BeerBo Printing: Analysis Summary

1. Analysis Process

The analysis of BeerBo Printing's operational data involved several structured steps, including:
- **Data cleaning:** Outlier detection (via IQR), zero-value handling, and missing value imputation.
- **Downtime analysis:** Examining both planned and unplanned stop times, their causes, and line-level differences.
- **Quality and production review:** Calculating reject rates, identifying common reject reasons, and evaluating line-wise throughput.
- **Team-level performance:** Comparing average unplanned downtime and reject rates among different teams.
- **Correlation checks**: Exploring the relationship between unplanned downtime and production rejects.

2. Key Findings

• Outliers & Runtime Distribution:

- 87 runtime outliers were detected and either removed or capped. A significant number of near-zero runtime entries were adjusted using median imputation (≈630.6s).
- After cleanup, the runtime distribution became unimodal and more stable for analysis.

• Downtime Overview:

- Total downtime was 615,516 seconds, with unplanned downtime making up 76.48% (470,738s).
- Median unplanned stop time was 0s across all lines, but Line 4 had the highest mean (41.85s) and widest range (468.44s).

• Root Causes of Unplanned Downtime:

- 'Main B Bus Undervoltage' and 'Electrical Fault' were the most frequent causes, each with over 400 occurrences.
- A Pareto analysis revealed that the top 3 causes contributed to over 50% of all unplanned stops.

• Production & Rejects:

- Overall reject rate was 4.24%, calculated from 121,933 rejects out of 2,873,891 total units.
- Most common reject reasons included 'Detected by Max WIP', 'Cosmetic Defect', and 'Out of Spec'.

• Efficiency Across Devices:

- Line 1 had the highest production efficiency at ~1300 units/hour, compared to ~1000 units/hour on Lines 3 and 4.

• Correlation Analysis:

- Pearson correlation between unplanned stop time and reject count was negligible ($r = 0.01$, $p = 0.9120$).

• Team-Level Performance:

- Team 1 outperformed others with the lowest reject rate (4.20%) despite similar average downtime (~36s) across all teams.

3. Operational Implications

This analysis reveals that unplanned downtime is a major driver of inefficiency at BeerBo Printing. Addressing a small number of high-impact failure types could significantly reduce downtime. Production efficiency varies by line, suggesting optimization opportunities through equipment upgrades or standardization. Rejects seem more linked to process or material issues rather than downtime itself, highlighting the need for better inspection and quality control systems. Team 1's performance offers a benchmark for operational consistency that other teams might emulate through training and procedural alignment.

Code:

