels=product_downtime.index, autopct='%1.1f%%', startangle=140,
        colors=plt.cm.Paired.colors)
plt.title("Total Downtime by Product")
plt.axis('equal')  # Equal aspect ratio ensures a perfect circle
plt.tight_layout()
plt.show()
```

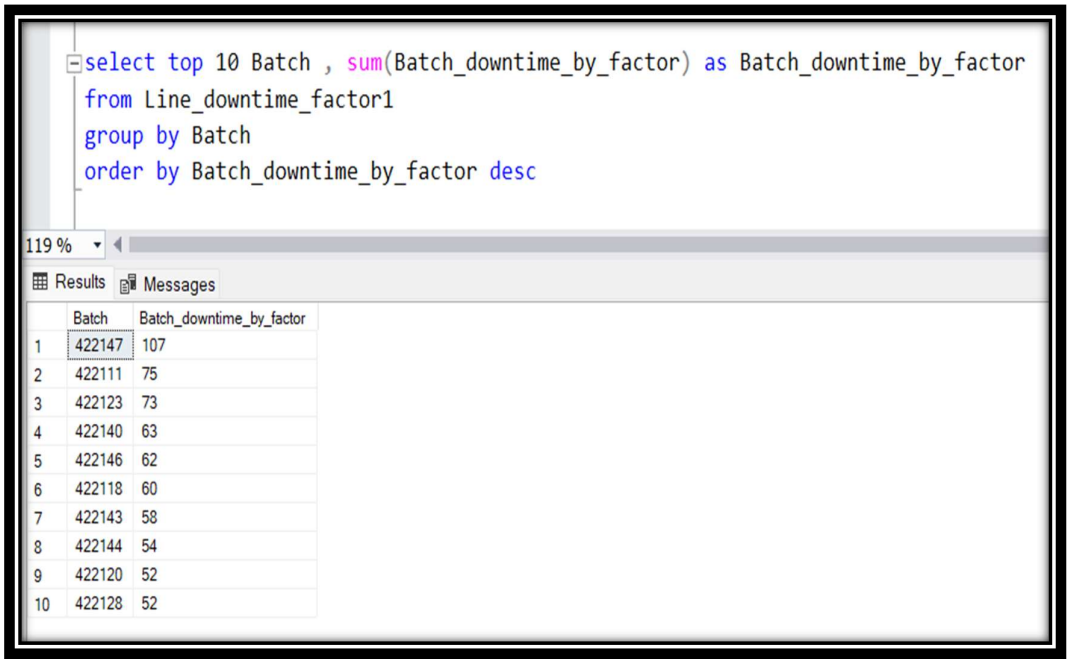


Q20) Which Top 10 Batches with Most Downtime ?

Excel:



SQL:



```
select top 10 Batch , sum(Batch_downtime_by_factor) as Batch_downtime_by_factor
from Line_downtime_factor1
group by Batch
order by Batch_downtime_by_factor desc
```

119 %

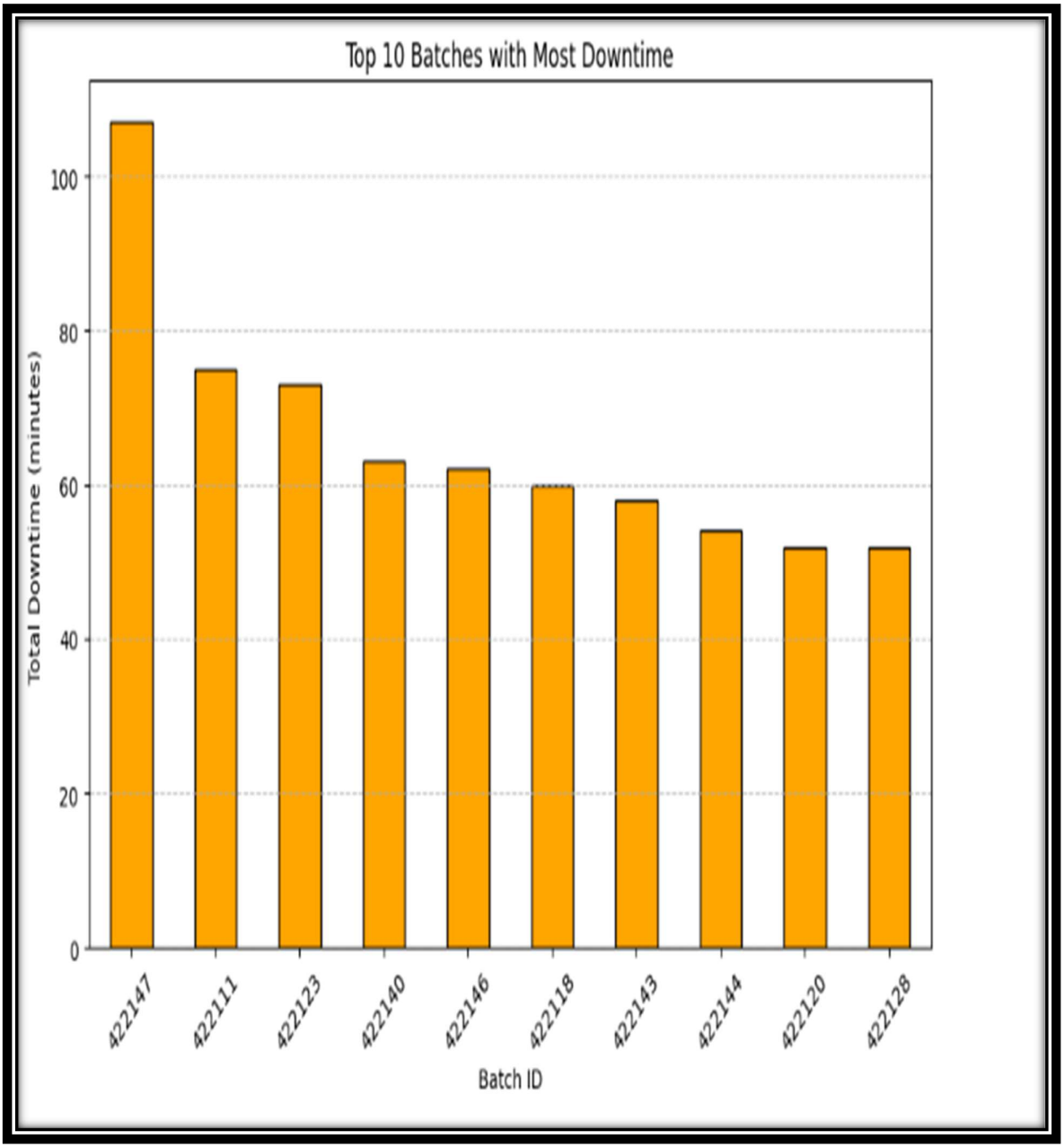
Results | Messages

	Batch	Batch_downtime_by_factor
1	422147	107
2	422111	75
3	422123	73
4	422140	63
5	422146	62
6	422118	60
7	422143	58
8	422144	54
9	422120	52
10	422128	52

Python :

```
#Q20) Which Top 10 Batches with Most Downtime ?


