Devskiller Test

Unified View build

Dan A. Tshisungu

2024-09-19

Table of contents

Introduction

A certain foundry is producing some automotive parts and they would like to know the casting temperature, silicon content and furnace pressure relevant to each part, for a specific period of production.

This data has been collected in the following dataframes:

- production_data_df: contains the unique_part_identifer, cycle_start_timestamp and PART_TYPE
- pressure_data_df: contains the pressure profile recorded by the pressure_sensor
- temperature_data_df: contains the casting Temperature
- silicon_data_df: contains the furnace_silicon_content

Objective:

The objective is to build a unified view function that will extract all relevant information for each part produced.

You will have to build a dataframe, a ground truth, where each row represents a single part and each column represents the relevant data from each process that needs to be extracted and aligned.

About the dataset

- **production_data_df:** Parts are produced per batch of the same product type. A part is produced every 30 minutes and has a unique part identifier. Batches of the same product type are identified with the product type name which is logged on the control system when the type changes and will apply from the next part.
- **pressure_data_df:** During casting, the pressure sensor logs the exerted pressure every 10 seconds (the pressure increases during this time), until the casting is completed. The result is a pressure cycle that lasts for ~30 minutes and resets when casting is completed. You will need to extract the maximum pressure reached for each cycle and the time elapsed (in minutes) from when to production cycle starts to the moment this peak is reached
- **temperature_data_df:** The casting temperature is recorded at the beginning of the casting cycle (in the first 10 minutes)
- **silicon_data_df:** Bulk quantities of metal are melted in the furnace, and as a result, the chemistry remains relatively similar for a few hours of production, and is therefore recorded less frequently. The silicon in particular is recorded every ~4 hours, and should apply for the next 4 hours of production.

1. Load data

```r
library(arrow)
library(tidyverse)
library(dplyr)
library(tidyr)
library(here)
library(janitor)
library(reticulate)
library(knitr)
library(lubridate)

## Casting temperature data
casting_df <-
  read_parquet(here("data",
                    "casting_temperature_data.parquet"))%>%
  clean_names() %>%
  select(everything(), timestamp = index_level_0) %>%
  mutate(timestamp = ymd_hms(timestamp))

## Furnace silicon data
furnace_df <-
  read_parquet(here("data", "furnace_silicon_data.parquet")) %>%
  clean_names() %>%
  select(everything(), timestamp = index_level_0) %>%
  mutate(timestamp = ymd_hms(timestamp))

## Pressure data
pressure_df <-
  read_parquet(here("data", "pressure_data.parquet")) %>%
  clean_names() %>%
  select(everything(), timestamp = index_level_0) %>%
  mutate(timestamp = ymd_hms(timestamp))

## Production logging data

production_df <-
  read_parquet(here("data", "production_logging_data.parquet")) %>%
  clean_names() %>%
  select(everything(), timestamp = index_level_0) %>%
  mutate(timestamp = ymd_hms(timestamp))

cat("We have 4 datasets and below are their dimensions: \n",
    "- Casting temperature data", nrow(casting_df), "columns and ", ncol(casting_df),"variabl
```

```
    "- Furnace silicon data", nrow(furnace_df), "columns and ", ncol(furnace_df),"variables,
    "- Pressure data", nrow(pressure_df), "columns and ", ncol(pressure_df),"variables and,\n
    "- Production logging data",nrow(production_df),"columns and ",ncol(production_df),"varia
```

```
We have 4 datasets and below are their dimensions:
 - Casting temperature data 49 columns and  2 variables,
 - Furnace silicon data 6 columns and  2 variables,
 - Pressure data 8641 columns and  2 variables and,
 - Production logging data 51 columns and  4 variables.
```

2. Implementation

2.1. Block 1: Preprocess "production data"

We just fill in the part_type based on the setup initial part and we remove the two lines of the setup.

> **Tip**
>
> We just have to be sure that the final dataframe has 49 rows and 4 columns with no NA values as it must be a dense dataframe.

R

```
production_r_clean <-
  production_df %>%
  tidyr::fill(part_type, .direction = "down") %>%
  drop_na() %>%
  rename(cycle_start = cycle_start_timestamp)

production_r_clean
```

```
# A tibble: 49 x 4
   unique_part_identifer cycle_start         part_type       timestamp
   <chr>                 <dttm>              <chr>           <dttm>
 1 uid_11285             2024-01-15 02:00:00 BMW_LARGE_WHEEL 2024-01-15 02:00:00
 2 uid_11286             2024-01-15 02:30:00 BMW_LARGE_WHEEL 2024-01-15 02:30:00
 3 uid_11287             2024-01-15 03:00:00 BMW_LARGE_WHEEL 2024-01-15 03:00:00
 4 uid_11288             2024-01-15 03:30:00 BMW_LARGE_WHEEL 2024-01-15 03:30:00
```

```
 5 uid_11289             2024-01-15 04:00:00 BMW_LARGE_WHEEL 2024-01-15 04:00:00
 6 uid_11290             2024-01-15 04:30:00 BMW_LARGE_WHEEL 2024-01-15 04:30:00
 7 uid_11291             2024-01-15 05:00:00 BMW_LARGE_WHEEL 2024-01-15 05:00:00
 8 uid_11292             2024-01-15 05:30:00 BMW_LARGE_WHEEL 2024-01-15 05:30:00
 9 uid_11293             2024-01-15 06:00:00 BMW_LARGE_WHEEL 2024-01-15 06:00:00
10 uid_11294             2024-01-15 06:30:00 BMW_LARGE_WHEEL 2024-01-15 06:30:00
# i 39 more rows
```

