Data Analyst Assignment: Analyzing Manufacturing Data to Optimize Production Efficiency

Christian Moser

October 15, 2024

1 Background

You've been hired as a Data Analyst on the **manufacturing team** at SpaceX. Your first task is to analyze data from one of the rocket production lines. The goal is to identify bottlenecks and inefficiencies in the process that are affecting the overall production time and suggest ways to improve throughput.

2 Data Provided

You have access to a dataset that includes the following columns for the past month:

- `Step_ID`: Identifier for the production step (e.g., Assembly, Welding, Inspection, Testing).
- `Duration (hours)`: Time taken for that step to complete.
- `Start_Time`: When the step started.
- `End_Time`: When the step ended.
- `Operator_ID`: The ID of the operator responsible for that step.
- `Machine_ID`: The ID of the machine used in that step (if applicable).
- `Fault_Flag`: A binary flag (1 or 0) indicating if there was a fault during the process that caused a delay.

3 Steps to Complete the Assignment

This guide will take you through each step in Python using commonly used libraries like Pandas for data manipulation and Matplotlib or Seaborn for visualizations. Each section will include Python code examples and explanations.

3.1 1. Setting Up Jupyter Notebook in VSCode

Before starting, make sure you have Visual Studio Code (VSCode) installed on your machine. VSCode is a lightweight, powerful editor that can be used for running Jupyter Notebooks.

Follow these steps to set up Jupyter in VSCode:

- Open VSCode.
- Install the necessary extensions:
  1. Open the Extensions view by clicking the Extensions icon on the left sidebar or by pressing `Ctrl+Shift+X`.
  2. Search for and install the following extensions:
     - **Python** (provided by Microsoft): This extension enables Python support and integrates with Jupyter.

- **Jupyter** (provided by Microsoft): This extension allows you to run Jupyter Notebooks inside VSCode.

- Once the extensions are installed, create a new Jupyter Notebook:
  1. Open the Command Palette by pressing `Ctrl+Shift+P`.
  2. Type `Jupyter: Create New Blank Notebook` and press `Enter`.
  3. This will open a new notebook file with a `.ipynb` extension.

- Select your Python interpreter:
  1. When the notebook opens, VSCode will prompt you to select a Python interpreter. Choose the Python environment that already has the necessary libraries installed (such as `pandas`, `numpy`, `matplotlib`, and `seaborn`).
  2. If no prompt appears, you can manually select the Python interpreter by clicking on the Python version shown at the bottom-left of the VSCode window.

- Now, you can start writing and running your Python code in the Jupyter Notebook cells.
  - To create a new cell, press `Shift+Enter` after writing your code, and it will automatically run the current cell and move to the next.

At this point, your Jupyter Notebook environment in VSCode should be set up and ready to use for data analysis.

3.2 2. Data Cleaning

Now that your environment is ready, open a Python file or a Jupyter Notebook and begin by importing the necessary libraries:

```
import pandas as pd
import numpy as np
```

Next, you need to load the dataset into Python. If the dataset is provided in CSV format, you can load it using Pandas:

```
df = pd.read_csv('path_to_your_dataset.csv')
```

Once the data is loaded, your task is to clean it:

- Check for missing values by using:

  ```
  df.isnull().sum()
  ```

  This will show if any columns have missing values. You can handle them by either removing the rows with missing values or filling them in with appropriate values:

  ```
  df.dropna()  # To drop rows with missing values
  df.fillna(value=0)  # To fill missing values with 0
  ```

- Verify that the `Start_Time` and `End_Time` columns are properly formatted as dates:

  ```
  df['Start_Time'] = pd.to_datetime(df['Start_Time'])
  df['End_Time'] = pd.to_datetime(df['End_Time'])
  ```

- Ensure the `Duration` column is consistent with `Start_Time` and `End_Time`. You can calculate the duration and compare:

  ```
  df['Calculated_Duration'] = (df['End_Time'] - df['Start_Time']).dt.total_seconds() / 3600
  ```

  If there are discrepancies, you might need to correct the data.

3.3 3. Exploratory Data Analysis (EDA)

Start analyzing the dataset to get a sense of the production steps and faults:

- Calculate the average time for each production step:

```
avg_duration_per_step = df.groupby('Step_ID')['Duration'].mean()
print(avg_duration_per_step)
```

- Check how many faults occurred during each step:

```
faults_per_step = df.groupby('Step_ID')['Fault_Flag'].sum()
print(faults_per_step)
```

3.4 4. Visualizing the Data

Creating visualizations will help in understanding trends better:

- To create a bar chart of the average duration per step, use Matplotlib:

```
import matplotlib.pyplot as plt
avg_duration_per_step.plot(kind='bar')
plt.title('Average Duration per Production Step')
plt.xlabel('Step ID')
plt.ylabel('Duration (hours)')
plt.show()
```

- Visualize the faults in each step using a bar chart:

```
faults_per_step.plot(kind='bar', color='red')
plt.title('Number of Faults per Production Step')
plt.xlabel('Step ID')
plt.ylabel('Number of Faults')
plt.show()
```

3.5 5. Bottleneck Identification

You can identify bottlenecks by looking at the steps that take the longest:

- From the average duration per step calculated earlier, the step with the highest duration represents a potential bottleneck.
- Additionally, compare the number of faults in each step with the duration to see if there's a correlation.

3.6 6. Operator and Machine Performance Analysis

To evaluate operator and machine performance:

- Calculate the average duration per operator:

```
avg_duration_per_operator = df.groupby('Operator_ID')['Duration'].mean()
avg_duration_per_operator.plot(kind='bar')
plt.title('Average Duration per Operator')
plt.xlabel('Operator ID')
plt.ylabel('Duration (hours)')
plt.show()
```

- Do the same for machines to check their efficiency:

```
avg_duration_per_machine = df.groupby('Machine_ID')['Duration'].mean()
avg_duration_per_machine.plot(kind='bar')
plt.title('Average Duration per Machine')
plt.xlabel('Machine ID')
plt.ylabel('Duration (hours)')
plt.show()
```

3.7 7. Fault Analysis

Investigating the steps and machines with the highest number of faults:

- Visualize the faults per machine:

```
faults_per_machine = df.groupby('Machine_ID')['Fault_Flag'].sum()
faults_per_machine.plot(kind='bar', color='orange')
plt.title('Number of Faults per Machine')
plt.xlabel('Machine ID')
plt.ylabel('Number of Faults')
plt.show()
```

- Use similar methods to analyze which steps or operators experience the most faults.

3.8 8. Recommendations

Finally, write a summary of your findings and recommendations based on the analysis:

- Identify the bottlenecks and suggest ways to address them.
- Recommend improvements for machines or operators that have high fault rates.
- Suggest overall process optimizations, such as additional operator training or machine maintenance.

4 Expected Deliverables

By the end of the assignment, you should have the following:

- A cleaned dataset after addressing missing or incorrect values.
- Visualizations (charts) that show trends in the data for steps, operators, machines, and faults.
- A short report (1 page) summarizing your findings and recommendations.

5 Tools You Can Use

- **Python** with libraries such as Pandas for data manipulation, and Matplotlib or Seaborn for data visualization.
- For creating reports, you can use a word processor like **Microsoft Word** or **Google Docs**.