Exercise3_Group1

February 19, 2025

1 Exercise 3: Data-Driven Computing Architectures

In this exercise, we will work with Delta tables and the Medallion Architecture. You can gain 15 points, which will be awarded based on how effectively you implement the Bronze, Silver, and Gold layers, the quality of your data transformations and analysis, and how well your visualizations communicate insights. A well-designed pipeline should also allow new files to be uploaded and processed smoothly without requiring major modifications.

Make sure to include clear explanations of what you did and why throughout your report and visualizations. Grading will also consider how well you justify your choices, so do not just present results but explain your reasoning.

Useful links: - https://docs.delta.io/latest/index.html - https://delta.io/blog/delta-lake-medallion-architecture/

1.1 Scenario

You're a Data Engineer at a manufacturing company that produces industrial components. The factory runs 10 specialized machines, producing drill bits, gears, shafts, conveyor belts, turbine parts, robot components, stamped metal, polished surfaces, laser-cut materials, and 3D-printed prototypes.

Each machine is equipped with sensors to track performance, while production logs record output and defects, and maintenance records document repairs. A team of production operators manages manufacturing, while maintenance operators handle scheduled and emergency repairs.

Recently, management has raised concerns about machine efficiency, defect rates, and maintenance costs, and they want continuous data-driven insights to improve operations. You have been given raw data from three sources:

- **Sensor Data:** Real-time readings from industrial machines.
- **Production Logs:** Daily records of production output and defects.
- **Maintenance Records:** Logs of scheduled and emergency maintenance events.

1.2 Your Assignment

Your task is to build a Medallion Architecture pipeline using Delta Lake to clean, structure, and analyze this data. 1. **Ingest the raw data** into **Bronze Layer** Delta tables.
2. **Clean and standardize the data** in the **Silver Layer**.

3. **Aggregate and generate business insights** in the **Gold Layer**.
4. **Visualize key metrics** to make informed decisions, using for example Matplotlib or Seaborn.

1.3 Data Description

The csv files are available in the `/shared` folder in Noppe. ## **1. Sensor Data**
Captures real-time sensor readings from machines, tracking temperature, vibration, power consumption, and operational status.

1.3.1 Key Fields:

- `sensor_id`: Unique identifier for the sensor.

- `machine`: Name of the machine.
- `time_stamp`: Timestamp of the sensor reading.

- `temp C`: Temperature reading (°C).

- `vibration_lvl`: Vibration level reading.

- `power_kW`: Power consumption (kW).

- `def_ct`: Number of defective sensor readings.

- `status_flag`: Operational status (e.g., "Running", "Stopped").

- `noise_val`: Random noise factor in the data.

- `extra_param`: Additional machine-related parameter.

1.4 2. Production Logs

Tracks machine output, defect rates, and operator activity. Machines in poor condition tend to produce more defects.

1.4.1 Key Fields:

- `log_id`: Unique identifier for the production log.

- `product_type`: Type of product produced.

- `units_produced`: Number of units produced.

- `defective_units`: Number of defective units.

- `time_stamp`: Timestamp of production record.
- `machine`: Machine responsible for production.
- `operator_id`: Identifier of the operator overseeing production.
- `remarks`: Additional notes about the production process (e.g., quality concerns, machine adjustments).
- `batch_info`: Batch identifier for tracking specific production runs.

1.5 3. Maintenance Records

Logs maintenance activities, including scheduled upkeep, emergency repairs, and associated costs. Machines in poor condition require more frequent emergency maintenance.

1.5.1 Key Fields:

- `maintenance_id`: Unique identifier for the maintenance event.
- `machine`: Machine undergoing maintenance.
- `maintenance_date`: Timestamp of maintenance event.
- `maintenance_type`: Type of maintenance (Scheduled, Unscheduled, Emergency).
- `duration_minutes`: Length of the maintenance event.
- `cost`: Cost of the maintenance.
- `operator`: Identifier of the maintenance operator performing the task.
- `notes`: Description of the maintenance issue or action taken.

1.6 4. Operator Dimension Table (Predefined Silver Table)

A reference table with details about production and maintenance operators.

1.6.1 Key Fields:

- `operator_id`: Unique identifier for the operator.
- `operator_name`: Full name of the operator.
- `operator_type`: "Production" or "Maintenance".

1.7 Medallion Architecture Implementation

1.7.1 1. Bronze Layer – Raw Data Storage (2p)

- Ingest raw data as-is into Delta tables.
- No transformations at this stage.

1.7.2 2. Silver Layer – Cleaning & Standardization (3p)

For example: - Convert columns into a proper format.
- Rename columns for consistency, for example (`time_stamp` → `timestamp`).
- Remove duplicate records.

1.7.3 3. Gold Layer – Business Insights (4p)

For example: - Aggregate sensor, production, and maintenance data to create daily machine performance metrics.
- Join tables to uncover correlations between sensor readings, defects, and maintenance events.
- Visualize the tables and metrics. —

1.8 Example Visualizations:

1.8.1 Daily Sensor Metrics

- Average temperature, vibration, and power consumption per machine.

- Number of downtime events (`status_flag = "Stopped"`).

1.8.2 Daily Production Metrics

- Total units produced and defective units per machine.

- Production yield: (total_units_produced - defective_units) / total_units_produced

1.8.3 Daily Maintenance Metrics

- Number of maintenance events per machine.

- Total maintenance costs per machine.

1.8.4 Advanced Insights

- Correlation analysis between high vibration levels and production defects.

- Identify the most frequent operator per machine per day.

- Estimate energy consumption trends over time.

You are encouraged to explore and define your own insights.

1.9 Example Directory Structure

```
data_lake/
  bronze/
      sensor_data_bronze/          # Raw Sensor Data (Delta table)
      production_data_bronze/      # Raw Production Data (Delta table)
      maintenance_data_bronze/     # Raw Maintenance Data (Delta table)
  silver/
      sensor_data_silver/          # Cleaned Sensor Data (Delta table)
      production_data_silver/      # Cleaned Production Data (Delta table)
      maintenance_data_silver/     # Cleaned Maintenance Data (Delta table)
      dim_operator_silver/         # Operator Dimension Table (Delta table)
  gold/
    gold_machine_performance/  # Aggregated Machine Performance (Delta table)
```

---

[1]:
```
pip install delta-spark==3.0.0
```

```
Collecting delta-spark==3.0.0
  Downloading delta_spark-3.0.0-py3-none-any.whl.metadata (2.0 kB)
Requirement already satisfied: pyspark<3.6.0,>=3.5.0 in /usr/local/spark/python
(from delta-spark==3.0.0) (3.5.1)
Requirement already satisfied: importlib-metadata>=1.0.0 in
/opt/conda/lib/python3.11/site-packages (from delta-spark==3.0.0) (7.1.0)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.11/site-
packages (from importlib-metadata>=1.0.0->delta-spark==3.0.0) (3.17.0)
Collecting py4j==0.10.9.7 (from pyspark<3.6.0,>=3.5.0->delta-spark==3.0.0)
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl.metadata (1.5 kB)
Downloading delta_spark-3.0.0-py3-none-any.whl (21 kB)
Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
                         200.5/200.5 kB
5.5 MB/s eta 0:00:00a 0:00:01
Installing collected packages: py4j, delta-spark
Successfully installed delta-spark-3.0.0 py4j-0.10.9.7
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```python
from pyspark.sql import SparkSession
from delta import configure_spark_with_delta_pip

# Configure the Spark session with Delta support
builder = SparkSession.builder \
    .appName("Exercise1") \
    .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
    .config("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.
 ↪catalog.DeltaCatalog") \
    .config("spark.jars.packages", "io.delta:delta-core_2.12:3.0.0")

# Create the Spark session
```

```
spark = configure_spark_with_delta_pip(builder).getOrCreate()

print("Spark session with Delta Lake configured successfully!")
spark
```

```
Spark session with Delta Lake configured successfully!
```

[2]:
```
<pyspark.sql.session.SparkSession at 0x7fa939709550>
```

1.10 Imports

[3]:
```
from pyspark.sql.functions import col, sum, to_date, avg, count, when, isnan
from pyspark.sql.types import DoubleType, IntegerType, TimestampType
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

1.11 1. Bronze Layer – Raw Data Storage

1.11.1 Sensor_data

[4]:
```
sensor_data_path = "sensor_data.csv"

# Read the raw sensor data from CSV
sensor_df = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load(sensor_data_path)

# Whitespaces in the columns names were giving error -> all spaces were changed␣
↪with underlines
sensor_df = sensor_df.select([col(c).alias(c.replace(" ", "_")) for c in␣
↪sensor_df.columns])

sensor_df.show(5)
sensor_df.printSchema()
```

```
+--------------------+--------------+-------------------+------+-------------+--
------+------+-----------+---------+-----------+
|           sensor_id|       machine|         time_stamp|temp_C|vibration_lvl|po
wer_kW|def_ct|status_flag|noise_val|extra_param|
+--------------------+--------------+-------------------+------+-------------+--
------+------+-----------+---------+-----------+
|e0310580-9228-434…|DrillPress-100|2024-10-01 00:00:00|  48.1|         1.03|
20.9|     0|    Stopped|    0.505|        103|
|496a0d42-dd46-441…|  CNC-Mill-200|2024-10-01 00:02:00|  48.0|          0.9|
19.25|     0|    Stopped|    0.716|        189|
```

```
|e4bf3234-33f4-42d…|     Lathe-300|2024-10-01 00:04:00| 45.23|         1.07|
22.06|     0|    Stopped|     0.22|        175|
|aeb1a7d1-0f1c-48d…|  Conveyor-400|2024-10-01 00:06:00| 52.82|         1.81|
18.92|     0|    Stopped|    0.869|        197|
|f38f4927-e44b-447…|   Turbine-500|2024-10-01 00:08:00| 52.57|         1.94|
22.67|     0|    Stopped|    0.698|        143|
+--------------------+--------------+-------------------+------+-------------+--
------+------+-----------+---------+-----------+
only showing top 5 rows

root
 |-- sensor_id: string (nullable = true)
 |-- machine: string (nullable = true)
 |-- time_stamp: string (nullable = true)
 |-- temp_C: string (nullable = true)
 |-- vibration_lvl: string (nullable = true)
 |-- power_kW: string (nullable = true)
 |-- def_ct: string (nullable = true)
 |-- status_flag: string (nullable = true)
 |-- noise_val: string (nullable = true)
 |-- extra_param: string (nullable = true)
```

[5]:
```python
# Write raw data to Bronze Layer as a Delta table
bronze_sensor_path = "data_lake/bronze/sensor_data_bronze/"
os.makedirs(bronze_sensor_path, exist_ok=True)

sensor_df.write.format("delta") \
    .mode("append") \
    .save(bronze_sensor_path)

print("Raw sensor data successfully ingested into the Bronze Layer.")
bronze_sensor_df = spark.read.format("delta").load(bronze_sensor_path)
```