SQL code to join ProductionMetric and Quality datasets

```
SELECT

    pm.prodmetric_stream_key,

    pm.deviceKey AS production_device,

    pm.start_time,

    pm.end_time,

    pm.good_count,

    pm.reject_count AS production_reject_count,

    pm.ideal_time,

    pm.run_time,

    pm.unplanned_stop_time,
```

```sql
    pm.planned_stop_time,
    pm.performance_impact_display_name,
    pm.process_state_display_name,
    pm.process_state_reason_display_name,
    pm.job_display_name,
    pm.part_display_name,
    pm.shift_display_name,
    pm.team_display_name,
    q.quality_stream_key,
    q.deviceKey AS quality_device,
    q.count AS quality_count,
    q.reject_reason_display_name
FROM
    ProductionMetric pm
LEFT JOIN
    Quality q ON pm.prodmetric_stream_key = q.prodmetric_stream_key
ORDER BY
    pm.start_time DESC;
```

```python
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
import seaborn as sns
# Load datasets
device_prop = pd.read_csv('DeviceProperty.csv')
prod_metric = pd.read_csv('ProductionMetric.csv')
```

```
quality = pd.read_csv('Quality.csv')

# Check for missing values in all datasets

device_prop_missing = device_prop.isnull().sum()

prod_metric_missing = prod_metric.isnull().sum()

quality_missing = quality.isnull().sum()


# Display missing values for each dataset

print("Device Property Missing Values:\n", device_prop_missing)

print("\nProduction Metric Missing Values:\n", prod_metric_missing)

print("\nQuality Missing Values:\n", quality_missing)
# Check for duplicates in each dataset

device_prop_duplicates = device_prop.duplicated().sum()

prod_metric_duplicates = prod_metric.duplicated().sum()

quality_duplicates = quality.duplicated().sum()


# Print the number of duplicates

print(f"Device Property Duplicates: {device_prop_duplicates}")

print(f"Production Metric Duplicates: {prod_metric_duplicates}")

print(f"Quality Duplicates: {quality_duplicates}")
# data_df is the dataset which obtained from SQL code

BeerBo_df = pd.read_csv('data_df.csv')

BeerBo_df

# Check for outliers in 'run_time' column using IQR (Interquartile Range)

Q1 = BeerBo_df['run_time'].quantile(0.25)

Q3 = BeerBo_df['run_time'].quantile(0.75)

IQR = Q3 - Q1

outliers = BeerBo_df[(BeerBo_df['run_time'] < (Q1 - 1.5 * IQR)) | (BeerBo_df['run_time'] > (Q3
+ 1.5 * IQR))]

print(f"Outliers in run_time:\n{outliers}")

outliers_details = BeerBo_df.loc[outliers.index]
```

```
print(outliers_details[['production_device', 'start_time', 'end_time', 'run_time']])

# Remove outliers

cleaned_prod_metric = BeerBo_df[BeerBo_df['run_time'] <= (Q3 + 1.5 * IQR)]

# Cap the run_time values at the 95th percentile

cap_value = BeerBo_df['run_time'].quantile(0.95)

BeerBo_df['run_time'] = np.where(BeerBo_df['run_time'] > cap_value, cap_value,
BeerBo_df['run_time'])

# Visualize the distribution of 'run_time' after removing or capping outliers

sns.histplot(BeerBo_df['run_time'], kde=True)

plt.title("Distribution of Run Time After Outlier Treatment")

plt.show()


# log(1 + x) to avoid log(0) issues
BeerBo_df['log_run_time'] = np.log1p(BeerBo_df['run_time'])

# Count zero or near-zero values

zero_counts = BeerBo_df[BeerBo_df['run_time'] <= 5].shape[0]

total_counts = BeerBo_df.shape[0]


print(f"Number of zero or near-zero values: {zero_counts}")

print(f"Percentage of zero or near-zero values: {zero_counts / total_counts * 100:.2f}%")

# Check zero run_time across different devices

zero_counts_by_device = BeerBo_df[BeerBo_df['run_time'] <=
5]['production_device'].value_counts()

print(zero_counts_by_device)


# Check zero run_time across different shifts

zero_counts_by_shift = BeerBo_df[BeerBo_df['run_time'] <=
5]['shift_display_name'].value_counts()

print(zero_counts_by_shift)

# Check run_time stats for each line
```

```python
BeerBo_df.groupby("production_device")["run_time"].describe()

# Total runs per shift
total_counts_by_shift = BeerBo_df['shift_display_name'].value_counts()


# Compute zero percentage
zero_percentage_by_shift = (zero_counts_by_shift / total_counts_by_shift) * 100
print(zero_percentage_by_shift)

# Impute with median runtime
median_run_time = BeerBo_df[BeerBo_df["run_time"] > 5]["run_time"].median()
