Project Submission:
Milestone Project 1

Project Motivation:

Analyze and predict internal failures of components in a manufacturing assembly line based on various test conditions. The reason behind taking this project was to help the company produce quality products at lower manufacturing cost while avoiding product recalls later.

"How to avoid manufacturing of defective products in an assembly line?" kept us going.

Specific questions: This can be divided in to 3 sub-sections "Category, Date and Numeric":

1. Category: What is the relationship between components and each production line? How many components were being assembled per line, what were those components and how frequently? The goal was also to explore how we can connect processing of components with assembly time as well as inspection test conditions (pass/fail).
2. Datetime and Numeric Response: How to extract summary statistics per line such as mean, standard deviation between response target and assembly time as well as on an average how many test observations contributed to the summary statistics? Since, the dataset was large as well as sparse, this approach was useful but at the expense of processing time.
3. Combined analysis: What is the breakdown of lines which are failing test inspections purely based on numeric responses without any merge? Then, how should we merge extracted summary from category/datetime and Numeric analysis, and see has anything changed. Example, do we have fewer number of lines after the merge? If yes, then why and if that is significant then can we do similarity analysis with the rest?

Data Sources and data properties: The dataset was available on Kaggle. Please see link https://www.kaggle.com/c/bosch-production-line-performance/data

The dataset, due to its large nature, was split into 6 files: **train_date.csv, train_numeric.csv, train_categorical.csv, test_date.csv, test_numeric.csv, test_categorical.csv,** that contains train data to build the model and test data to validate the model. Since train and test sets will have the same input features, we were only explaining the training dataset.
More than 14 GB data (split 7 GB train and 7 GB test) was stored in 3 csv files (ALL STRINGS), had various column labels, significant amount of missing values, highly imbalanced target class (99% passed Vs 1% failed inspection).
It was extremely difficult to load or reshape and merge all 3 csv files on the local machine.

Overview:

- 4 production lines L0 to L3 , 52+ processing stations S0 to S51, 4268 features in 3 csv files and 1,183,747 rows (x 3) in each training table for merge.

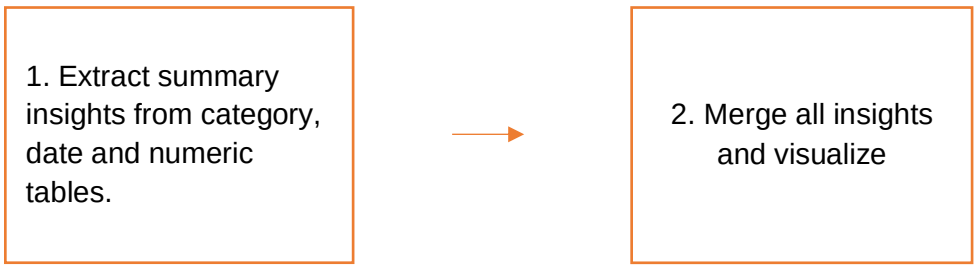
A. **train_categorical.csv:** This file contained LSF features with components (T_1, T2, T3 etc.) along with null values, in some row instances > 95%.
   - Size: 2.1gb, Shape: 1183747*2141

B. **train_date.csv:** This file contained LSD features with timestamp. These timestamps were represented in numerical values between 0-1718 sprawled across the table with more than 90% null values in between.
   - Size: 2.9gb, Shape: 1183747*1157

C. **train_numeric.csv:** This file contained LSF features with test values between -1 to 1
   - Size: 2.7gb, Shape: 1183747*970

Data Manipulation Methods: For each of your two sources, describe how you manipulated the data. For example:

- **Approach taken to summarize full dataset:** Due to large file size, we had to use a combination of pyspark function and python lists and dictionaries to summarize csv files, then use pandas to further manipulate, merge and make plots.
- **Processing Steps:** After creating spark session, we used a simple function which collected assembly component counts and its usage per production line while avoiding null values. Then, extracted summary statistics as python list or dictionary as seem fit to store results as a csv file for further analysis.
- **Data Joining Schema:** After extracting information from individual tables, in order to connect the 3 csv files, we used production line LSF integer values as foreign keys to establish the relationship. The first merge operation was between category table and the numeric table which helped us gather response variable (pass/fail) per production line. Then, we merged again to see assembly times. However, it was interesting to note that the final merged table had only 22% of the production lines where assembly time and test observations were captured, making us believe that these lines probably acted as a node network for performing critical operations such as assembly of sub-assemblies, which company wanted to keep a watch on.
- **Source Code workflow:** Below is the workflow diagram. In first block "extract", we created 3 notebooks to summarize summary insights such as count, mean, spread, usage for each file. Then, in the second block we merged this information to see which production lines are failing more often, and if there is a similarity pattern with the rest. The main objective was to categorize maintenance severity and identify which lines should be maintained on a priority to avoid flow disruption.