# Read the Transformed Line Downtime sheet
df_downtime = pd.read_excel(file_path, sheet_name="Transformed Line Productivity")
df_downtime.columns = df_downtime.columns.str.strip()  # Remove extra spaces

# Print actual column names
print("Columns in Transformed Line Downtime sheet:", df_downtime.columns.tolist())

# Manually set correct column names after checking the output
batch_column = "Batch"  # Replace with actual batch column name
duration_column = "Total Downtime by Batch"  # Replace with actual downtime column name

# Group by batch and sum downtime duration
batch_downtime = df_downtime.groupby(batch_column)[duration_column].sum()

# Get the top 10 batches with the most downtime
top_batches = batch_downtime.nlargest(10)

# Print the results
print("Top Batches with Most Downtime:\n", top_batches)

# Visualize as a bar chart
plt.figure(figsize=(10, 6))
top_batches.plot(kind="bar", color="orange", edgecolor="black")

# Add labels and title
plt.xlabel("Batch ID")
plt.ylabel("Total Downtime (minutes)")
plt.title("Top 10 Batches with Most Downtime")
plt.xticks(rotation=45)
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Show the plot
plt.show()
```

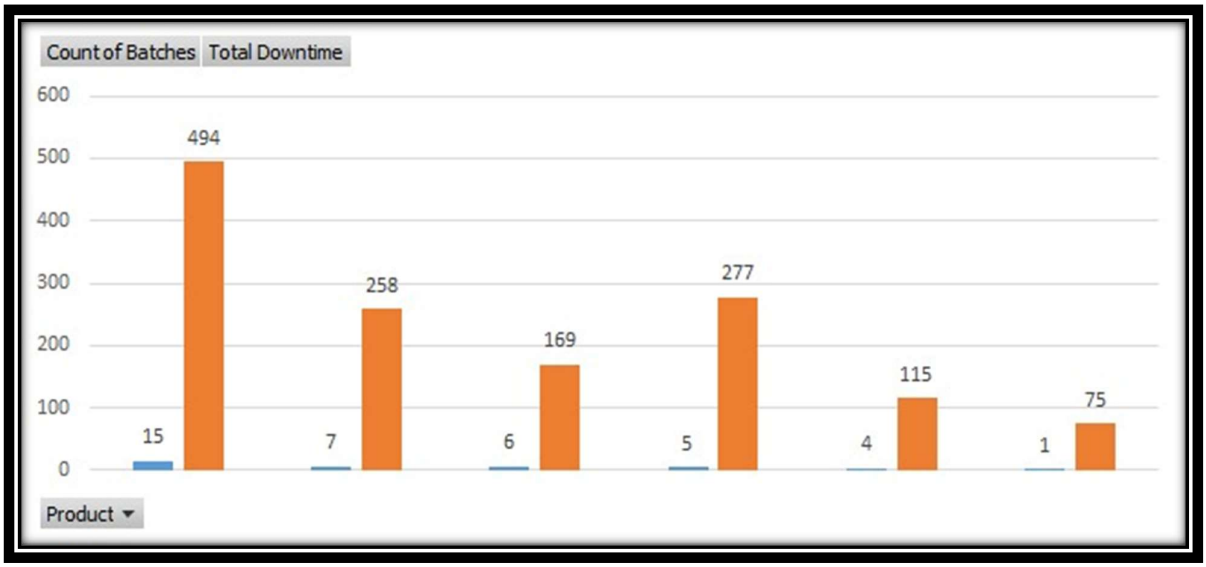
```
Columns in Transformed Line Downtime sheet: ['Date', 'Product', 'Batch', 'Operator', 'Start Time', 'End Time', 'Batch Duration', 'Min batch time', 'Tot
al Downtime by Batch']
Top Batches with Most Downtime:
 Batch
422147    107
422111     75
422123     73
422140     63
422146     62
422118     60
422143     58
422144     54
422120     52
422128     52
Name: Total Downtime by Batch, dtype: int64
```


Top 10 Batches with Most Downtime
Total Downtime (minutes)
0
20
40
60
80
100
422147
422111
422123
422140
422146
422118
422143
422144
422120
422128
Batch ID

Performance Trends

Q21) Is there a relationship between batch size and downtime?

Excel:



SQL:

```
select lp.Product ,count(Batch) as Batch_count
from Line_productivity lp left join Products p
on lp.Product=p.Product
group by lp.Product
order by count(Batch) desc