```
Raw sensor data successfully ingested into the Bronze Layer.
```

1.11.2 Production_data

[6]:
```python
production_data_path = "production_data.csv"

# Read the raw production data from CSV
production_df = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load(production_data_path)

# Whitespaces in the columns names were giving error -> all spaces were changed
 ↪with underlines
```

```
production_df = production_df.select([col(c).alias(c.replace(" ", "_")) for c␣
 ↪in production_df.columns])

production_df.show(5)
production_df.printSchema()
```

```
+--------------------+---------------+--------------+--------------+----------
---------+--------------+--------------------+------------+--------------------+
|              log_id|   product_type|units_produced|defective_units|
time_stamp|   MachineName|             remarks|operator_id|          batch_info|
+--------------------+---------------+--------------+--------------+----------
---------+--------------+--------------------+------------+--------------------+
|831bdeda-388c-4f3…|     Turbine Hub|            96|               5|2024-10-01
10:05:00|   Turbine-500|Minor delays due …|       OP17|Batch-Turbine-500…|
|41274f57-82a0-4a4…|Polished Surface|            87|               4|2024-10-01
11:59:00|   Grinder-800|Normal operations…|       OP14|Batch-Grinder-800…|
|7a943af7-dc35-4e7…|       Drill Bit|           137|               8|2024-10-01
21:49:00|DrillPress-100|Slight quality co…|       OP20|Batch-DrillPress-…|
|5365b96b-62e9-470…|       Drill Bit|           135|               7|2024-10-01
05:27:00|DrillPress-100|Normal operations…|       OP10|Batch-DrillPress-…|
|e49c5179-038f-40f…|     Motor Shaft|           122|               6|2024-10-01
04:18:00|  CNC-Mill-200|Slight quality co…|       OP10|Batch-CNC-Mill-20…|
+--------------------+---------------+--------------+--------------+----------
---------+--------------+--------------------+------------+--------------------+
only showing top 5 rows

root
 |-- log_id: string (nullable = true)
 |-- product_type: string (nullable = true)
 |-- units_produced: string (nullable = true)
 |-- defective_units: string (nullable = true)
 |-- time_stamp: string (nullable = true)
 |-- MachineName: string (nullable = true)
 |-- remarks: string (nullable = true)
 |-- operator_id: string (nullable = true)
 |-- batch_info: string (nullable = true)
```

[7]:
```
# Write raw data to Bronze Layer as a Delta table
bronze_production_path = "data_lake/bronze/production_data_bronze/"
os.makedirs(bronze_production_path, exist_ok=True)

production_df.write.format("delta") \
    .mode("append") \
    .save(bronze_production_path)

print("Raw production data successfully ingested into the Bronze Layer.")
```

```
bronze_production_df = spark.read.format("delta").load(bronze_production_path)
```

Raw production data successfully ingested into the Bronze Layer.

1.11.3 Maintenance Data

[8]:
```
maintenance_data_path = "maintenance_data.csv"

# Read the raw production data from CSV
maintenance_df = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load(maintenance_data_path)

# Whitespaces in the columns names were giving error -> all spaces were changed␣
↪with underlines
maintenance_df = maintenance_df.select([col(c).alias(c.replace(" ", "_")) for c␣
↪in maintenance_df.columns])

maintenance_df.show(5)
maintenance_df.printSchema()
```

```
+--------------------+--------------+-------------------+----------------+------
----------+-------+--------+--------------------+
|      maintenance_id|       machine|
maintenance_date|maintenance_type|duration_minutes|   cost|operator|
notes|
+--------------------+--------------+-------------------+----------------+------
----------+-------+--------+--------------------+
|e0c400c6-e7a8-4b4…|  Conveyor-400|2024-10-02 05:48:00|       Emergency|
140|1312.39|   MT104|Critical failure …|
|0f2cbacd-52d6-49a…|DrillPress-100|2024-10-03 03:46:00|     Unscheduled|
49| 263.19|   MT105|Replaced small co…|
|48ab287e-4e96-462…|  Conveyor-400|2024-10-03 18:15:00|       Scheduled|
66| 284.63|   MT101|Calibrated machin…|
|b170c9d2-c3fc-4c9…|  RobotArm-600|2024-10-03 03:04:00|       Scheduled|
91| 265.16|   MT103|Replaced worn-out…|
|4f6c00a5-5eaa-425…|  Conveyor-400|2024-10-04 13:04:00|       Scheduled|
99| 212.26|   MT102|Routine check-up …|
+--------------------+--------------+-------------------+----------------+------
----------+-------+--------+--------------------+
only showing top 5 rows

root
 |-- maintenance_id: string (nullable = true)
 |-- machine: string (nullable = true)
 |-- maintenance_date: timestamp (nullable = true)
```

```
 |-- maintenance_type: string (nullable = true)
 |-- duration_minutes: integer (nullable = true)
 |-- cost: double (nullable = true)
 |-- operator: string (nullable = true)
 |-- notes: string (nullable = true)
```

[9]:
```
# Write raw data to Bronze Layer as a Delta table
bronze_maintenance_path = "data_lake/bronze/maintenance_data_bronze/"
os.makedirs(bronze_maintenance_path, exist_ok=True)

maintenance_df.write.format("delta") \
    .mode("append") \
    .save(bronze_maintenance_path)

print("Raw maintenance data successfully ingested into the Bronze Layer.")
bronze_maintenance_df = spark.read.format("delta").load(bronze_maintenance_path)
```

```
Raw maintenance data successfully ingested into the Bronze Layer.
```

1.12 Silver Layer – Cleaning & Standardization

1.12.1 Sensor Data

Renaming Columns

[10]:
```
bronze_sensor_df.columns
```

[10]:
```
['sensor_id',
 'machine',
 'time_stamp',
 'temp_C',
 'vibration_lvl',
 'power_kW',
 'def_ct',
 'status_flag',
 'noise_val',
 'extra_param']
```

[11]:
```
silver_sensor_df = (
    bronze_sensor_df.withColumnRenamed("time_stamp", "timestamp")
                    .withColumnRenamed("temp_C", "temperature_celsius")
                    .withColumnRenamed("vibration_lvl", "vibration_level")
                    .withColumnRenamed("power_kW", "power_kw")
                    .withColumnRenamed("def_ct", "defect_count")
)
silver_sensor_df.columns
```

[11]:

```
['sensor_id',
 'machine',
 'timestamp',
 'temperature_celsius',
 'vibration_level',
 'power_kw',
 'defect_count',
 'status_flag',
 'noise_val',
 'extra_param']
```

Changing Types

[12]:

```python
silver_sensor_df.dtypes
```

[12]:

```
[('sensor_id', 'string'),
 ('machine', 'string'),
 ('timestamp', 'string'),
 ('temperature_celsius', 'string'),
 ('vibration_level', 'string'),
 ('power_kw', 'string'),
 ('defect_count', 'string'),
 ('status_flag', 'string'),
 ('noise_val', 'string'),
 ('extra_param', 'string')]
```

[13]:

```python
silver_sensor_df = silver_sensor_df \
    .withColumn("timestamp", col("timestamp").cast(TimestampType())) \
    .withColumn("temperature_celsius", col("temperature_celsius").
↪cast(DoubleType())) \
    .withColumn("vibration_level", col("vibration_level").cast(DoubleType())) \
    .withColumn("power_kw", col("power_kw").cast(DoubleType())) \
    .withColumn("defect_count", col("defect_count").cast(IntegerType())) \
    .withColumn("noise_val", col("noise_val").cast(DoubleType())) \
    .withColumn("extra_param", col("extra_param").cast(DoubleType()))

silver_sensor_df.dtypes
```

[13]:

```
[('sensor_id', 'string'),
 ('machine', 'string'),
 ('timestamp', 'timestamp'),
 ('temperature_celsius', 'double'),
 ('vibration_level', 'double'),
 ('power_kw', 'double'),
 ('defect_count', 'int'),
 ('status_flag', 'string'),
 ('noise_val', 'double'),
```

```
 ('extra_param', 'double')]
```

Checking Missing Values

[14]:
```python
# Count missing (NaN/null) values in each column
missing_values = silver_sensor_df.select(
    [count(when(col(c).isNull(), c)).alias(c) for c in silver_sensor_df.columns]
)
missing_values.show()
```

```
+---------+-------+---------+-------------------+---------------+--------+------
------+-----------+---------+-----------+
|sensor_id|machine|timestamp|temperature_celsius|vibration_level|power_kw|defect
_count|status_flag|noise_val|extra_param|
+---------+-------+---------+-------------------+---------------+--------+------
------+-----------+---------+-----------+
|      108|    116|      336|                256|            292|     312|
336|         84|      252|        324|
+---------+-------+---------+-------------------+---------------+--------+------
------+-----------+---------+-----------+
```

[15]:
```python
# Dropping NaN/null values - removing rows which contain them
silver_sensor_df = silver_sensor_df.dropna()
missing_values = silver_sensor_df.select(
    [count(when(col(c).isNull(), c)).alias(c) for c in silver_sensor_df.columns]
)

missing_values.show()
```

```
+---------+-------+---------+-------------------+---------------+--------+------
------+-----------+---------+-----------+
|sensor_id|machine|timestamp|temperature_celsius|vibration_level|power_kw|defect
_count|status_flag|noise_val|extra_param|
+---------+-------+---------+-------------------+---------------+--------+------
------+-----------+---------+-----------+
|        0|      0|        0|                  0|              0|       0|
0|          0|        0|          0|
+---------+-------+---------+-------------------+---------------+--------+------
------+-----------+---------+-----------+
```

[16]:
```python
unique_machines = silver_sensor_df.select("machine").distinct()
unique_machines.show()
```