BeerBo_df.loc[BeerBo_df["run_time"] <= 5, "run_time"] = median_run_time

# Check number of missing shift values
print(BeerBo_df['shift_display_name'].isnull().sum())


# Check if 'No Shift' and 'Unknown Shift' only contain zero run times
no_shift_data = BeerBo_df[BeerBo_df['shift_display_name'].isin(['No Shift', 'Unknown Shift'])]
print(no_shift_data)
print(no_shift_data['run_time'].describe())

BeerBo_df = BeerBo_df[~BeerBo_df['team_display_name'].isin(['Unknown Team', 'No Team'])]
plt.figure(figsize=(6,4))
sns.histplot(BeerBo_df["run_time"], bins=30, kde=True)
plt.xlabel("Run Time After Cleaning")
plt.title("Updated Distribution After Handling Zeros")
plt.show()

# Step 1: Total values
total_unplanned = BeerBo_df['unplanned_stop_time'].sum()
total_planned = BeerBo_df['planned_stop_time'].sum()
total_downtime = total_unplanned + total_planned
```

```
# Step 2: Proportions
unplanned_prop = total_unplanned / total_downtime
planned_prop = total_planned / total_downtime


# Step 3: Display
print(f"Total Unplanned Downtime: {total_unplanned}")
print(f"Total Planned Downtime: {total_planned}")
print(f"Total Downtime: {total_downtime}\n")


print(f"Proportion of Unplanned Downtime: {unplanned_prop:.2%}")
print(f"Proportion of Planned Downtime: {planned_prop:.2%}")
# Grouping by 'deviceKey' and summarizing unplanned and planned stop times
downtime_summary = BeerBo_df.groupby('production_device')[['unplanned_stop_time',
'planned_stop_time']].agg(
    ['mean', 'median', 'std', 'min', 'max']
)


# Flatten multi-level columns
downtime_summary.columns = ['_'.join(col).strip() for col in
downtime_summary.columns.values]
downtime_summary.reset_index(inplace=True)


# Calculate range
downtime_summary['unplanned_range'] = downtime_summary['unplanned_stop_time_max'] -
downtime_summary['unplanned_stop_time_min']
downtime_summary['planned_range'] = downtime_summary['planned_stop_time_max'] -
downtime_summary['planned_stop_time_min']


# Melt for boxplot
```

```
melted_df = BeerBo_df[['production_device', 'unplanned_stop_time', 'planned_stop_time']].melt(
    id_vars='production_device', var_name='Downtime Type', value_name='Duration'
)

# Set up the figure
plt.figure(figsize=(14, 6))
sns.boxplot(data=melted_df, x='production_device', y='Duration', hue='Downtime Type')
plt.title('Boxplot of Downtime Duration per DeviceKey')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

downtime_summary.head()
# Filter only rows with non-zero unplanned_stop_time
unplanned_df = BeerBo_df[BeerBo_df['unplanned_stop_time'] > 0]
# Count frequency
reason_counts =
unplanned_df['process_state_reason_display_name'].value_counts().reset_index()
reason_counts.columns = ['process_state_reason_display_name', 'frequency']
# Sort by frequency
reason_counts = reason_counts.sort_values(by='frequency', ascending=False)

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(x='process_state_reason_display_name', y='frequency', data=reason_counts.head(10),
palette='viridis')
plt.xticks(rotation=45, ha='right')
plt.title('Top 10 Reasons for Unplanned Downtime')
plt.xlabel('Process State Reason')
```

```
plt.ylabel('Frequency')

plt.tight_layout()

plt.show()

# Step 3: Sort by frequency and calculate cumulative %

reason_counts = reason_counts.sort_values(by='frequency', ascending=False)

reason_counts['cum_percentage'] = reason_counts['frequency'].cumsum() /
reason_counts['frequency'].sum() * 100


# Step 4: Plot

plt.figure(figsize=(14, 7))


# Bar plot

sns.barplot(x='process_state_reason_display_name', y='frequency', data=reason_counts.head(10),
color='skyblue')


# Cumulative % line (on secondary y-axis)

ax2 = plt.twinx()

ax2.plot(reason_counts['process_state_reason_display_name'].head(10),

    reason_counts['cum_percentage'].head(10), color='red', marker='o', linewidth=2)

ax2.set_ylabel('Cumulative Percentage (%)')


# Labels and styling

plt.xticks(rotation=45, ha='right')

plt.title('Pareto Chart: Reasons for Unplanned Downtime')

plt.xlabel('Process State Reason')

plt.ylabel('Frequency')

plt.grid(True, which='both', axis='y', linestyle='--', alpha=0.5)

plt.tight_layout()
```