select lp.Product, SUM(Emergency_stop+Batch_change
                       +Labeling_error+Inventory_shortage+Product_spill
                       +Machine_adjustment+Machine_failure+Batch_coding_error+Conveyo_belt_jam
                       +Calibration_error+Label_switch+Other) AS Total_Downtime_Duration
from Line_productivity lp JOIN Line_downtime_factor ldf
on lp.Batch = ldf.Batch
group by lp.Product
order by Total_Downtime_Duration desc
```

108 %

Results | Messages

	Product	Batch_count
1	CO-600	15
2	RB-600	7
3	LE-600	6
4	CO-2L	5
5	DC-600	4
6	OR-600	1

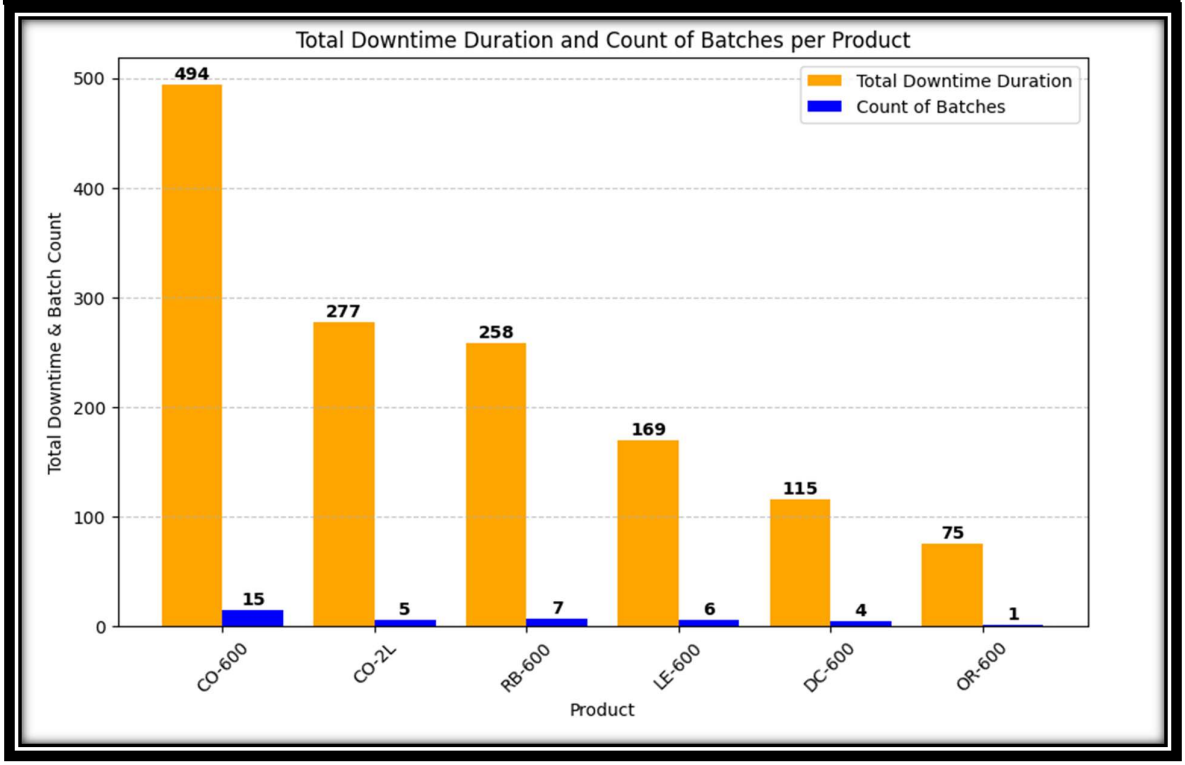
	Product	Total_Downtime_Duration
1	CO-600	494
2	CO-2L	277
3	RB-600	258
4	LE-600	169
5	DC-600	115
6	OR-600	75

Python:

```python
#Q21) Is there a relationship between batch size and downtime?


from scipy.stats import pearsonr

# Remove any extra spaces in column names
df.columns = df.columns.str.strip()

# Print column names for verification
print(df.columns)

# Ensure correct column names
if "Product" not in df.columns or "Total Downtime by Batch" not in df.columns:
    raise KeyError("Check column names! Expected 'Product' and 'Total Downtime by Batch'.")

# Aggregate total downtime and batch count by product
downtime_per_product = df.groupby("Product")["Total Downtime by Batch"].sum()
batch_count_per_product = df.groupby("Product")["Batch"].count()

# Check correlation between batch count and downtime
correlation, p_value = pearsonr(batch_count_per_product, downtime_per_product)

# Determine relationship
if abs(correlation) > 0.5 and p_value < 0.05:
    print(f"✅ There is a significant relationship between batch count and downtime (Correlation: {correlation:.2f}, p-value: {p_value:.5f})")
else:
    print(f"❌ There is no strong relationship between batch count and downtime (Correlation: {correlation:.2f}, p-value: {p_value:.5f})")
```

```python
# Sort by total downtime in descending order
downtime_per_product = downtime_per_product.sort_values(ascending=False)
batch_count_per_product = batch_count_per_product.loc[downtime_per_product.index]  # Reorder batch count to match sorted downtime

# Plot grouped bar chart
fig, ax = plt.subplots(figsize=(10, 6))

# Define bar width and positions
bar_width = 0.4
x_indexes = np.arange(len(downtime_per_product))

# Plot bars
bars1 = ax.bar(x_indexes - bar_width/2, downtime_per_product, bar_width, color="orange", label="Total Downtime Duration")
bars2 = ax.bar(x_indexes + bar_width/2, batch_count_per_product, bar_width, color="blue", label="Count of Batches")

# Add data labels
for bar in bars1:
    ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 5, int(bar.get_height()), ha="center", fontsize=10, fontweight="bold")

for bar in bars2:
    ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 5, int(bar.get_height()), ha="center", fontsize=10, fontweight="bold")

# Labels and title
ax.set_xlabel("Product")
ax.set_ylabel("Total Downtime & Batch Count")
ax.set_title("Total Downtime Duration and Count of Batches per Product")
ax.set_xticks(x_indexes)
ax.set_xticklabels(downtime_per_product.index, rotation=45)
ax.legend()

plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

```
Index(['Date', 'Product', 'Batch', 'Operator', 'Start Time', 'End Time',
       'Batch Duration', 'Min batch time', 'Total Downtime by Batch'],
      dtype='object')
✅ There is a significant relationship between batch count and downtime (Correlation: 0.94, p-value: 0.00455)
```



Q22) Are there peak production hours or days where efficiency is highest?

No

Q23) How does downtime impact the number of batches produced daily?

More downtime less patches produced

Phase III

Forecasting Approach.

Forecasting Approach.

Model Selection

- **Regression Models:** Random Forest Regressor

Forecasting Questions

```
#Phase 3: Forecasting Questions
```

```
#1) Data Preparation: Extracting time series data for downtime and batch production.

# Check available sheet names
xls = pd.ExcelFile(file_path)
xls.sheet_names
```

```
['Line productivity',
 'Products',
 'Downtime factors',
 'Line downtime',
 'Transformed Line downtime',
 'Transformed Products',
 'Transformed Line Productivity']
```

```
# Display the first few rows
df.head()
```

	Date	Product	Batch	Operator	Start Time	End Time	Batch Duration	Min batch time	Total Downtime by Batch
0	2024-08-29	OR-600	422111	Mac	11:50:00	14:05:00	135	60	75
1	2024-08-29	LE-600	422112	Mac	14:05:00	15:45:00	100	60	40
2	2024-08-29	LE-600	422113	Mac	15:45:00	17:35:00	110	60	50
3	2024-08-29	LE-600	422114	Mac	17:35:00	19:15:00	100	60	40
4	2024-08-29	LE-600	422115	Charlie	19:15:00	20:39:00	84	60	24

```
# Convert "Date" to datetime format
df["Date"] = pd.to_datetime(df["Date"])

# Aggregate total downtime and batch count per day
daily_data = df.groupby("Date").agg(
    Total_Downtime=("Total Downtime by Batch", "sum"),
    Batch_Count=("Batch", "count")
).reset_index()

# Display the first few rows of the processed data
daily_data.head()
```