```
+---------------+
|        machine|
+---------------+
```

```
|   Conveyor-400|
|LaserCutter-900|
|    Grinder-800|
|        unknown|
|   CNC-Mill-200|
|   RobotArm-600|
|      Lathe-300|
| DrillPress-100|
|      Press-700|
| 3DPrinter-1000|
|    Turbine-500|
|            N/A|
+---------------+
```

[17]:
```
silver_sensor_df = silver_sensor_df.filter((silver_sensor_df.machine != "N/A")␣
↪& (silver_sensor_df.machine != "unknown"))
```

[18]:
```
unique_machines = silver_sensor_df.select("machine").distinct()
unique_machines.show()
```

```
+---------------+
|        machine|
+---------------+
|   Conveyor-400|
|LaserCutter-900|
|    Grinder-800|
|   CNC-Mill-200|
|   RobotArm-600|
|      Lathe-300|
| DrillPress-100|
|      Press-700|
| 3DPrinter-1000|
|    Turbine-500|
+---------------+
```

Checking Duplicates

[19]:
```
print(f"Number of rows before removing duplicates: {silver_sensor_df.count()}")
silver_sensor_df = silver_sensor_df.dropDuplicates()
print(f"Number of rows after removing duplicates: {silver_sensor_df.count()}")
```

```
Number of rows before removing duplicates: 198140
Number of rows after removing duplicates: 49535
```

Checking Anomalies

[20]:
```
silver_sensor_df.describe().show()
```

```
+-------+--------------------+--------------+-------------------+---------------
---+------------------+------------------+-----------+------------------+------
------------+
|summary|           sensor_id|       machine|temperature_celsius|
vibration_level|          power_kw|      defect_count|status_flag|
noise_val|       extra_param|
+-------+--------------------+--------------+-------------------+---------------
---+------------------+------------------+-----------+------------------+------
------------+
|  count|               49535|         49535|              49535|
49535|             49535|             49535|      49535|             49535|
49535|
|   mean|                NULL|          NULL|
53.32816715453714|1.8233606540829692|21.787283940647985|0.1859695165034824|
NULL| 0.5002125971535277| 150.0378318360755|
| stddev|                NULL|          NULL|
3.9770878025493053|0.5092000216898719| 1.815363019171402|0.6327945751348232|
NULL|0.28787625796276245|29.162658232450056|
|    min|0e033d6a-4e2b-41e…|3DPrinter-1000|              36.41|
-0.59|             14.13|                 0|        N/A|               0.0|
100.0|
|    max|             unknown|   Turbine-500|              69.32|
4.18|             30.08|                 3|    unknown|               1.0|
200.0|
+-------+--------------------+--------------+-------------------+---------------
---+------------------+------------------+-----------+------------------+------
------------+
```

[21]:
```python
negative_vibration_count = silver_sensor_df.filter(col("vibration_level") < 0).
  ↪count()
print(f"Rows with negative vibration_level: {negative_vibration_count}")
```

```
Rows with negative vibration_level: 18
```

[22]:
```python
silver_sensor_df = silver_sensor_df.filter(col("vibration_level") >= 0) #␣
  ↪vibration levels for machines should not be negative in a physical sense ->␣
  ↪could be an error in data reading
```

Saving

[23]:
```python
silver_sensor_path = "data_lake/silver/sensor_data_silver/"
os.makedirs(silver_sensor_path, exist_ok=True)

silver_sensor_df.write.format("delta") \
    .mode("overwrite") \
    .save(silver_sensor_path)
```

```python
print("Cleaned sensor data successfully stored in the Silver Layer.")
```

```
Cleaned sensor data successfully stored in the Silver Layer.
```

1.12.2 Production Data

Renaming Columns

[24]:
```python
bronze_production_df.columns
```

[24]:
```
['log_id',
 'product_type',
 'units_produced',
 'defective_units',
 'time_stamp',
 'MachineName',
 'remarks',
 'operator_id',
 'batch_info']
```

[25]:
```python
silver_production_df = (
    bronze_production_df.withColumnRenamed("time_stamp", "timestamp")
                        .withColumnRenamed("MachineName", "machine")
)

silver_production_df.columns
```

[25]:
```
['log_id',
 'product_type',
 'units_produced',
 'defective_units',
 'timestamp',
 'machine',
 'remarks',
 'operator_id',
 'batch_info']
```

Changing Types

[26]:
```python
silver_production_df.dtypes
```

[26]:
```
[('log_id', 'string'),
 ('product_type', 'string'),
 ('units_produced', 'string'),
 ('defective_units', 'string'),
 ('timestamp', 'string'),
 ('machine', 'string'),
 ('remarks', 'string'),
 ('operator_id', 'string'),
```

```
 ('batch_info', 'string')]
```

[27]:
```
silver_production_df = silver_production_df \
    .withColumn("timestamp", col("timestamp").cast(TimestampType())) \
    .withColumn("units_produced", col("units_produced").cast(IntegerType())) \
    .withColumn("defective_units", col("defective_units").cast(IntegerType()))
silver_production_df.dtypes
```

[27]:
```
[('log_id', 'string'),
 ('product_type', 'string'),
 ('units_produced', 'int'),
 ('defective_units', 'int'),
 ('timestamp', 'timestamp'),
 ('machine', 'string'),
 ('remarks', 'string'),
 ('operator_id', 'string'),
 ('batch_info', 'string')]
```

Checking Missing Values

[28]:
```
# Checking for NaN/null values
silver_production_df.select(
    [count(when(col(c).isNull(), c)).alias(c) for c in silver_production_df.
 ↪columns]
).show()
```

```
+------+------------+--------------+---------------+---------+-------+-------+--
---------+----------+
|log_id|product_type|units_produced|defective_units|timestamp|machine|remarks|op
erator_id|batch_info|
+------+------------+--------------+---------------+---------+-------+-------+--
---------+----------+
|     8|           4|            20|             16|        8|     12|     16|
4|         8|
+------+------------+--------------+---------------+---------+-------+-------+--
---------+----------+
```

[29]:
```
# Removing NaN/null values
silver_production_df = silver_production_df.dropna()
silver_production_df.select(
    [count(when(col(c).isNull(), c)).alias(c) for c in silver_production_df.
 ↪columns]
).show()
```

```
+------+------------+--------------+---------------+---------+-------+-------+--
---------+----------+
|log_id|product_type|units_produced|defective_units|timestamp|machine|remarks|op
```

```
erator_id|batch_info|
+------+------------+--------------+--------------+---------+-------+-------+--
---------+----------+
|     0|           0|             0|             0|        0|      0|      0|
0|         0|
+------+------------+--------------+--------------+---------+-------+-------+--
---------+----------+
```

[30]:
```python
unique_machines = silver_production_df.select("machine").distinct()
unique_machines.show()
```

```
+---------------+
|        machine|
+---------------+
|   Conveyor-400|
|LaserCutter-900|
|    Grinder-800|
|        unknown|
|   CNC-Mill-200|
|   RobotArm-600|
|      Lathe-300|
| DrillPress-100|
|      Press-700|
| 3DPrinter-1000|
|    Turbine-500|
|            N/A|
+---------------+
```

[31]:
```python
silver_production_df = silver_production_df.filter((silver_production_df.
 ↪machine != "N/A") & (silver_production_df.machine != "unknown"))
```

[32]:
```python
unique_machines = silver_production_df.select("machine").distinct()
unique_machines.show()
```

```
+---------------+
|        machine|
+---------------+
|   Conveyor-400|
|LaserCutter-900|
|    Grinder-800|
|   CNC-Mill-200|
|   RobotArm-600|
|      Lathe-300|
| DrillPress-100|
|      Press-700|
| 3DPrinter-1000|
```

```
|    Turbine-500|
+---------------+
```

Checking Duplicates

[33]:
```python
print(f"Number of rows before removing duplicates: {silver_production_df.
    ↪count()}")
silver_production_df = silver_production_df.dropDuplicates()
print(f"Number of rows after removing duplicates: {silver_production_df.
    ↪count()}")
```

```
Number of rows before removing duplicates: 11900
Number of rows after removing duplicates: 2975
```

Checking Anomalies

[34]:
```python
silver_production_df.describe().show()
```

```
+-------+--------------------+---------------+------------------+---------------
---+--------------+--------------------+-----------+--------------------+
|summary|              log_id|   product_type|    units_produced|
defective_units|       machine|             remarks|operator_id|
batch_info|
+-------+--------------------+---------------+------------------+---------------
---+--------------+--------------------+-----------+--------------------+
|  count|                2975|           2975|              2975|
2975|          2975|                2975|       2975|                2975|
|   mean|                NULL|
NULL|125.15327731092437|22.805042016806723|          NULL|                NULL|
NULL|                NULL|
| stddev|                NULL|           NULL| 35.66397813052963|
7.910582247185824|          NULL|                NULL|       NULL|
NULL|
|    min|00130a22-b979-4b9…|3D Printed Part|                42|
-1|3DPrinter-1000|Adjusted machine …|       OP10|Batch-3DPrinter-1…|
|    max|ffee8087-c73c-43d…|    Turbine Hub|               237|
50|   Turbine-500|             unknown|       OP20|             unknown|
+-------+--------------------+---------------+------------------+---------------
---+--------------+--------------------+-----------+--------------------+
```

[35]:
```python
negative_defective_unit_count = silver_production_df.
    ↪filter(col("defective_units") < 0).count()
print(f"Rows with negative defective units: {negative_defective_unit_count}")
```