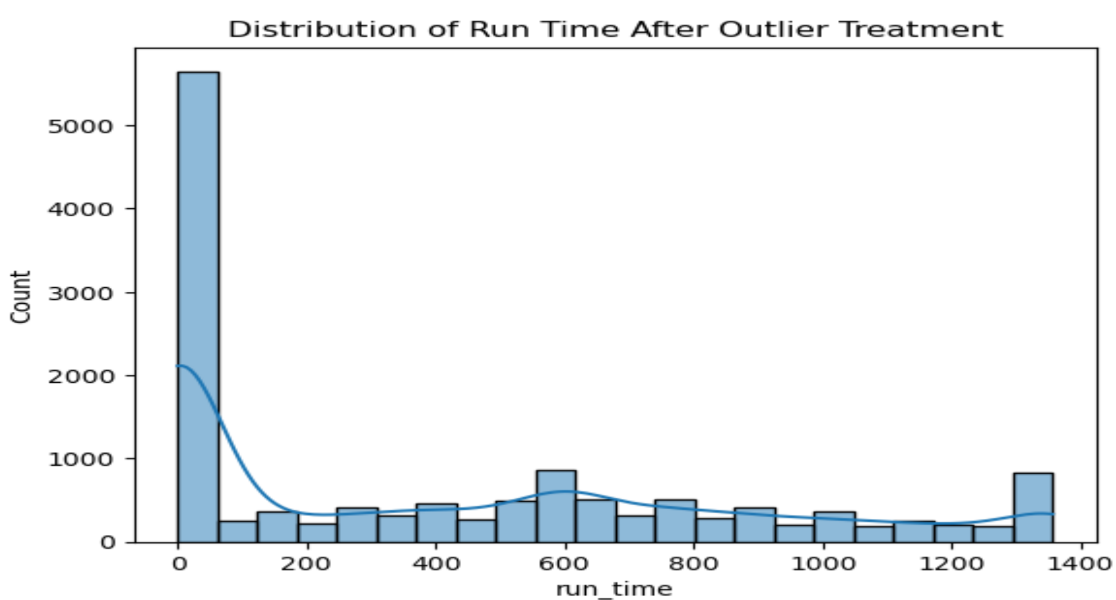
```python
plt.show()
# Sum up the total good and reject counts
total_good_count = BeerBo_df['good_count'].sum()
total_reject_count = BeerBo_df['production_reject_count'].sum()

# Calculate reject rate
reject_rate = total_reject_count / (total_good_count + total_reject_count)
reject_rate_percentage = reject_rate * 100

print(f"Total Good Count: {total_good_count}")
print(f"Total Reject Count: {total_reject_count}")
print(f"Overall Reject Rate: {reject_rate_percentage:.2f}%")
# Group by reject_reason_display_name and count
reason_counts = BeerBo_df['reject_reason_display_name'].value_counts().reset_index()
reason_counts.columns = ['reject_reason_display_name', 'count']

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(data=reason_counts, x='reject_reason_display_name', y='count', palette='mako')
plt.xticks(rotation=45, ha='right')
plt.title('Most Common Reject Reasons')
plt.xlabel('Reject Reason')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
# Group by deviceKey and aggregate good_count and run_time
device_efficiency = BeerBo_df.groupby('production_device').agg({
    'good_count': 'sum',
```

```
    'run_time': 'sum'
}).reset_index()


# Calculate good count per hour
device_efficiency['good_per_hour'] = device_efficiency['good_count'] /
(device_efficiency['run_time'] / 3600)


# Sort by performance
device_efficiency = device_efficiency.sort_values(by='good_per_hour', ascending=False)


# Plotting
plt.figure(figsize=(12, 6))
sns.barplot(data=device_efficiency, x='production_device', y='good_per_hour', palette='viridis')
plt.title('Average Good Count per Hour of Run Time by Device')
plt.xlabel('Device Key')
plt.ylabel('Good Count per Hour')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
# Filter relevant rows
relevant_data = BeerBo_df[(BeerBo_df['unplanned_stop_time'] > 0) &
(BeerBo_df['production_reject_count'] > 0)]


# Compute Pearson correlation
corr, p_value = pearsonr(relevant_data['unplanned_stop_time'],
relevant_data['production_reject_count'])
print(f"Pearson Correlation: {corr:.2f} (p-value: {p_value:.4f})")


# Plotting
plt.figure(figsize=(10, 6))
```