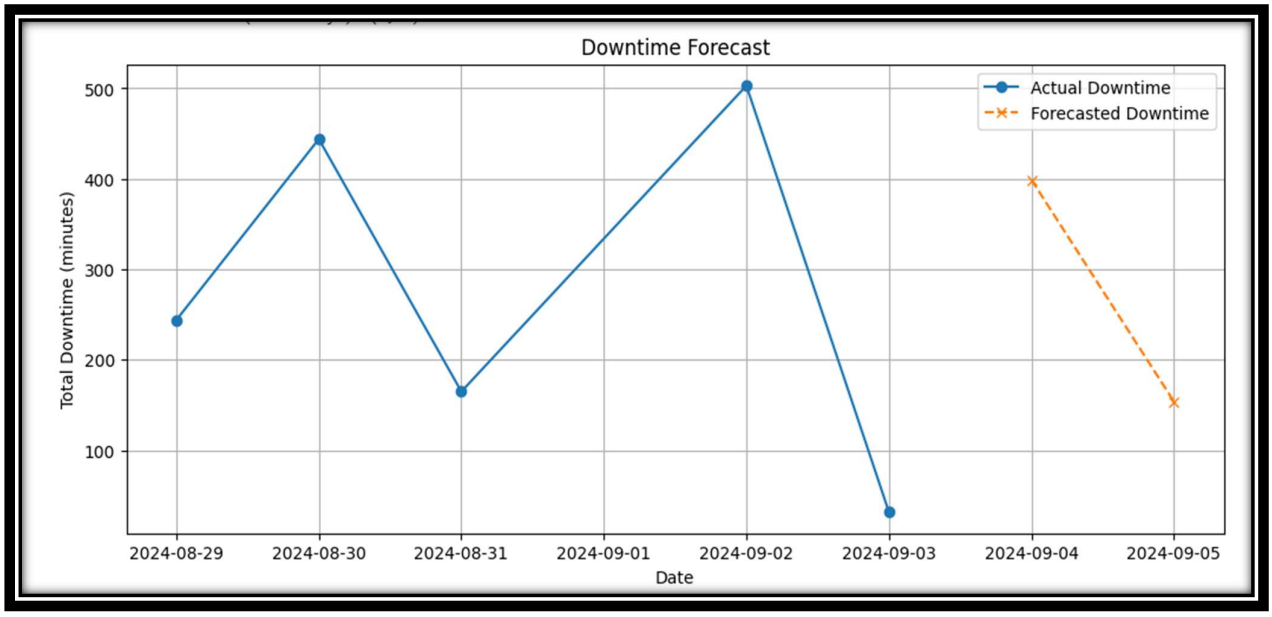
	Date	Total_Downtime	Batch_Count
0	2024-08-29	244	7
1	2024-08-30	444	12
2	2024-08-31	165	7
3	2024-09-02	503	11
4	2024-09-03	32	1

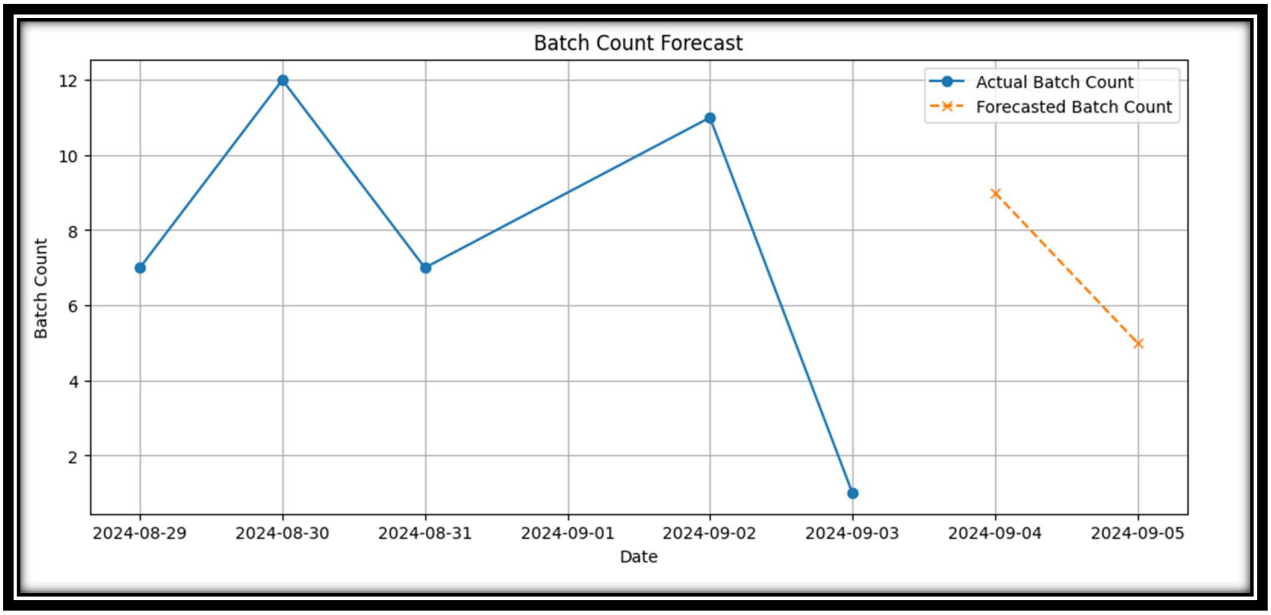
```python
# Forecasting Downtime and Batch Count

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor

# Load the Excel file
file_path = r"F:\Rawad Misr\Data Analysis\Projects\Final Projects\Final Project\Manufacturing_Line_Productivity.xlsx"
xls = pd.ExcelFile(file_path)

# Parse the productivity sheet and convert dates
productivity_df = xls.parse('Transformed Line Productivity')
productivity_df['Date'] = pd.to_datetime(productivity_df['Date'])

# Aggregate total downtime and batch count per day
daily_summary = productivity_df.groupby('Date').agg(
    total_downtime=('Total Downtime by Batch', 'sum'),
    batch_count=('Batch', 'nunique')
).reset_index()

# Create lag features for time series forecasting
def create_lag_features(df, target_col, lags=[1, 2, 3]):
    df = df.copy()
    for lag in lags:
        df[f'{target_col}_lag{lag}'] = df[target_col].shift(lag)
    return df.dropna()

# Downtime model
lagged_downtime = create_lag_features(daily_summary, 'total_downtime')
X_downtime = lagged_downtime[[f'total_downtime_lag{i}' for i in [1, 2, 3]]]
y_downtime = lagged_downtime['total_downtime']
downtime_model = RandomForestRegressor(random_state=42).fit(X_downtime, y_downtime)

# Batch count model
lagged_batch = create_lag_features(daily_summary, 'batch_count')
X_batch = lagged_batch[[f'batch_count_lag{i}' for i in [1, 2, 3]]]
y_batch = lagged_batch['batch_count']
batch_model = RandomForestRegressor(random_state=42).fit(X_batch, y_batch)
```