```
Rows with negative defective units: 15
```

[36]:
```
silver_production_df = silver_production_df.filter(col("defective_units") >= 0)␣
↪# defective units cannot be negative -> error
```

Saving

[37]:
```
silver_production_path = "data_lake/silver/production_data_silver/"
os.makedirs(silver_production_path, exist_ok=True)

silver_production_df.write.format("delta") \
    .mode("overwrite") \
    .save(silver_production_path)

print("Cleaned production data successfully saved to the Silver Layer.")
```

```
Cleaned production data successfully saved to the Silver Layer.
```

1.12.3 Maintenance Data

Renaming Columns

[38]:
```
bronze_maintenance_df.columns
```

[38]:
```
['maintenance_id',
 'machine',
 'maintenance_date',
 'maintenance_type',
 'duration_minutes',
 'cost',
 'operator',
 'notes']
```

[39]:
```
silver_maintenance_df = bronze_maintenance_df \
    .withColumnRenamed("maintenance_date", "timestamp") \
    .withColumnRenamed("operator", "operator_id")
silver_maintenance_df.columns
```

[39]:
```
['maintenance_id',
 'machine',
 'timestamp',
 'maintenance_type',
 'duration_minutes',
 'cost',
 'operator_id',
 'notes']
```

Changing Types

[40]:
```
silver_maintenance_df.dtypes
```

[40]:
```
[('maintenance_id', 'string'),
 ('machine', 'string'),
 ('timestamp', 'timestamp'),
 ('maintenance_type', 'string'),
 ('duration_minutes', 'int'),
 ('cost', 'double'),
 ('operator_id', 'string'),
 ('notes', 'string')]
```

[41]:
```
silver_maintenance_df = silver_maintenance_df \
    .withColumn("timestamp", col("timestamp").cast(TimestampType())) \
    .withColumn("duration_minutes", col("duration_minutes").
 ↪cast(IntegerType())) \
    .withColumn("cost", col("cost").cast(DoubleType()))
silver_maintenance_df.dtypes
```

[41]:
```
[('maintenance_id', 'string'),
 ('machine', 'string'),
 ('timestamp', 'timestamp'),
 ('maintenance_type', 'string'),
 ('duration_minutes', 'int'),
 ('cost', 'double'),
 ('operator_id', 'string'),
 ('notes', 'string')]
```

Checking Missing Values

[42]:
```
silver_maintenance_df.select(
    [count(when(col(c).isNull(), c)).alias(c) for c in silver_maintenance_df.
 ↪columns]
).show()
```

```
+--------------+-------+---------+----------------+----------------+----+-------
----+-----+
|maintenance_id|machine|timestamp|maintenance_type|duration_minutes|cost|operato
r_id|notes|
+--------------+-------+---------+----------------+----------------+----+-------
----+-----+
|             0|      0|        0|               0|               0|   0|
0|    0|
+--------------+-------+---------+----------------+----------------+----+-------
----+-----+
```

[43]:
```
unique_machines = silver_maintenance_df.select("machine").distinct()
unique_machines.show()
```

```
+----------------+
```

```
|         machine|
+----------------+
|    Conveyor-400|
|LaserCutter-900|
|     Grinder-800|
|    CNC-Mill-200|
|    RobotArm-600|
|       Lathe-300|
|  DrillPress-100|
|       Press-700|
|  3DPrinter-1000|
|     Turbine-500|
+----------------+
```

Checking Duplicates

[44]:
```
print(f"Number of rows before removing duplicates: {silver_maintenance_df.
↪count()}")
silver_maintenance_df = silver_maintenance_df.dropDuplicates()
print(f"Number of rows after removing duplicates: {silver_maintenance_df.
↪count()}")
```

```
Number of rows before removing duplicates: 984
Number of rows after removing duplicates: 246
```

Checking Anomalies

[45]:
```
silver_maintenance_df.describe().show()
```

```
+-------+--------------------+--------------+----------------+-----------------
+------------------+-----------+--------------------+
|summary|      maintenance_id|       machine|maintenance_type| duration_minutes|
cost|operator_id|               notes|
+-------+--------------------+--------------+----------------+-----------------
+------------------+-----------+--------------------+
|  count|                 246|           246|             246|              246|
246|        246|                 246|
|   mean|                NULL|          NULL|            NULL|104.3170731707317|
570.1231707317074|       NULL|                NULL|
| stddev|                NULL|          NULL|
NULL|56.06116442151969|403.81712937623195|       NULL|                NULL|
|    min|002999f5-c036-476…|3DPrinter-1000|       Emergency|                31|
164.19|      MT100|Addressed minor l…|
|    max|fdf1f306-6ace-4d8…|   Turbine-500|     Unscheduled|               235|
1497.8|        N/A|Unexpected shutdo…|
+-------+--------------------+--------------+----------------+-----------------
+------------------+-----------+--------------------+
```

Saving

[46]:
```
silver_maintenance_path = "data_lake/silver/maintenance_data_silver/"
os.makedirs(silver_maintenance_path, exist_ok=True)

silver_maintenance_df.write.format("delta") \
    .mode("overwrite") \
    .save(silver_maintenance_path)

print("Cleaned maintenance data successfully saved to the Silver Layer.")
```

```
Cleaned maintenance data successfully saved to the Silver Layer.
```

1.12.4 Operator Dimension Table

[47]:
```
dim_operator_path = "dim_operator.csv"

dim_operator_df = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load(dim_operator_path)
dim_operator_df.show(5)
dim_operator_df.printSchema()
```

```
+-----------+---------------+-------------+
|operator_id|  operator_name|operator_type|
+-----------+---------------+-------------+
|      MT100|   Allison Hill|  Maintenance|
|       OP12|    Noah Rhodes|   Production|
|      MT101|Angie Henderson|  Maintenance|
|      MT102|  Daniel Wagner|  Maintenance|
|       OP18|Cristian Santos|   Production|
+-----------+---------------+-------------+
only showing top 5 rows

root
 |-- operator_id: string (nullable = true)
 |-- operator_name: string (nullable = true)
 |-- operator_type: string (nullable = true)
```

[48]:
```
# Save to Silver Layer as a Delta table
silver_operator_path = "data_lake/silver/dim_operator_silver/"
os.makedirs(silver_operator_path, exist_ok=True)

dim_operator_df.write.format("delta") \
    .mode("overwrite") \
    .save(silver_operator_path)
```

```
print("Operator Dimension Table successfully saved to the Silver Layer.")
```

```
Operator Dimension Table successfully saved to the Silver Layer.
```

1.13 Gold Layer – Business Insights

[49]:
```
# Load Silver Layer tables
sensor_df = spark.read.format("delta").load("data_lake/silver/
↪sensor_data_silver")
production_df = spark.read.format("delta").load("data_lake/silver/
↪production_data_silver")
maintenance_df = spark.read.format("delta").load("data_lake/silver/
↪maintenance_data_silver")
operator_df = spark.read.format("delta").load("data_lake/silver/
↪dim_operator_silver")
```

1.13.1 Daily Sensor Metrics

[50]:
```
# Convert timestamp to date for daily aggregation
sensor_metrics = sensor_df.withColumn("date", to_date(col("timestamp"))) \
    .groupBy("machine", "date") \
    .agg(
        avg("temperature_celsius").alias("avg_temp_celsius"),
        avg("vibration_level").alias("avg_vibration_level"),
        avg("power_kW").alias("avg_power_kW"),
        sum("defect_count").alias("total_defective_sensor_readings"),
        count(when(col("status_flag") == "Stopped", True)).
↪alias("downtime_count")
    ) \
    .orderBy(col("downtime_count").desc())
sensor_metrics.show()
```

```
+---------------+----------+------------------+-------------------+-------------
-----+-------------------------------+--------------+
|        machine|      date|  avg_temp_celsius|avg_vibration_level|
avg_power_kW|total_defective_sensor_readings|downtime_count|
+---------------+----------+------------------+-------------------+-------------
-----+-------------------------------+--------------+
|   CNC-Mill-200|2024-11-28| 59.22699999999999| 1.8848000000000003|
22.1024|                              2|            50|
|    Grinder-800|2024-10-22|           51.4114|
1.8628000000000002|21.371000000000002|                             18|
50|
|      Press-700|2024-10-08| 52.92700000000001|             1.7162|
21.3132|                             15|            50|
|    Turbine-500|2024-11-27| 55.72120000000001|
```

```
1.9028|21.992800000000003|                              7|             50|
|   Conveyor-400|2024-10-24|            54.218|             1.6632|
21.2046|                              8|             50|
|   CNC-Mill-200|2024-11-16|56.274800000000006| 1.8396000000000001|
21.5108|                              2|             50|
|    Grinder-800|2024-10-02|48.940999999999995|             1.7226|
21.0508|                              5|             50|
| 3DPrinter-1000|2024-11-06|           49.4552|
1.7806|21.417199999999998|                             14|             50|
|   RobotArm-600|2024-12-29|54.657599999999995| 1.8774000000000002|
21.9312|                              7|             50|
|LaserCutter-900|2024-10-23|           50.4344|
1.8115999999999999|20.758999999999997|                             10|
50|
|   CNC-Mill-200|2024-10-28|54.379799999999996|             1.7482|
21.4448|                              6|             50|
|      Press-700|2024-11-04|           55.8904|
2.0220000000000002|22.529600000000002|                              9|
50|
|    Grinder-800|2024-12-11|           53.3012| 1.8163999999999998|
21.9822|                              3|             50|
|LaserCutter-900|2024-10-04| 49.40079999999999| 1.9265999999999999|
20.8426|                             10|             50|
|   RobotArm-600|2024-12-04| 56.78799999999999| 1.8628000000000002|
22.4544|                             20|             50|
|LaserCutter-900|2024-12-03|           50.5732| 1.7462000000000004|
21.8658|                             16|             50|
|    Grinder-800|2024-10-08| 48.54600000000001|             1.6454|
21.1152|                              6|             50|
|   CNC-Mill-200|2024-11-21|           58.8006|
1.9422|21.852800000000002|                             16|             50|
|    Grinder-800|2024-11-06|            52.839|
1.9054|21.755399999999998|                             13|             50|
|    Grinder-800|2024-11-21| 54.21940000000001| 1.9432000000000003|
22.174|                              7|             50|
+---------------+----------+------------------+-------------------+-------------
-----+------------------------------+--------------+
only showing top 20 rows
```

1.13.2 Daily Production Metrics

[51]:
```
# Convert timestamp to date for daily aggregation
production_metrics = production_df.withColumn("date",␣
↪to_date(col("timestamp"))) \
    .groupBy("machine", "date") \
    .agg(
```