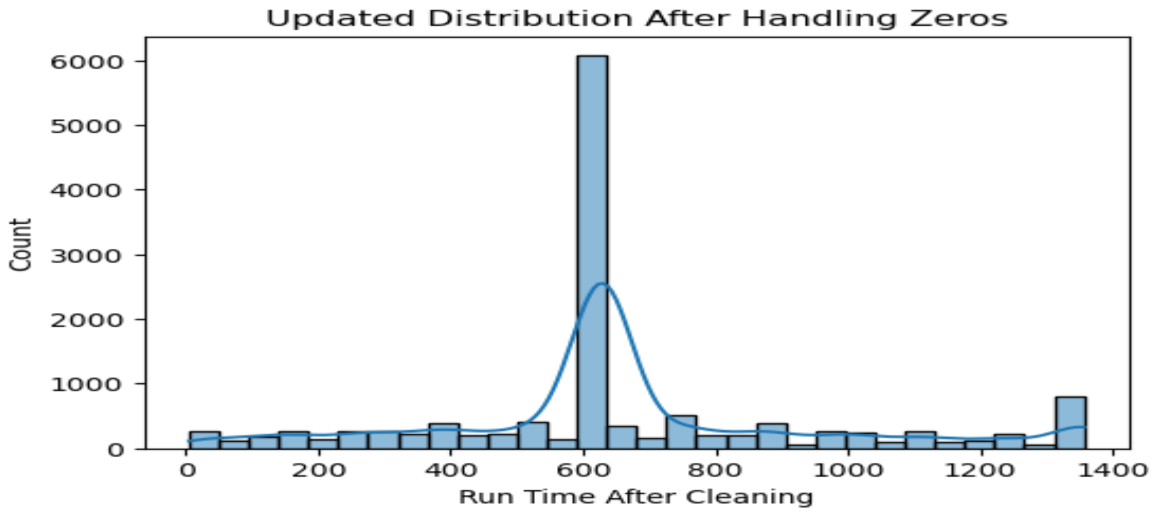
```
sns.scatterplot(data=relevant_data, x='unplanned_stop_time', y='production_reject_count',
alpha=0.5)

sns.regplot(data=relevant_data, x='unplanned_stop_time', y='production_reject_count',
scatter=False, color='red', label='Trend Line')

plt.title('Correlation Between Unplanned Downtime and Reject Count')

plt.xlabel('Unplanned Stop Time (seconds)')

plt.ylabel('Reject Count')

plt.legend()

plt.tight_layout()

plt.show()

team_metrics = BeerBo_df.groupby('team_display_name').agg({

    'unplanned_stop_time': 'mean',

    'good_count': 'sum',

    'production_reject_count': 'sum'

}).reset_index()


team_metrics['reject_rate'] = (team_metrics['production_reject_count'] /

                                (team_metrics['good_count'] + team_metrics['production_reject_count'])) *
100


team_metrics.rename(columns={

    'unplanned_stop_time': 'avg_unplanned_downtime'

}, inplace=True)


# Plotting

fig, ax1 = plt.subplots(figsize=(12, 6))

sns.barplot(data=team_metrics, x='team_display_name', y='avg_unplanned_downtime',
color='skyblue', ax=ax1)