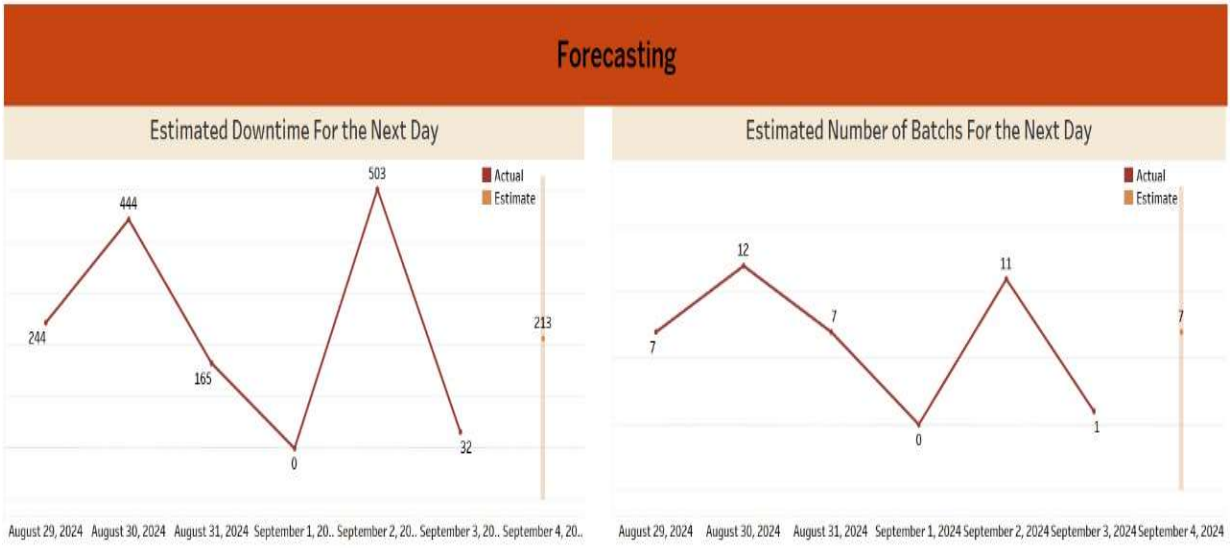
```python
# Forecast next two days
def forecast_next_two(model, last_values):
    pred_day1 = model.predict([last_values])[0]
    pred_day2 = model.predict([[pred_day1] + list(last_values[:2])])[0]
    return round(pred_day1), round(pred_day2)

last_downtime_vals = daily_summary['total_downtime'].iloc[-3:].values[::-1]
downtime_forecast = forecast_next_two(downtime_model, last_downtime_vals)

last_batch_vals = daily_summary['batch_count'].iloc[-3:].values[::-1]
batch_forecast = forecast_next_two(batch_model, last_batch_vals)

print("Predicted Downtime (next 2 days):", downtime_forecast)
print("Predicted Batch Count (next 2 days):", batch_forecast)
```

```python
# Visualization
forecast_dates = pd.date_range(start=daily_summary['Date'].max() + pd.Timedelta(days=1), periods=2)

# Extend actual data for plotting
plot_df = daily_summary.copy()
forecast_df = pd.DataFrame({
    'Date': forecast_dates,
    'total_downtime': downtime_forecast,
    'batch_count': batch_forecast
})

# Plot downtime
plt.figure(figsize=(12, 5))
plt.plot(plot_df['Date'], plot_df['total_downtime'], label='Actual Downtime', marker='o')
plt.plot(forecast_df['Date'], forecast_df['total_downtime'], label='Forecasted Downtime', marker='x', linestyle='--')
plt.title('Downtime Forecast')
plt.xlabel('Date')
plt.ylabel('Total Downtime (minutes)')
plt.legend()
plt.grid(True)
plt.show()

# Plot batch count
plt.figure(figsize=(12, 5))
plt.plot(plot_df['Date'], plot_df['batch_count'], label='Actual Batch Count', marker='o')
plt.plot(forecast_df['Date'], forecast_df['batch_count'], label='Forecasted Batch Count', marker='x', linestyle='--')
plt.title('Batch Count Forecast')
plt.xlabel('Date')
plt.ylabel('Batch Count')
plt.legend()
plt.grid(True)
plt.show()
```

```
Predicted Downtime (next 2 days): (399, 154)
Predicted Batch Count (next 2 days): (9, 5)
```


Downtime Forecast
Actual Downtime
Forecasted Downtime
Total Downtime (minutes)
500
400
300
200
100
2024-08-29
2024-08-30
2024-08-31
2024-09-01
2024-09-02
2024-09-03
2024-09-04
2024-09-05
Date


Batch Count Forecast
Actual Batch Count
Forecasted Batch Count
Batch Count
12
10
8
6
4
2
2024-08-29
2024-08-30
2024-08-31
2024-09-01
2024-09-02
2024-09-03
2024-09-04
2024-09-05
Date


Forecasting
Estimated Downtime For the Next Day
Actual
Estimate
503
444
244
213
165
0
32
August 29, 2024
August 30, 2024
August 31, 2024
September 1, 20..
September 2, 20..
September 3, 20..
September 4, 20..
Estimated Number of Batchs For the Next Day
Actual
Estimate
12
11
7
7
0
1
7
August 29, 2024
August 30, 2024
August 31, 2024
September 1, 2024
September 2, 2024
September 3, 2024
September 4, 2024

Phase IV

Dashboard &Visualization

Results & Insights.