```
        sum("units_produced").alias("total_units_produced"),
        sum("defective_units").alias("total_defective_units"),
        (sum("defective_units") / sum("units_produced")).alias("defect_rate"),
        ((sum("units_produced") - sum("defective_units")) /␣
↪sum("units_produced")).alias("production_yield")
    ) \
    .orderBy(col("defect_rate").desc())
production_metrics.show()
```

```
+--------------+----------+--------------------+---------------------+----------
---------+------------------+
|       machine|      date|total_units_produced|total_defective_units|
defect_rate|  production_yield|
+--------------+----------+--------------------+---------------------+----------
---------+------------------+
|   Turbine-500|2024-11-27|                 114|
27|0.23684210526315788|0.7631578947368421|
|   Grinder-800|2024-11-13|                  81|                   19|
0.2345679012345679|0.7654320987654321|
|     Lathe-300|2024-10-15|                  65|
15|0.23076923076923078|0.7692307692307693|
|     Press-700|2024-11-27|                 265|
61|0.23018867924528302| 0.769811320754717|
|   Turbine-500|2024-12-02|                 340|
78|0.22941176470588234|0.7705882352941177|
|   Turbine-500|2024-11-30|                 315|
72|0.22857142857142856|0.7714285714285715|
|     Press-700|2024-12-19|                 321|
73|0.22741433021806853|0.7725856697819314|
|DrillPress-100|2024-12-18|                 285|
64|0.22456140350877193| 0.775438596491228|
|     Lathe-300|2024-11-17|                 225|                   50|
0.2222222222222222|0.7777777777777778|
|   Turbine-500|2024-12-04|                 315|                   70|
0.2222222222222222|0.7777777777777778|
|   Turbine-500|2025-01-03|                 440|
97|0.22045454545454546|0.7795454545454545|
|  Conveyor-400|2024-12-29|                 992|
218|0.21975806451612903| 0.780241935483871|
|   Turbine-500|2024-12-07|                 405|
89|0.21975308641975308|0.7802469135802469|
|     Lathe-300|2025-01-01|                 456|
100|0.21929824561403508|0.7807017543859649|
|DrillPress-100|2024-12-19|                 151|                   33|
0.2185430463576159|0.7814569536423841|
|     Lathe-300|2024-10-29|                 447|
97|0.21700223713646533|0.7829977628635347|
```

```
|  RobotArm-600|2024-12-02|                 272|
59|0.21691176470588236|0.7830882352941176|
|3DPrinter-1000|2024-12-02|                 120|
26|0.21666666666666667|0.7833333333333333|
|   Grinder-800|2024-12-11|                 568|
123|0.21654929577464788|0.7834507042253521|
|   Turbine-500|2024-12-28|                 611|                 132|
0.2160392798690671|0.7839607201309329|
+--------------+----------+--------------------+--------------------+----------
---------+------------------+
only showing top 20 rows
```

1.13.3 Daily Maintenance Metrics

[52]:

```python
maintenance_metrics = maintenance_df.withColumn("date",␣
↪to_date(col("timestamp"))) \
    .groupBy("machine", "date") \
    .agg(
        sum("duration_minutes").alias("total_maintenance_duration"),
        count(when(col("maintenance_type") == "Scheduled", True)).
↪alias("scheduled_maintenance"),
        count(when(col("maintenance_type") == "Unscheduled", True)).
↪alias("unscheduled_maintenance"),
        count(when(col("maintenance_type") == "Emergency", True)).
↪alias("emergency_maintenance"),
        sum("cost").alias("total_maintenance_cost")
    ) \
    .orderBy(col("total_maintenance_cost").desc())
maintenance_metrics.show()
```

```
+---------------+----------+--------------------------+---------------------+---
--------------------+---------------------+----------------------+
|        machine|      date|total_maintenance_duration|scheduled_maintenance|uns
cheduled_maintenance|emergency_maintenance|total_maintenance_cost|
+---------------+----------+--------------------------+---------------------+---
--------------------+---------------------+----------------------+
|   Conveyor-400|2024-11-17|                       209|                    0|
0|                    1|               1497.8|
|LaserCutter-900|2024-12-01|                       216|                    0|
0|                    1|              1486.73|
|   RobotArm-600|2025-01-04|                       176|                    0|
0|                    1|              1486.64|
|   RobotArm-600|2024-12-21|                       179|                    0|
0|                    1|              1472.55|
|   RobotArm-600|2024-10-23|                       214|                    0|
0|                    1|              1453.07|
```

```
| 3DPrinter-1000|2024-10-12|                        209|                     0|
0|                     1|                1443.28|
|    Turbine-500|2024-12-14|                        218|                     0|
0|                     1|                1414.64|
|   CNC-Mill-200|2024-12-27|                        167|                     0|
0|                     1|                1412.29|
|   RobotArm-600|2024-12-28|                        161|                     0|
0|                     1|                1411.01|
|   RobotArm-600|2024-12-12|                        138|                     0|
0|                     1|                1410.59|
|   Conveyor-400|2024-11-14|                        179|                     0|
0|                     1|                1390.82|
|   Conveyor-400|2024-11-05|                        150|                     0|
0|                     1|                 1386.5|
|    Turbine-500|2024-11-19|                        180|                     0|
0|                     1|                1380.45|
|LaserCutter-900|2024-11-18|                        212|                     0|
0|                     1|                1375.71|
|    Grinder-800|2024-10-27|                        174|                     0|
0|                     1|                 1339.1|
|   CNC-Mill-200|2024-10-29|                        182|                     0|
0|                     1|                1329.23|
|   Conveyor-400|2024-10-02|                        140|                     0|
0|                     1|                1312.39|
|    Grinder-800|2024-11-10|                        148|                     0|
0|                     1|                1309.05|
|      Press-700|2024-11-11|                        192|                     0|
0|                     1|                1294.81|
| DrillPress-100|2024-11-30|                        164|                     0|
0|                     1|                 1291.8|
+---------------+----------+---------------------------+----------------------+---
--------------------+----------------------+----------------------+
only showing top 20 rows
```

1.13.4 Joining Data

[53]:

```python
# Join sensor, production, and maintenance metrics on machine and date
gold_df = sensor_metrics \
    .join(production_metrics, ["machine", "date"], "left") \
    .join(maintenance_metrics, ["machine", "date"], "left")

gold_df.columns
```

[53]:

```
['machine',
 'date',
 'avg_temp_celsius',
```

```
'avg_vibration_level',
'avg_power_kW',
'total_defective_sensor_readings',
'downtime_count',
'total_units_produced',
'total_defective_units',
'defect_rate',
'production_yield',
'total_maintenance_duration',
'scheduled_maintenance',
'unscheduled_maintenance',
'emergency_maintenance',
'total_maintenance_cost']
```

1.13.5 Saving Data

[54]:
```python
save_path = "data_lake/gold/gold_machine_performance"
os.makedirs(save_path, exist_ok=True)

gold_df.write.format("delta").mode("overwrite").save(save_path)
```

[55]:
```python
gold_df = spark.read.format("delta").load(save_path)
gold_df.show()
```

```
+---------------+----------+------------------+-------------------+-------------
-----+-------------------------------+--------------+--------------------+------
---------------+-------------------+------------------+-------------------------
-+---------------------+-----------------------+---------------------+----------
------------+
|        machine|      date|   avg_temp_celsius|avg_vibration_level|      avg_pow
er_kW|total_defective_sensor_readings|downtime_count|total_units_produced|total_
defective_units|        defect_rate|   production_yield|total_maintenance_duratio
n|scheduled_maintenance|unscheduled_maintenance|emergency_maintenance|total_main
tenance_cost|
+---------------+----------+------------------+-------------------+-------------
-----+-------------------------------+--------------+--------------------+------
---------------+-------------------+------------------+-------------------------
-+---------------------+-----------------------+---------------------+----------
------------+
|   CNC-Mill-200|2024-11-07|57.463469387755104| 1.8671428571428568|
21.21081632653061|                              3|            49|
175|                  28|                0.16|              0.84|
NULL|                    NULL|                       NULL|                 NULL|
NULL|
|   RobotArm-600|2024-11-21| 56.42632653061224|
1.8957142857142855|22.294693877551023|                              7|
13|                 683|
```