ax1.set_ylabel('Avg Downtime (seconds)', color='blue')

ax1.set_title('Performance Comparison Across Teams')
```

```
ax2 = ax1.twinx()
sns.lineplot(data=team_metrics, x='team_display_name', y='reject_rate',
             marker='o', color='red', ax=ax2)
ax2.set_ylabel('Reject Rate (%)', color='red')

plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

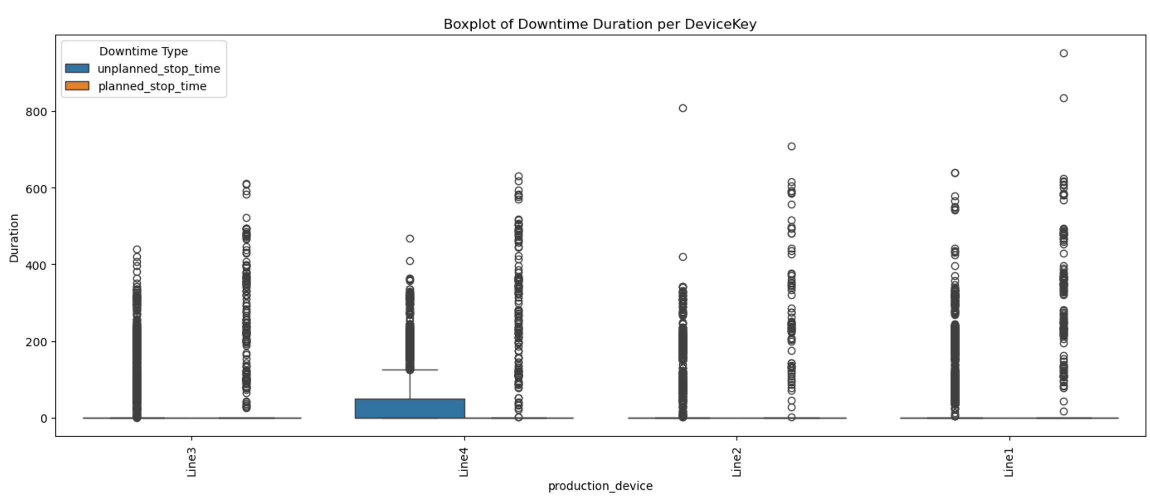
Visualizations:



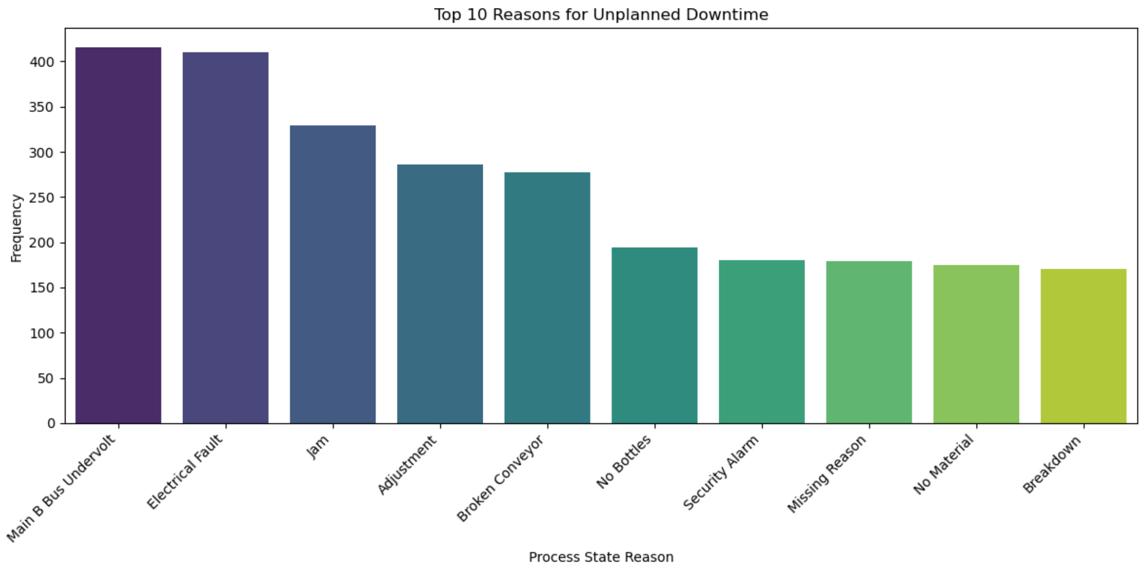
- A histogram with a kernel density estimate (KDE) line shows the **distribution of run_time after treatment**.
- The chart indicates a **heavy concentration of runs below 200 seconds**, with a long tail tapering toward the right, typical of production run-time data.



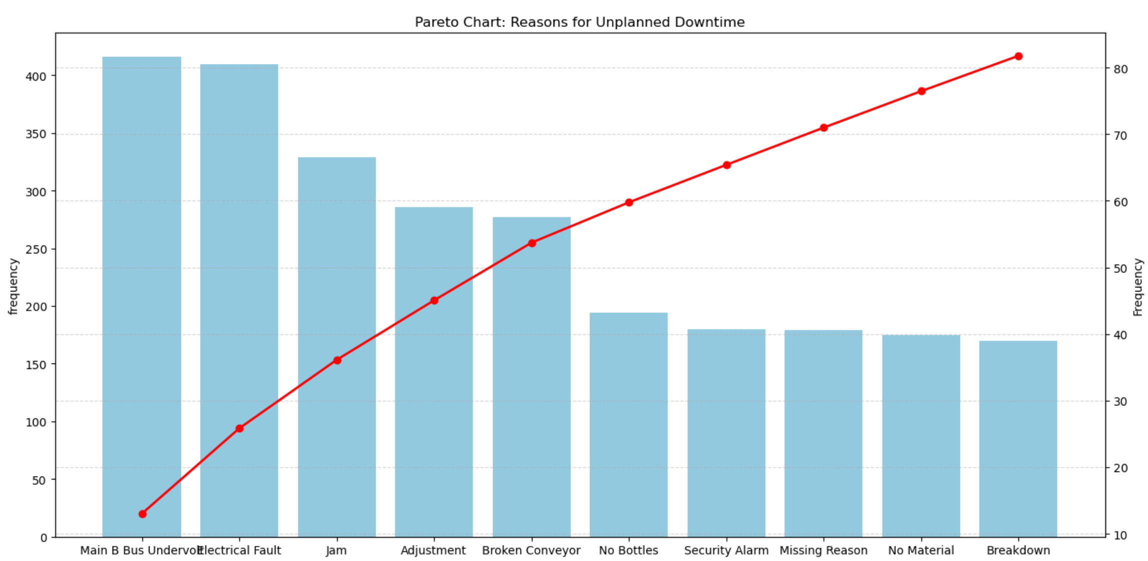
- A **histogram with KDE** shows the **post-cleaning distribution of run_time**.
- The updated distribution reveals a strong **central peak around the imputed median (≈ 630 seconds)**.
- Minor tails exist on either side, but the distribution is now **more symmetric and statistically stable**.



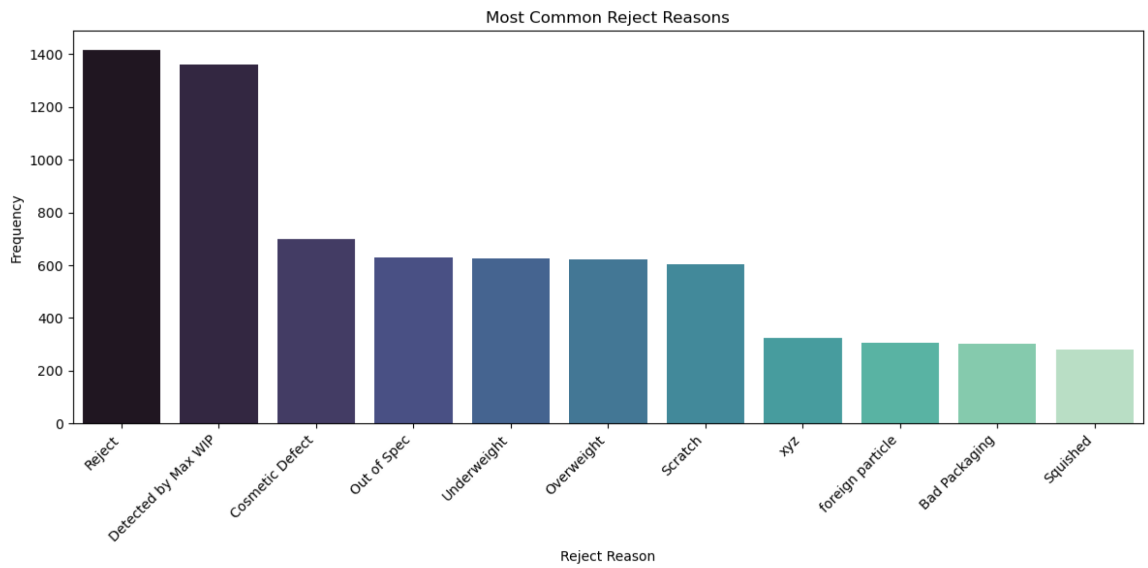