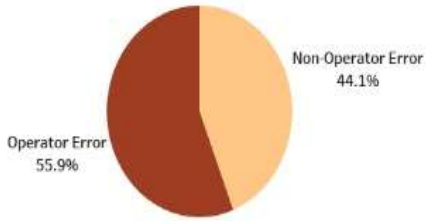
- Summary of key findings
- Model performance comparisons
- Business implications of downtime trends

Downtime Analysis

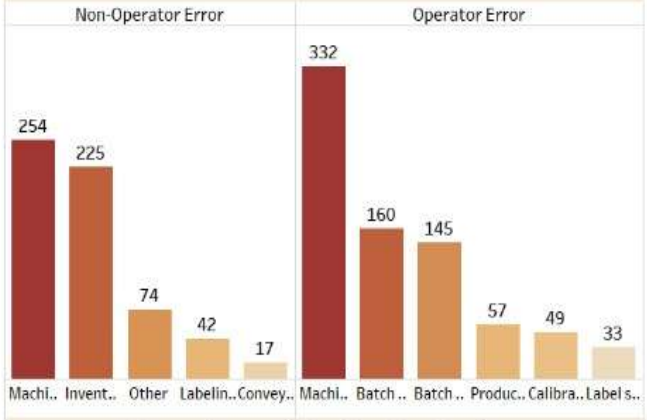
#Batchs	#Products	#Factors
38	6	12

Expected Prod. Time	Prod. Time	Downtime
2,470	3,858	1,388

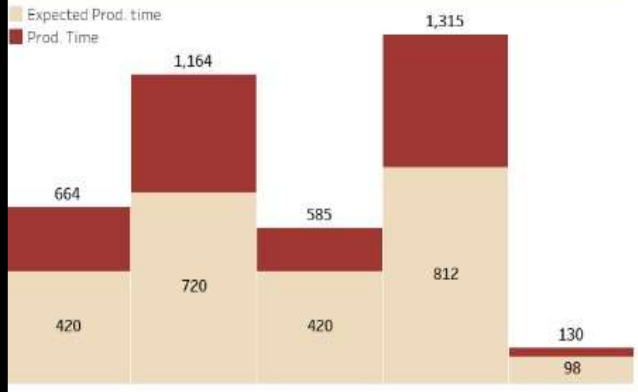
% of total Downtime per factor



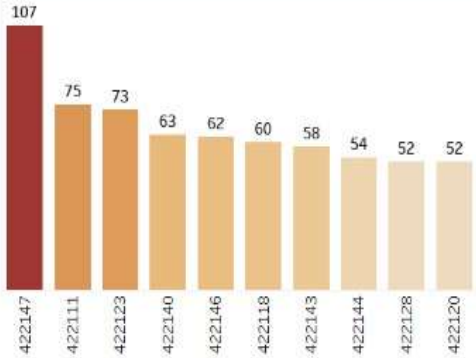
Total Downtime for each factor



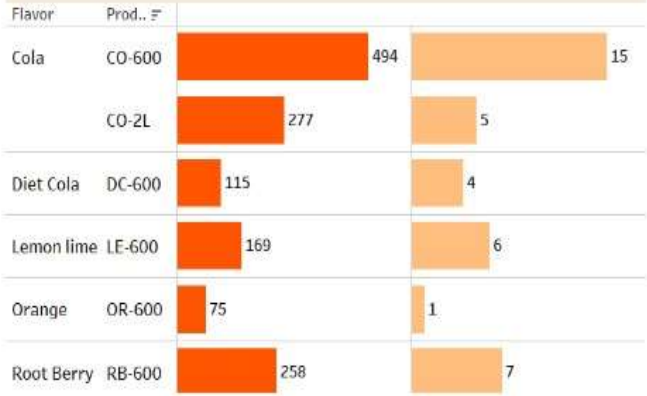
Expected VS Actual production duration

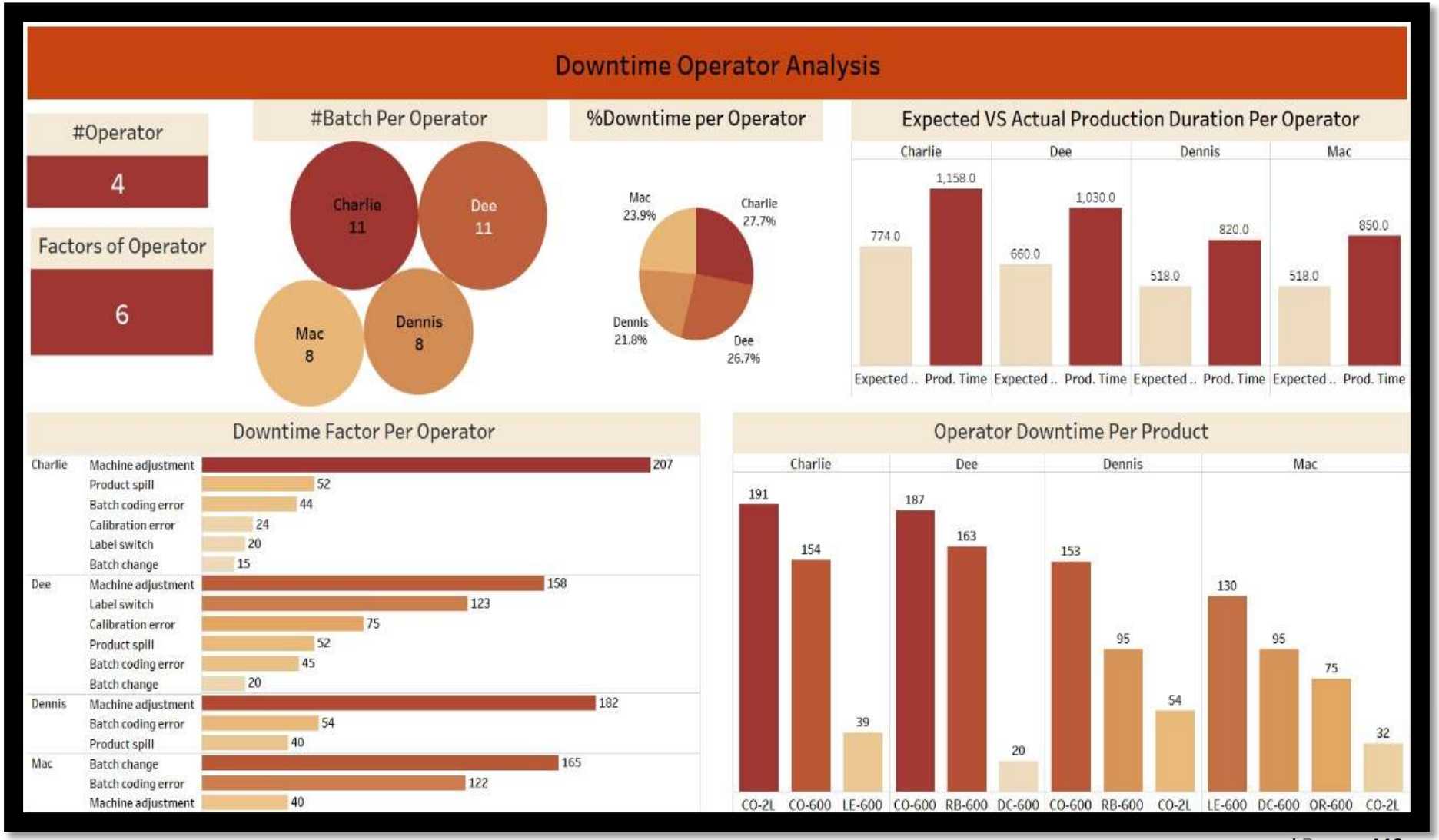


Top 10 Batches With Downtime



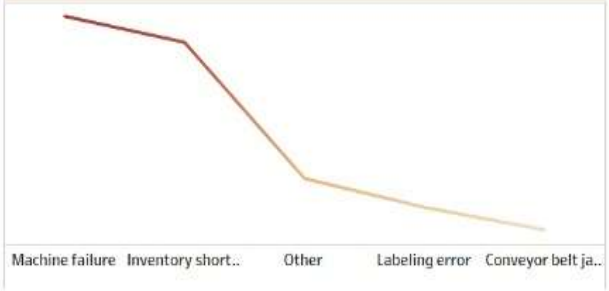
Downtime and Batches Per Product



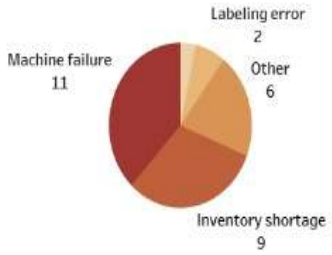

Downtime Operator Analysis
#Operator
4
Factors of Operator
6
#Batch Per Operator
Charlie 11
Dee 11
Mac 8
Dennis 8
%Downtime per Operator
Mac 23.9%
Charlie 27.7%
Dennis 21.8%
Dee 26.7%
Expected VS Actual Production Duration Per Operator
Charlie
774.0
1,158.0
Dee
660.0
1,030.0
Dennis
518.0
820.0
Mac
518.0
850.0
Expected .. Prod. Time
Downtime Factor Per Operator
Charlie
Machine adjustment 207
Product spill 52
Batch coding error 44
Calibration error 24
Label switch 20
Batch change 15
Dee
Machine adjustment 158
Label switch 123
Calibration error 75
Product spill 52
Batch coding error 45
Batch change 20
Dennis
Machine adjustment 182
Batch coding error 54
Product spill 40
Mac
Batch change 165
Batch coding error 122
Machine adjustment 40
Operator Downtime Per Product
Charlie
191
154
39
CO-2L CO-600 LE-600
Dee
187
163
20
CO-600 RB-600 DC-600
Dennis
153
95
54
CO-600 RB-600 CO-2L
Mac
130
95
75
32
LE-600 DC-600 OR-600 CO-2L