- **Challenges and solution:** During extraction stage, summarizing insights took long compute time, and had to leave the computer running whole night and check results in the morning. We had to run multiple iterations which took days to ensure that summary insights are accurate and useful. Example, category table took the longest time to extract nested list of assembly sub-components and storing results as a csv file.

Also, it was not clear initially how to connect 4000+ different column labels and build a consolidated view. Should we use cloud service and database or local machine to handle data processing? We chose to do this analysis on our local machine with one team member working on sampling study while the other team member on full dataset. Example: one approach for generating foreign keys between tables was to use regex to extract numbers from LSF and converting it into an integer value i.e. L1_S2_F34 would be written as 1234. Then adding or subtracting 1 to connect three tables. It was interesting to think, how company would have collected sensory data for those 22% lines. They probably had to program each sensor and label it differently with a tag number to follow their production flow, which may be proprietary. Hence, joining of tables was open to own's interpretation and not fixed on company's process flow.

Analysis and Visualization:

Combined Analysis and data visualization: Approach on summarizing full dataset involved creating 3 individual notebooks and one combined notebook. The names of these notebooks are as under. The second block uses summary insights from the first block for combined analysis.

1. Extract: Bosch_Category Analysis, Bosch_Numeric Analysis, Bosch_Date Analysis
2. Merge/ visualize: Bosch_Combined Analysis.

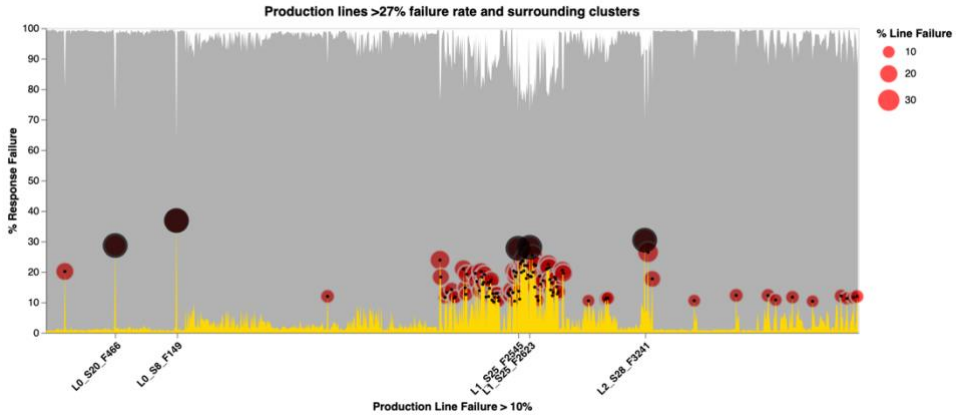
The main goal was to identify breakdown of production lines which were failing often, and then categorize maintenance severity based on similarity pattern with the rest.

To do this, here are the key steps taken to answer specific questions:

1. Prepare foreign keys for date and numeric table in alignment with the category table.
2. Perform feature engineering on summarized numeric table, so that we can estimate and plot percentage breakdown of all line failures prior to the merge. Here is the plot showing the breakdown of line failures by percentage. We used altair area and circle markings to generate this plot.

   Chart Observations: It was interesting to see the formation of clusters around the lines greater than 27% failure rate.

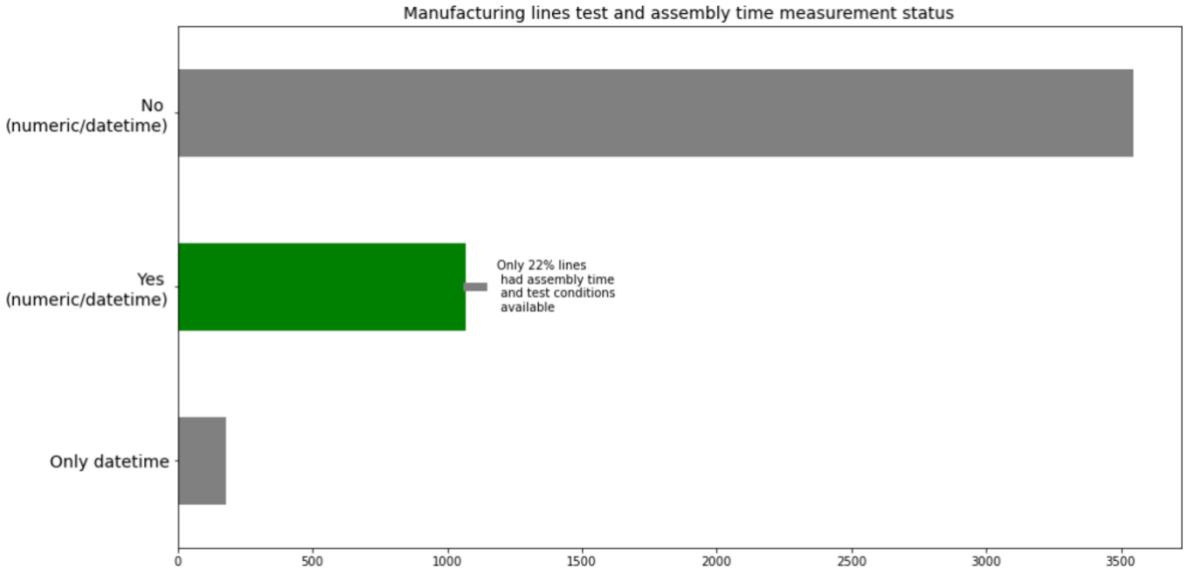
   - From chart above, big black bubbles show top 5 lines failure ranging from 27% to 36%.
   - Size of red bubbles indicate % failure of 10%, 20%, 30% and more.
   - Small black points, indicate formation of clusters mid-way in the graph.