- All lines experienced **more unplanned than planned downtime**.
- **Line 4** showed the widest range in unplanned stops with more extreme values.
- Outliers were common across all lines, reflecting sporadic long interruptions.



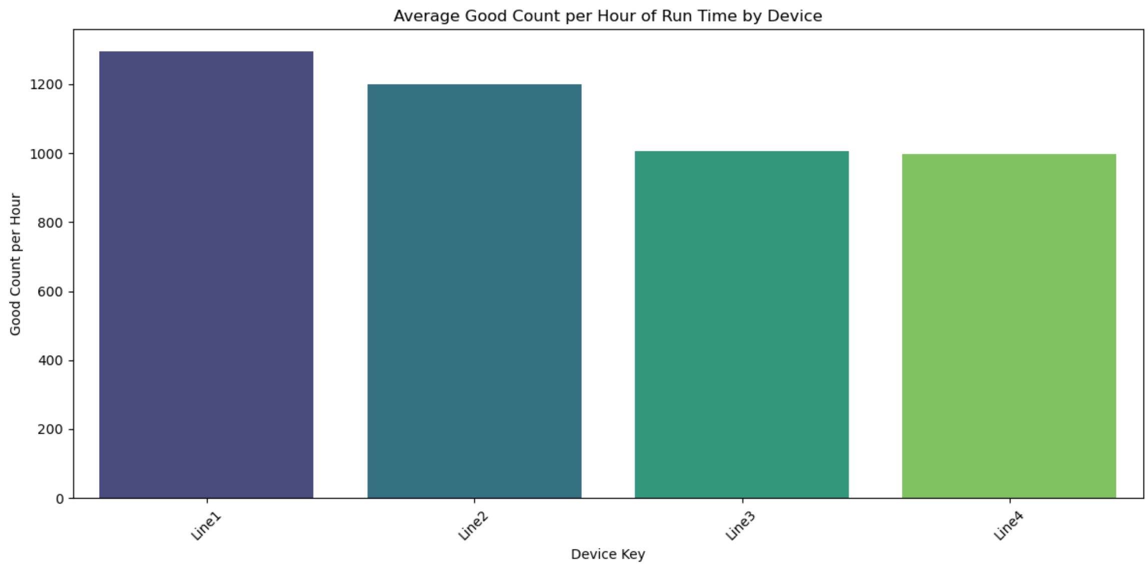
- Electrical and mechanical failures (**Main B Bus Undervoltage**, **Electrical Fault**, **Jam**) were dominant.
- Operational issues like **No Bottles** and **No Material** also contributed significantly.
- These top 10 reasons account for a **large proportion of total unplanned downtime**, offering a clear focus for targeted intervention (e.g., better preventive maintenance, material flow assurance, or automation alerts).



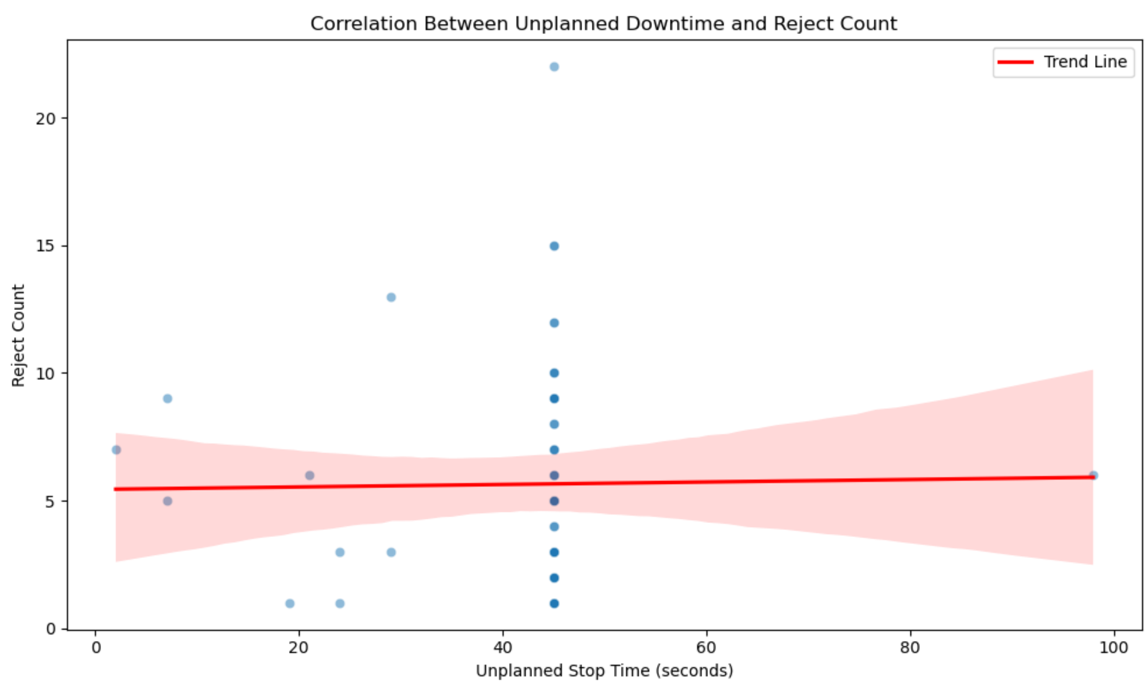
- Just **3 issues**—**Main B Bus Undervoltage**, **Electrical Fault**, and **Jam**—accounted for over **50% of all unplanned downtime events**.
- The **cumulative contribution** of the top 10 causes exceeds **80%**, validating the **Pareto principle (80/20 rule)** in this context.



- A 4.24% reject rate is **noteworthy** and represents a significant loss in yield.