Downtime Non-Operator Analysis

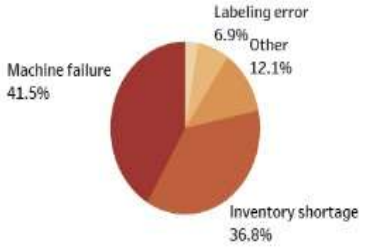
Downtime of factors



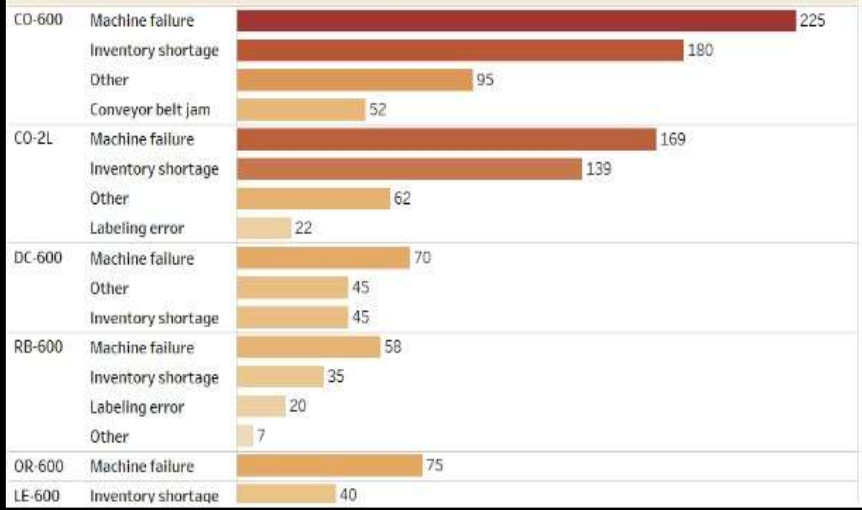
#Batch of Each Factor



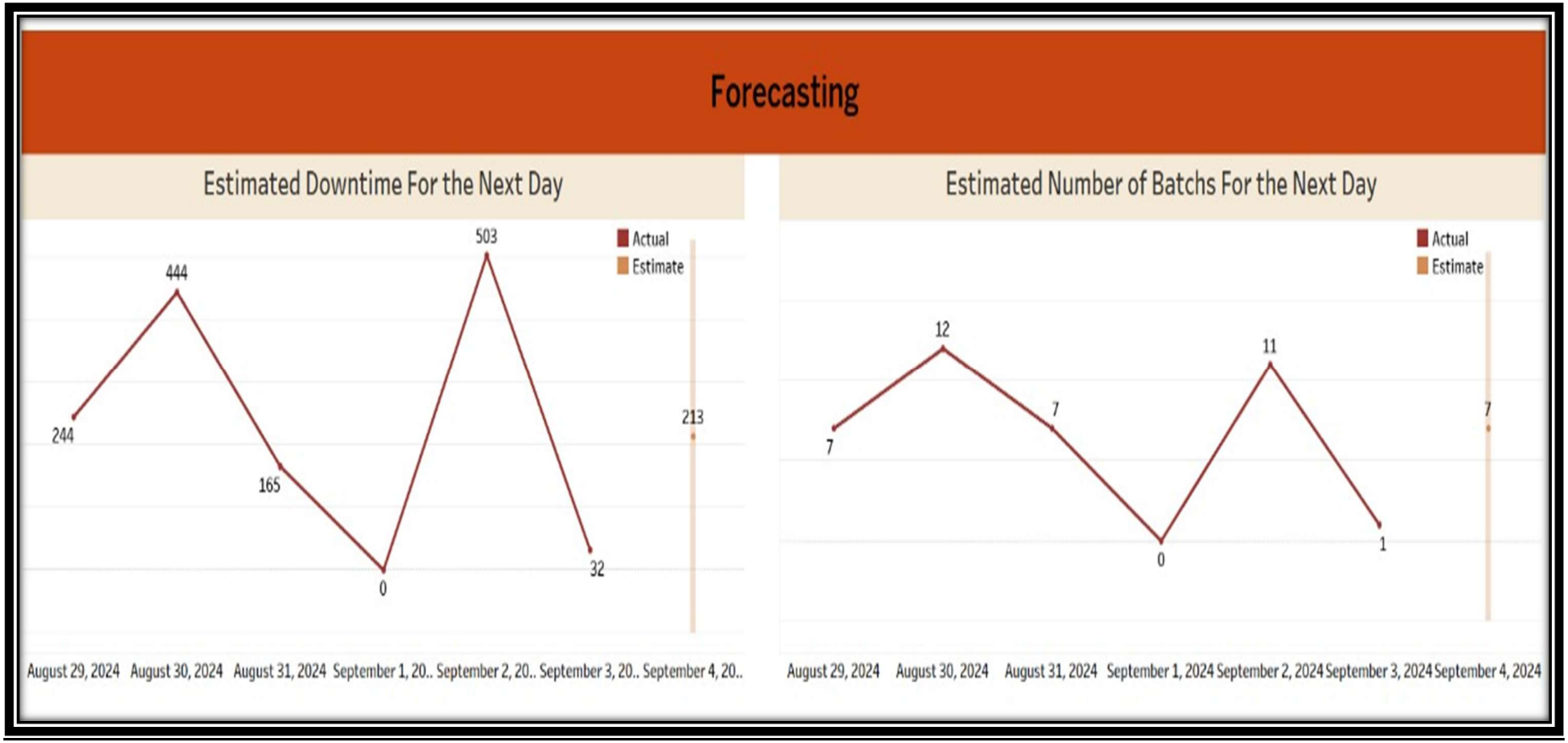
%Downtime of Each Factor



Downtime of factors for each Product



Expected VS Actual Production time


Forecasting
Estimated Downtime For the Next Day
Actual
Estimate
244
444
165
0
503
32
213
August 29, 2024
August 30, 2024
August 31, 2024
September 1, 20..
September 2, 20..
September 3, 20..
September 4, 20..
Estimated Number of Batchs For the Next Day
Actual
Estimate
7
12
7
0
11
1
7
August 29, 2024
August 30, 2024
August 31, 2024
September 1, 2024
September 2, 2024
September 3, 2024
September 4, 2024

Phase V

Recommendations

Recommendations.

Actionable Insights

- Predictive maintenance scheduling
- Process optimizations to reduce frequent downtime causes
- Future steps for model refinement and deployment

☑ 1. Project Summary (Recommendation Overview)

"This analysis identifies key contributors to manufacturing downtime, evaluates its impact on batch production, and builds a predictive model to optimize future performance. By understanding and forecasting downtime, we can improve scheduling, reduce inefficiencies, and maximize production output."

📊 2. Key Insights to Highlight

Include in our dashboard, report, or presentation:

- ☑ **Total number of unique products and batches**
- ☑ **Operator performance (duration, frequency, productivity)**
- ☑ **Most common downtime causes**
- ☑ **Total downtime trend over time**
- ☑ **Forecasted impact of downtime on batch output (Regression)**

💡 3. Recommendations for the Business

Area	Recommendation
🔧 **Downtime Causes**	Focus on the **top 3 downtime factors** — improve SOPs, maintenance schedules, or training around them.
👥 **Operator Optimization**	Cross-train underperforming operators or **reassign operators** based on historical productivity.
🗓️ **Batch Forecasting**	Use the **linear regression model** to plan production based on expected maintenance or shift availability.
📈 **KPI Tracking**	Implement a dashboard that tracks **daily downtime**, **batch count**, and **operator performance** in real time.
🧠 **Preventive Maintenance**	Use the downtime data to **schedule preventive maintenance** during low-production hours.

1. Address Operator-Driven Downtime (56% of total)

a. Reduce Machine Adjustment Time (332 mins – 24%)

- **Recommendation: Standardize machine settings with SOPs for each product.**
- **Action: Implement checklists or digital settings presets.**
- **Impact: Reduces variability and retraining needs.**

b. Improve Training for Batch Changes & Coding (305 mins)

- **Recommendation: Conduct targeted re-training for operators on batch changeover and coding protocols.**
- **Action: Develop quick-reference guides and conduct root cause reviews of past errors.**
- **Impact: Reduces avoidable errors and setup times.**

⚙️ **2. Tackle Major Non-Operator Issues (44% of total)**

a. Prevent Machine Failures (254 mins)

- **Recommendation: Strengthen preventive maintenance schedules.**
- **Action: Use sensors/logs to predict failures and schedule timely maintenance.**
- **Impact: Minimizes unexpected downtime.**

b. Resolve Inventory Shortages (225 mins)

- **Recommendation: Integrate real-time inventory monitoring with alerts.**
- **Action: Implement a monitoring system or improve coordination with supply chain.**
- **Impact: Ensures materials are available before they run out.**

🎯 Final Presentation Tip

“By analyzing downtime patterns and operator performance, we uncovered opportunities to increase production by forecasting and reducing unnecessary stoppages. ”

🙏 Thanks & Acknowledgements

This project was completed as part of a data analysis initiative focused on improving manufacturing productivity by analyzing downtime and production data.

Special Thanks To:

- Rawad Misr Team for providing access to real-world manufacturing data.
- Mentors & Reviewers for valuable feedback and support.
- Pandas, Matplotlib, and scikit-learn open-source communities for the incredible tools that made this analysis possible.

"Data is the new oil, but it's what you do with it that truly fuels progress." 🚀

"Turning downtime into insights, and insights into action."

Crafted with code, passion, and purpose. 💻 💖
#DataDrivenDecisions