```
128|0.18740849194729137|0.8125915080527086|                      NULL|
NULL|                    NULL|                      NULL|                      NULL|
|      Press-700|2024-12-02| 54.92416666666666| 1.8674999999999997|
22.456875|                              10|             48|                  554|
104|0.18772563176895307|0.8122743682310469|                      NULL|
NULL|                    NULL|                      NULL|                      NULL|
|    Turbine-500|2024-11-27| 55.72120000000001|
1.9028|21.992800000000003|                              7|             50|
114|                    27|0.23684210526315788|0.7631578947368421|
NULL|                  NULL|                        NULL|                    NULL|
NULL|
|      Lathe-300|2024-10-26|52.358163265306125|
1.810816326530612|21.934489795918367|                          7|
49|                 110|
20|0.18181818181818182|0.8181818181818182|                      NULL|
NULL|                    NULL|                      NULL|                      NULL|
| 3DPrinter-1000|2025-01-01| 48.35918367346939|
1.8591836734693876|21.476938775510206|                         1|
15|                 139|
23|0.16546762589928057|0.8345323741007195|                      NULL|
NULL|                    NULL|                      NULL|                      NULL|
| 3DPrinter-1000|2025-01-05| 46.14459999999999| 1.7688000000000001|
21.1772|                              1|             21|                  267|
42|0.15730337078651685|0.8426966292134831|                      NULL|
NULL|                    NULL|                      NULL|                      NULL|
|   Conveyor-400|2024-10-24|            54.218|             1.6632|
21.2046|                              8|             50|                  395|
58| 0.1468354430379747|0.8531645569620253|                      NULL|
NULL|                    NULL|                      NULL|                      NULL|
|   CNC-Mill-200|2024-11-16|56.274800000000006| 1.8396000000000001|
21.5108|                              2|             50|                  498|
103|0.20682730923694778|0.7931726907630522|                      NULL|
NULL|                    NULL|                      NULL|                      NULL|
|LaserCutter-900|2024-11-15|           52.0492|
1.9335999999999998|22.371399999999998|                         6|
7|                 130|
25|0.19230769230769232|0.8076923076923077|                        66|
1|                      0|                        0|                339.83|
|    Grinder-800|2024-10-02|48.940999999999995|             1.7226|
21.0508|                              5|             50|                  446|
70|0.15695067264573992|0.8430493273542601|                      NULL|
NULL|                    NULL|                      NULL|                      NULL|
| 3DPrinter-1000|2024-11-06|           49.4552|
1.7806|21.417199999999998|                             14|             50|
336|                    62|0.18452380952380953|0.8154761904761905|
NULL|                  NULL|                        NULL|                    NULL|
NULL|
|      Lathe-300|2024-12-26| 55.79104166666667|
```

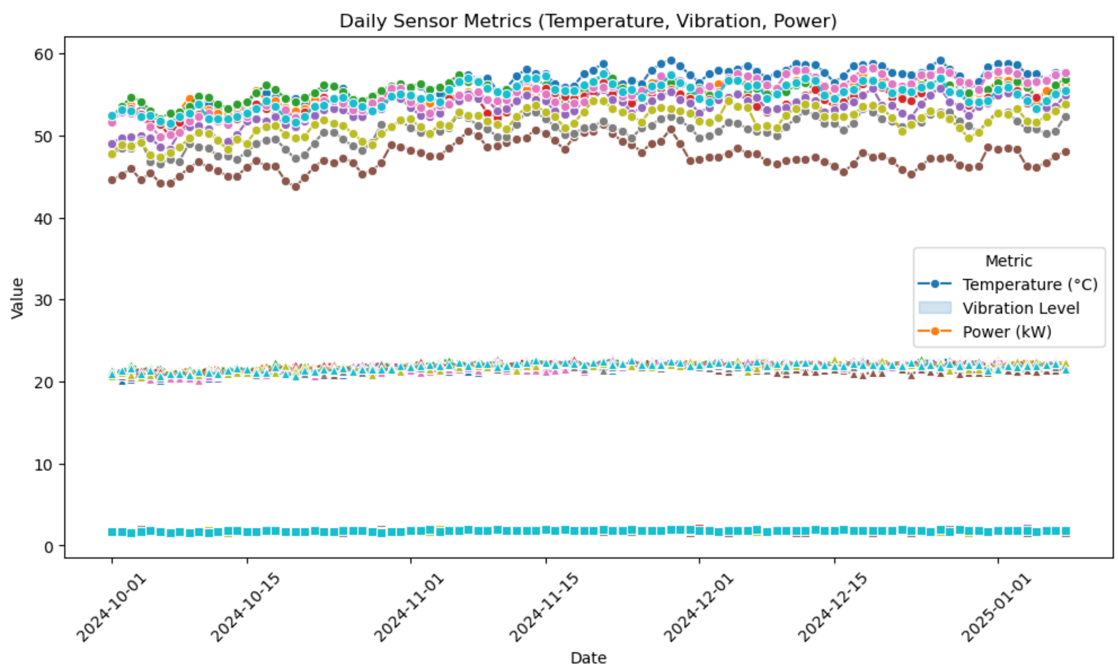
```
2.0727083333333334|22.481458333333336|                              11|
48|                493|
98|0.19878296146044624|0.8012170385395537|                      NULL|
NULL|                  NULL|                  NULL|                   NULL|
|LaserCutter-900|2024-12-17|           51.4824|
1.7444|21.839799999999997|                              13|            49|
220|                  44|                0.2|                0.8|
NULL|                NULL|                    NULL|                 NULL|
NULL|
| 3DPrinter-1000|2024-12-20| 47.50163265306122|
1.7448979591836735|21.169183673469387|                              8|
21|                188|
25|0.13297872340425532|0.8670212765957447|                      NULL|
NULL|                  NULL|                  NULL|                   NULL|
|   RobotArm-600|2024-12-29|54.657599999999995| 1.8774000000000002|
21.9312|                              7|              50|                 707|
145| 0.2050919377652051|0.7949080622347949|                      NULL|
NULL|                  NULL|                  NULL|                   NULL|
|LaserCutter-900|2024-10-23|           50.4344|
1.8115999999999999|20.758999999999997|                              10|
50|                204|
30|0.14705882352941177|0.8529411764705882|                      NULL|
NULL|                  NULL|                  NULL|                   NULL|
|   Conveyor-400|2024-12-10| 57.50755102040816| 2.0106122448979593|
22.5234693877551|                         13|            49|
169|                  31| 0.1834319526627219|0.8165680473372781|
NULL|                NULL|                    NULL|                 NULL|
NULL|
|   CNC-Mill-200|2024-10-28|54.379799999999996|            1.7482|
21.4448|                              6|              50|                 356|
55| 0.1544943820224719|0.8455056179775281|                      NULL|
NULL|                  NULL|                  NULL|                   NULL|
|LaserCutter-900|2024-11-11| 49.85166666666666|
1.76625|21.610833333333332|                              10|            48|
837|                 148| 0.1768219832735962|0.8231780167264038|
NULL|                NULL|                    NULL|                 NULL|
NULL|
+---------------+----------+------------------+-------------------+------------
-----+-------------------------------+-------------+-------------------+------
---------------+-------------------+------------------+------------------------
-+---------------------+------------------------+-------------------+----------
------------+
only showing top 20 rows
```

1.13.6 Visualizations

[56]:
```python
gold_pandas_df = gold_df.toPandas()
```

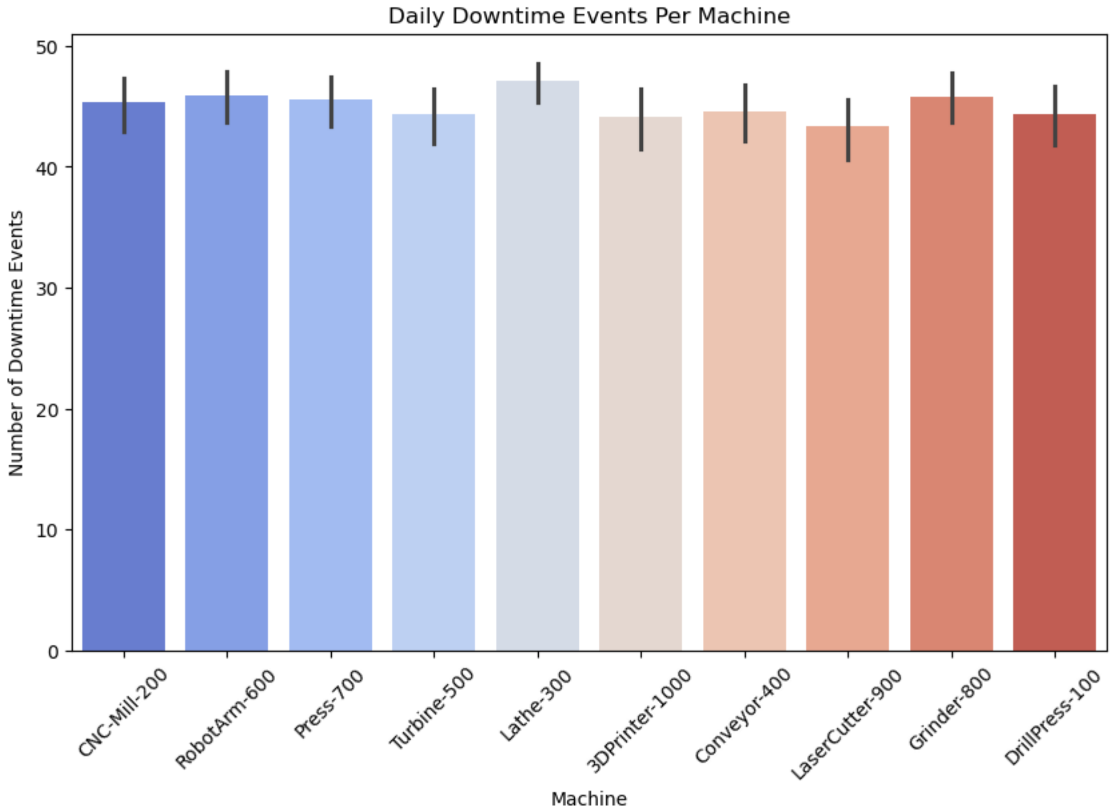
[57]:
```python
plt.figure(figsize=(12, 6))
sns.lineplot(data=gold_pandas_df, x="date", y="avg_temp_celsius", hue="machine", marker="o")
sns.lineplot(data=gold_pandas_df, x="date", y="avg_vibration_level", hue="machine", marker="s", linestyle="dashed")
sns.lineplot(data=gold_pandas_df, x="date", y="avg_power_kW", hue="machine", marker="^", linestyle="dotted")