- Several reasons (e.g., "Detected by Max WIP" and "Out of Spec") suggest issues with either upstream process control or end-of-line inspection accuracy.
- Addressing just the top 3 reject causes could potentially **halve the defect rate**, improving operational efficiency and reducing rework or scrap costs.

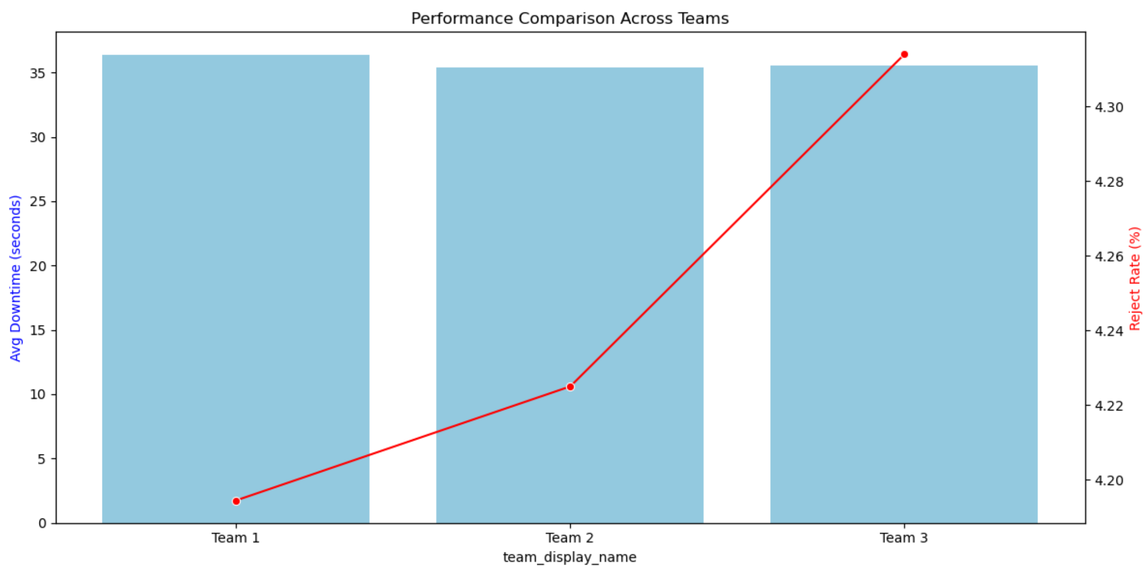


- **Line 1 consistently outperformed** all other lines in throughput efficiency.
- **Lines 3 and 4**, while operational, had noticeably lower output per hour, aligning with earlier findings about higher downtime or zero-runtime issues on those lines.



- **Pearson Correlation Coefficient: 0.01**
- **p-value: 0.9120**
- The correlation is **extremely weak and statistically insignificant**.

- The trend line is nearly flat, indicating **no meaningful linear relationship** between how long a line is stopped unexpectedly and the number of rejects produced in that run.



- Downtime was **relatively consistent** across all teams.
- **Team 3 had the highest reject rate**, despite similar downtime to others.
- **Team 1 demonstrated best overall performance**, combining low rejects with average downtime.