3. Merge Tables using schema to build a consolidated view of category, assembly time and response variable, which resulted in fewer production lines where assembly time and test observation were available. Here is the horizontal bar chart drawn in matplot showing the breakdown of lines with/without these measurements.

Chart Observations:

- Based on above, 22% lines had both assembly time and test observations.
- It is possible, these lines represent assembly of sub-assembly operations, hence maintaining these would be critical.



4. Estimate similarity pattern with the maximum failure line using cosine similarity. Here we used summ ary insights such as mean of test conditions, assembly time as cosine features, which could only be po ssible after merging the tables.

Here is the heatmap and distribution plot made using seaborn library, showing breakdown of similarit y pattern and distribution of critical production lines with the maximum failure line L1_S25_F2546 at 28%.

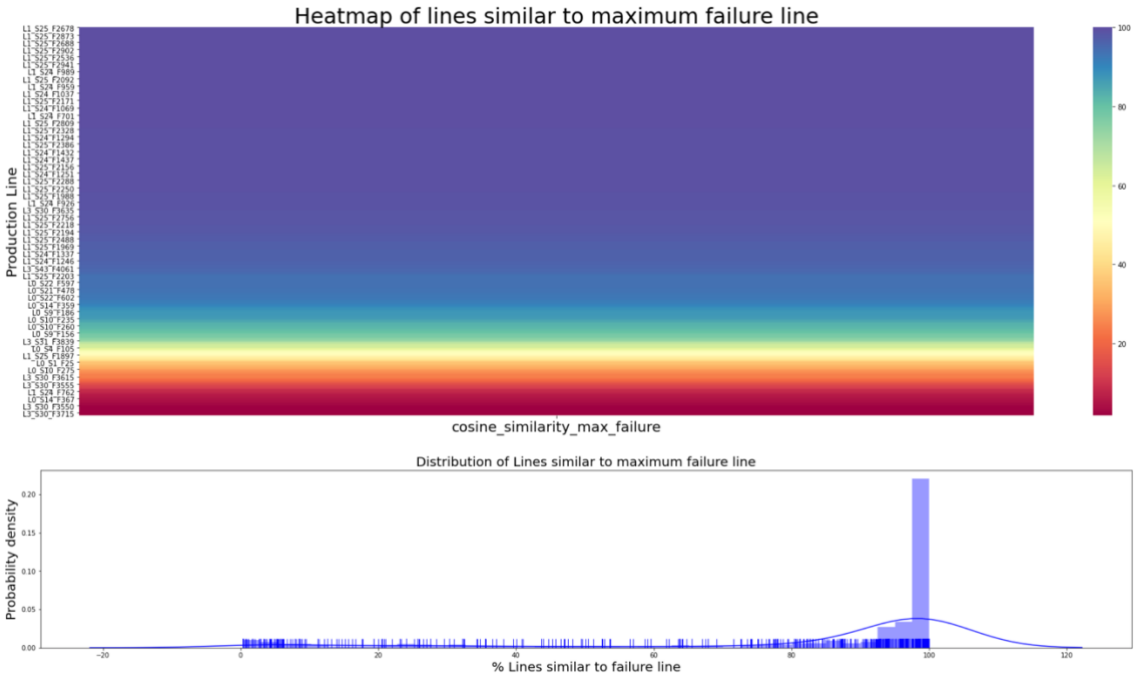


Chart Observations:

- From heatmap above, we can spot % similarity of manufacturing lines with the maximum failure line.
- Example: Lines closer to 100% similarity are represented by red color at the top of the heatmap, while lines under 20% similarity are placed at the bottom of the heatmap.
- Furthermore, from distribution plot, we noticed a bi-modal distribution in similarity pattern, which suggests that lines at these similarity levels in a breakdown situation is likely to impact neighboring lines more closely than others. This was a similar pattern we observed from the 'plot of line failures' earlier during numerical analysis, where we noticed formation of clusters mid-way in the graph. It's interesting to see that a more similar and prominent pattern may occur using cosine similarity, for other percentages of maximum line defect.