plt.title("Daily Sensor Metrics (Temperature, Vibration, Power)")
plt.xlabel("Date")
plt.ylabel("Value")
plt.xticks(rotation=45)
plt.legend(title="Metric", labels=["Temperature (°C)", "Vibration Level", "Power (kW)"])
plt.show()
```



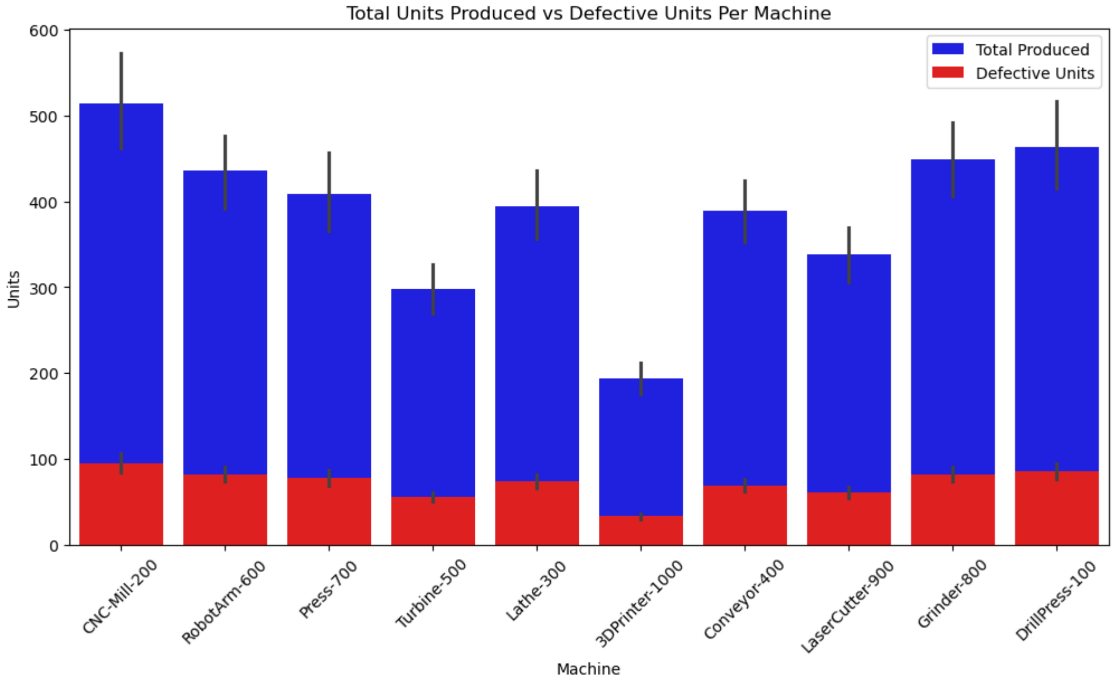
[58]:
```python
plt.figure(figsize=(10, 6))
sns.barplot(data=gold_pandas_df, x="machine", hue="machine", y="downtime_count", palette="coolwarm", legend=False)
plt.title("Daily Downtime Events Per Machine")
plt.xlabel("Machine")
```

```
plt.ylabel("Number of Downtime Events")
plt.xticks(rotation=45)
plt.show()
```

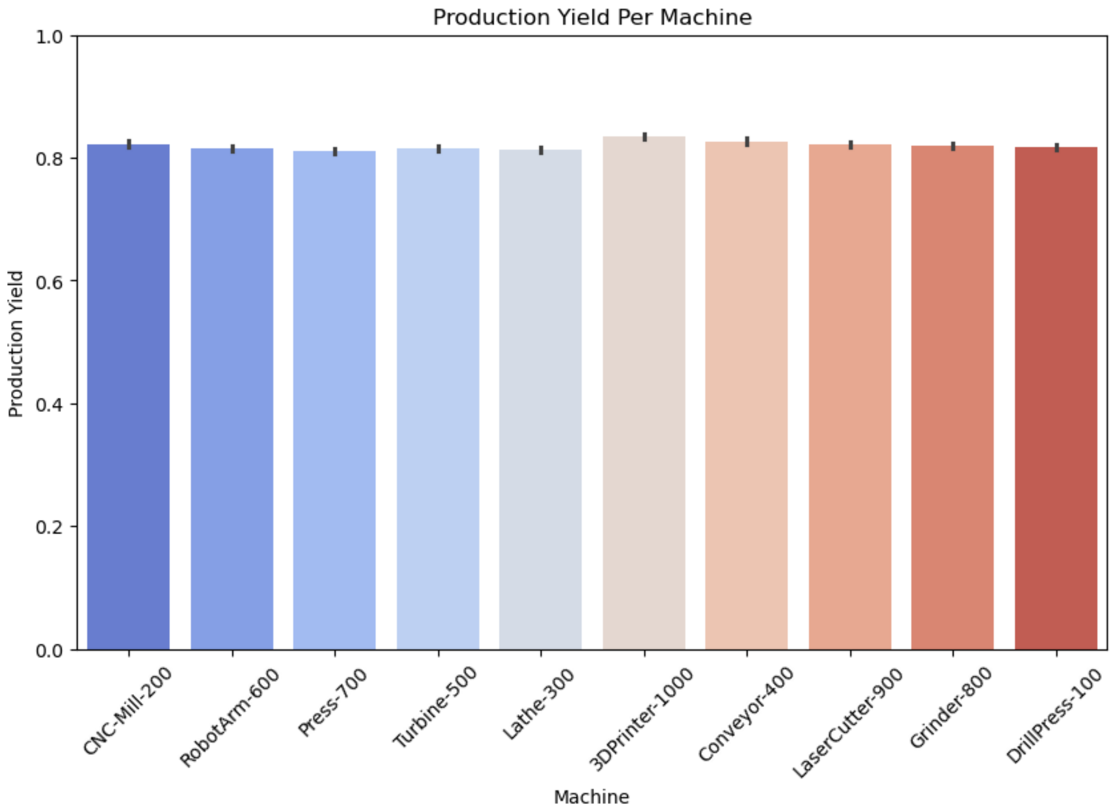


[59]:
```
plt.figure(figsize=(12, 6))
sns.barplot(data=gold_pandas_df, x="machine", y="total_units_produced",␣
 ↪color="blue", label="Total Produced")
sns.barplot(data=gold_pandas_df, x="machine", y="total_defective_units",␣
 ↪color="red", label="Defective Units")

plt.title("Total Units Produced vs Defective Units Per Machine")
plt.xlabel("Machine")
plt.ylabel("Units")
plt.xticks(rotation=45)
plt.legend()
plt.show()
```

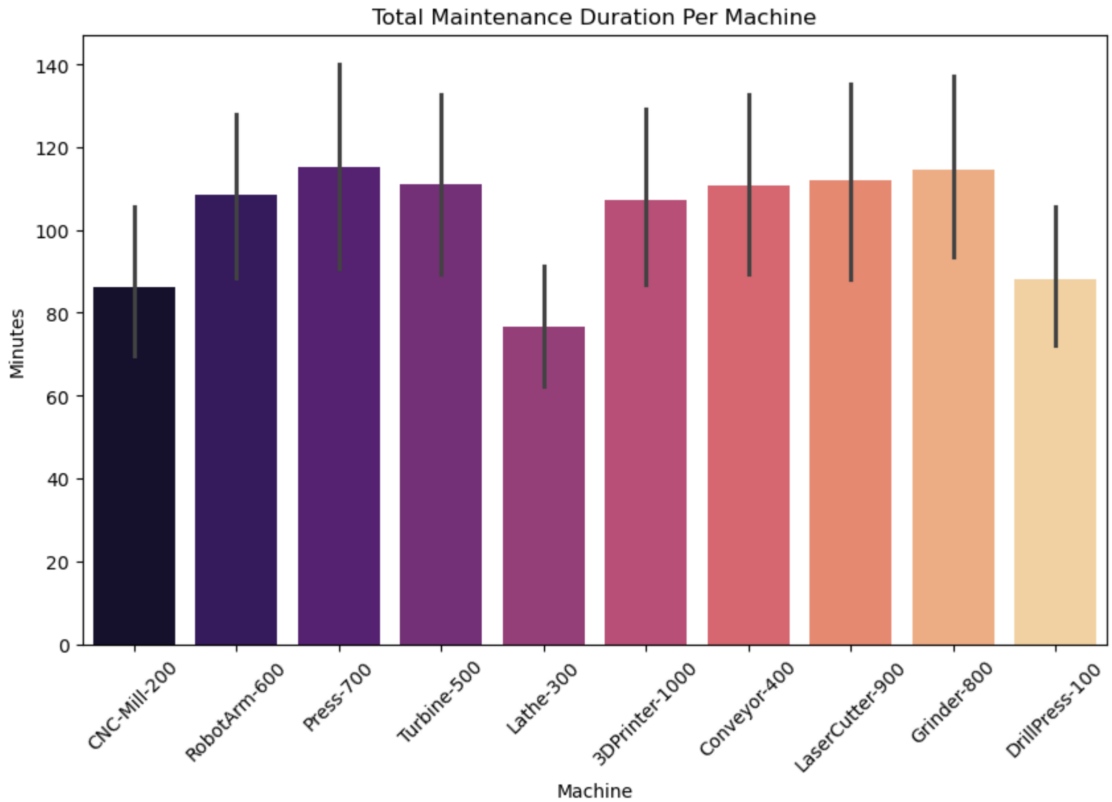


[60]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=gold_pandas_df, x="machine", hue="machine",␣
↪y="production_yield", palette="coolwarm")
plt.title("Production Yield Per Machine")
plt.xlabel("Machine")
plt.ylabel("Production Yield")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.show()
```

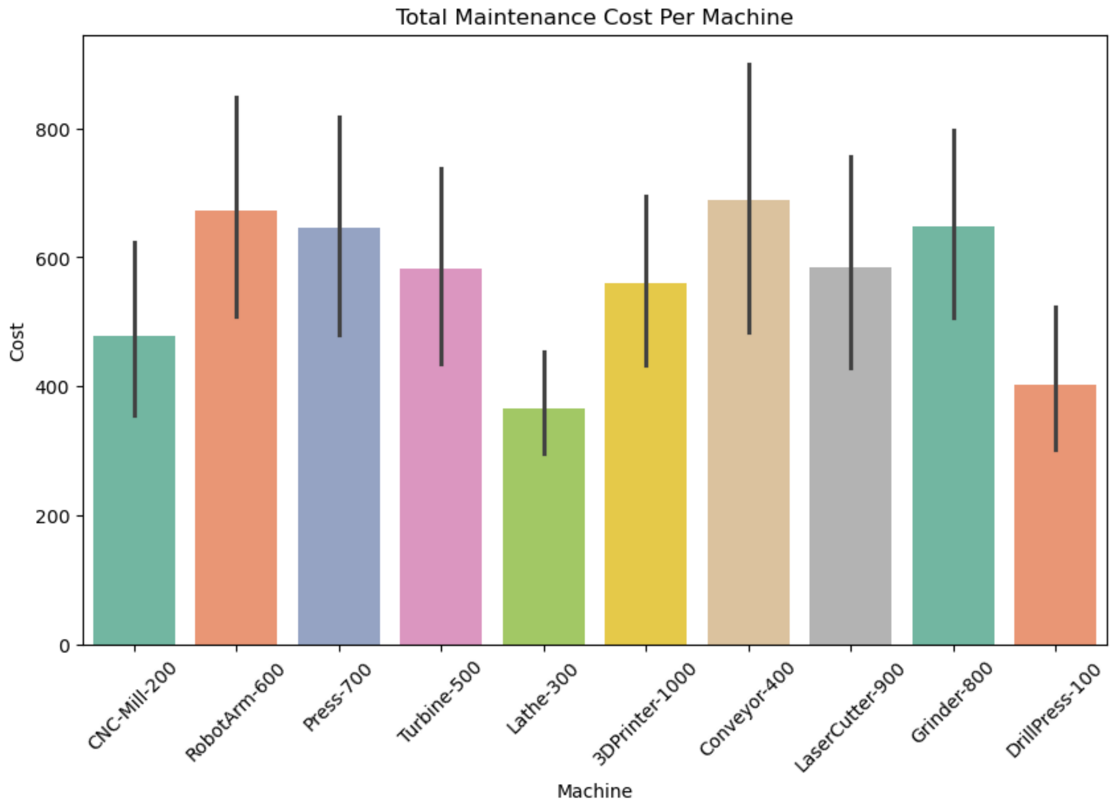


[61]:

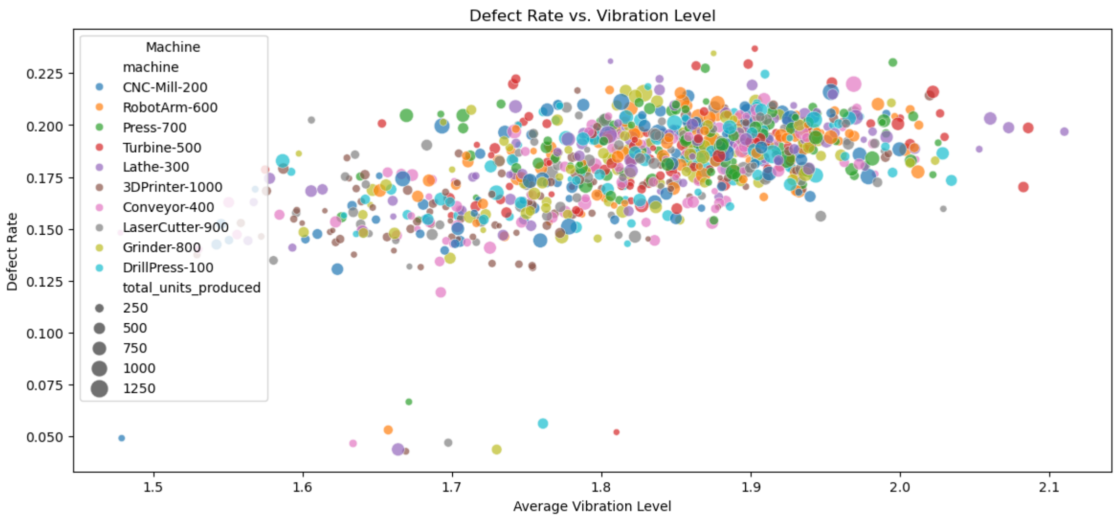
```
plt.figure(figsize=(10, 6))
sns.barplot(data=gold_pandas_df, x="machine", hue="machine",␣
 ↪y="total_maintenance_duration", palette="magma")
plt.title("Total Maintenance Duration Per Machine")
plt.xlabel("Machine")
plt.ylabel("Minutes")
plt.xticks(rotation=45)
plt.show()
```



[62]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=gold_pandas_df, x="machine", hue="machine",␣
 ↪y="total_maintenance_cost", palette="Set2")
plt.title("Total Maintenance Cost Per Machine")
plt.xlabel("Machine")
plt.ylabel("Cost")
plt.xticks(rotation=45)
plt.show()
```

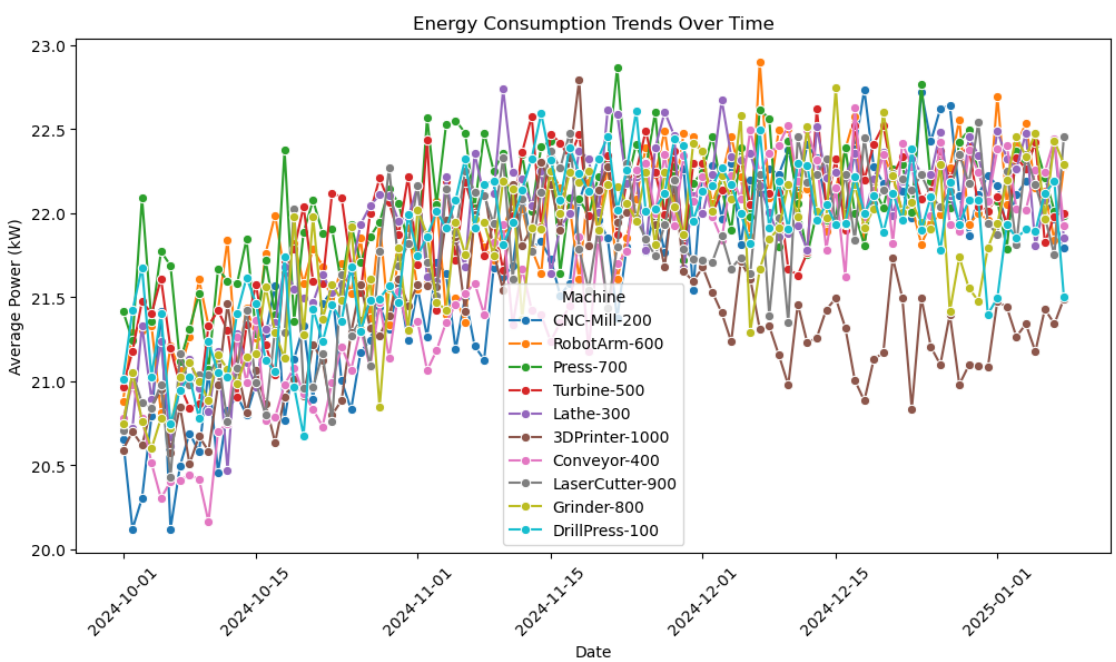


[64]:
```
plt.figure(figsize=(14, 6))
sns.scatterplot(data=gold_pandas_df, x="avg_vibration_level", y="defect_rate",␣
↪hue="machine", size="total_units_produced", sizes=(20, 200), alpha=0.7)
plt.title("Defect Rate vs. Vibration Level")
plt.xlabel("Average Vibration Level")
plt.ylabel("Defect Rate")
plt.legend(title="Machine")
plt.show()
```



[65]:

```
plt.figure(figsize=(12, 6))
sns.lineplot(data=gold_pandas_df, x="date", y="avg_power_kW", hue="machine",␣
 ↪marker="o")
plt.title("Energy Consumption Trends Over Time")
plt.xlabel("Date")
plt.ylabel("Average Power (kW)")
plt.xticks(rotation=45)
plt.legend(title="Machine")
plt.show()
```



1.14 Discussion Questions (6p)

1. **Have we now built a data lakehouse? Why or why not?**

Yes, by looking at the characteristics of data lakehouses (unified-storage, ACID, scalability, integration with ML etc.) we can say that we have built a basic data lakehouse.

One aspect of lake houses is that they combine the flexibility of data lakes and reliability of data warehouses. We structured data into Bronze, Silver and Gold layers using Delta Lakes; thus, we ensured data consistency and schema enforcement. We then performed data cleaning and standardization in the Silver layerand aggregated bussiness insights in the Gold layer. The use of Delta tables with ACID transactions ensured data integrity, which is one of the advantages Data lakehouses have over data lakes. We also performed automatic schema inference and schema evolution, which allows flexibility in handling changing data structures, which is one of the key aspects of delta lakehouses. Moreover, the dataframe we have in Gold layeris ready for Machine Learnning pipelines.

Data lakehouses can handle both structured and semi- or unstructured data. Since we only had structured data, our implementation could not fully utilize the flexibility of a data lakehouse. Additionally, we did not implement advanced features like indexing or caching, which are common in lakehouse environments.

In overall, our implementation meets the core concepts of a data lakehouse, but it could be even extended to do advanced operations that data warehouses and data lakes are incapable of doing.

2. **How does the medallion architecture enhance data quality and governance in this pipeline?**

Bronze Layer: This layer stores raw data exactly as extracted from different sources. By keeping a copy of raw data, it preserves data lineage and enables us to recover from errors or reprocess data if needed. Since this layer acts as a historical archive, it ensures data traceability and compliance by maintaining the record of all incoming data. It also provides schema enforcement, preventing invalid records from corrupting the data lake.

Silver Layer: This layer is responsible for cleaning, standardizing, and validating the raw data. We performed the following transformations to improve data quality: - Renaming columns for consistency. - Removing duplicate records to eliminate redundancy. - Handling missing or corrupt values to ensure meaningful analysis. - Standardizing data types for compatibility across datasets. - Filtering out invalid sensor readings (e.g., negative values in vibration levels).

All these transformations make our data more structured and high quality. In terms of governance, this layer ensured schema evolution and validation, preventing incorrect data from propogating downstream.

Gold Layer: This layer aggregates and enriches the cleaned data to provide business insights. We created meaningful machine performance metrics such as: - Daily sensor metrics (e.g., average temperature, vibration levels, power usage). - Daily Production metrics (e.g., units produced vs. defective units, defect rates). - Daily maintenance metrics (e.g., analysis (total maintenance duration, cost per machine). - Adcanced insight such as correlation between vibration levels and defects, energy consumption over time etc.

So this layer provides well-structured, curated datasets for analytics, reporting, and machine learning. It allows for fine-grained access control, ensuring that different user groups can access only relevant data while maintaining security and compliance.

3. **What challenges might arise when scaling this pipeline from batch-based to real-time streaming data?**

Scaling this pipeline from batch processing to real-time streaming might have some challenges. Data ingestion and latency become critical, requiring tools like Apache Kafka to handle continuous sensor, production, and maintenance data. Ensuring data quality is harder in a streaming setup, as records may arrive out of order or incomplete, requiring real-time validation and deduplication. Schema evolution also becomes a challenge since unexpected format changes can disrupt processing without proper handling. Additionally, fault tolerance and scalability must be addressed to handle system failures and spikes in data volume efficiently. Lastly, operational costs increase since streaming requires continuous resource usage.