5. To investigate this further, let's us categorize maintenance severity in to 3 quartiles (high, medium, low) and plot it so that we can see which lines should be maintained on a priority basis to minimize disruption. Here we used altair boxplot, and circle markings to create this plot.

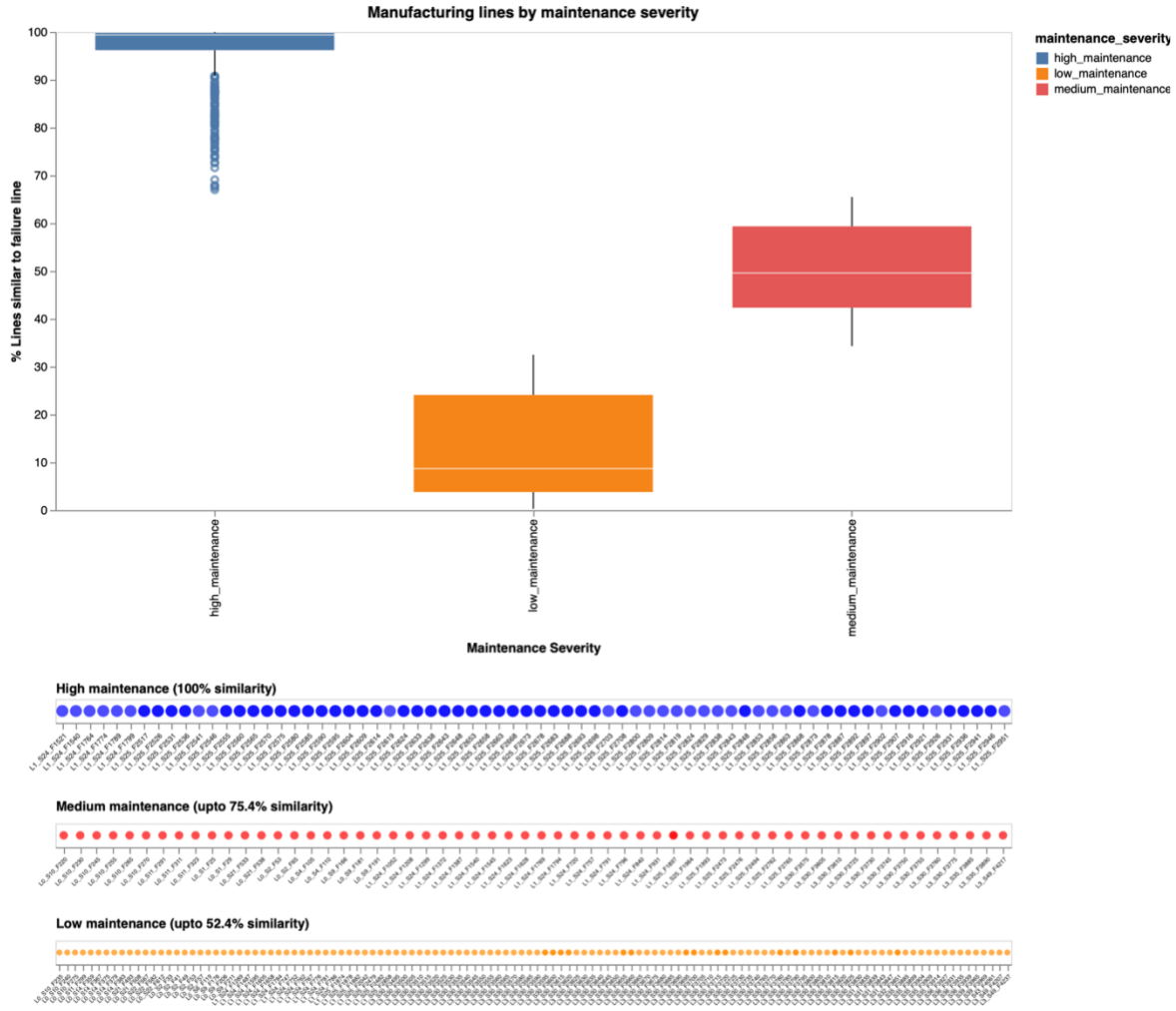


Chart Observations

- There are 115 high maintenance severity lines, which showed 100% similarity with the maximum failure line 'L1_S25_F2546' at 28%. This represents 11% lines which might require immediate maintenance to improve existing production disruption.
- In real environment, instead of using a single line as a maximum failure line, we can use a combination of failure lines (as an investigative tool) to see which other lines share similar attributes and would likely require more maintenance in future.
- We can also add and remove cosine features to study similarity patterns based on user needs.