Python

```python
import pandas as pd

production_df = r.production_df

## Fill the "part_type"
production_df['part_type'] = production_df['part_type'].ffill()

# Drop NA
production_df_clean = production_df.dropna()

# rename cycle time
production_df_clean = production_df_clean.rename(columns={'cycle_start_timestamp': 'cycle_sta

print(production_df_clean.head(10))
```

```
   unique_part_identifer  ...           timestamp
1              uid_11285  ... 2024-01-15 02:00:00
2              uid_11286  ... 2024-01-15 02:30:00
3              uid_11287  ... 2024-01-15 03:00:00
4              uid_11288  ... 2024-01-15 03:30:00
5              uid_11289  ... 2024-01-15 04:00:00
6              uid_11290  ... 2024-01-15 04:30:00
7              uid_11291  ... 2024-01-15 05:00:00
8              uid_11292  ... 2024-01-15 05:30:00
9              uid_11293  ... 2024-01-15 06:00:00
10             uid_11294  ... 2024-01-15 06:30:00

[10 rows x 4 columns]
```

2.2. Block 2: Preprocess "pressure data" + Join

Here have to do 2 things:

- preprocess the dataframe
- join it to the production data above

2.2.1. Preprocessing

R

```r
pressure_r_clean <-
  pressure_df %>%
  mutate(
    start_time = min(timestamp),
    time_diff = as.numeric(difftime(timestamp, start_time, units = "mins")),
    cycle_number = floor(time_diff / 30) + 1
    ) %>%
  group_by(cycle_number) %>%
  mutate(
    cycle_start_time = min(timestamp),
    time_to_max = difftime(
      timestamp[which.max(pressure_sensor)],
      cycle_start_time,
      units = "mins"
    ),
    max_pressure = max(pressure_sensor)
  ) %>%
  ungroup() %>%
  select(cycle_start_time,
         max_pressure,
         time_to_max) %>%
  distinct(cycle_start_time, .keep_all = TRUE)

pressure_r_clean
```

```
# A tibble: 49 x 3
   cycle_start_time    max_pressure time_to_max
   <dttm>                     <dbl> <drtn>
 1 2024-01-15 02:00:00         51.5 17.66667 mins
 2 2024-01-15 02:30:00         52.0 20.16667 mins
```

```
 3 2024-01-15 03:00:00         50.5 21.83333 mins
 4 2024-01-15 03:30:00         50.3 17.50000 mins
 5 2024-01-15 04:00:00         50.8 16.66667 mins
 6 2024-01-15 04:30:00         51.8 20.66667 mins
 7 2024-01-15 05:00:00         51.3 20.33333 mins
 8 2024-01-15 05:30:00         50.8 16.66667 mins
 9 2024-01-15 06:00:00         53.3 20.00000 mins
10 2024-01-15 06:30:00         50.3 20.33333 mins
# i 39 more rows
```

Python

```python
import pandas as pd

pressure_df = r.pressure_df

pressure_df['timestamp'] = pd.to_datetime(pressure_df['timestamp'])

## create the cycle_number
pressure_df['cycle_number'] = (pressure_df['timestamp'] - pressure_df['timestamp'].min()).dt

#print(pressure_df['cycle_number'].unique())

# cycle start time
pressure_df['cycle_start_time'] = pressure_df.groupby('cycle_number')['timestamp'].transform

# Max pressure in a cycle
pressure_df['max_pressure'] = pressure_df.groupby('cycle_number')['pressure_sensor'].transfo


# Isolate the rows where the pressure is equal to the max pressure for each cycle
df_max_pressure = pressure_df[pressure_df['pressure_sensor'] == pressure_df['max_pressure']]


# Calculate the 'time_to_max' in minutes from cycle_start_time to the timestamp where max_pr
df_max_pressure['time_to_max'] = (df_max_pressure['timestamp'] - df_max_pressure['cycle_start

# final
pressure_clean = df_max_pressure[['cycle_start_time', 'max_pressure', 'time_to_max']]

print(pressure_clean.head(10))
```

```
        cycle_start_time  max_pressure  time_to_max
106  2024-01-15 02:00:00     51.507711    17.666667
301  2024-01-15 02:30:00     52.036447    20.166667
491  2024-01-15 03:00:00     50.503723    21.833333
645  2024-01-15 03:30:00     50.345908    17.500000
820  2024-01-15 04:00:00     50.826550    16.666667
1024 2024-01-15 04:30:00     51.776302    20.666667
1202 2024-01-15 05:00:00     51.331237    20.333333
1360 2024-01-15 05:30:00     50.800816    16.666667
1560 2024-01-15 06:00:00     53.321533    20.000000
1742 2024-01-15 06:30:00     50.282106    20.333333
```

2.2.2. Join Production to Pressure

Now let us join the production data to the pressure data

R

```r
production_joined <- production_r_clean %>%
  dplyr::left_join(pressure_r_clean, join_by(timestamp == cycle_start_time))

production_joined
```

```
# A tibble: 49 x 6
   unique_part_identifer cycle_start         part_type       timestamp
   <chr>                 <dttm>              <chr>           <dttm>
 1 uid_11285             2024-01-15 02:00:00 BMW_LARGE_WHEEL 2024-01-15 02:00:00
 2 uid_11286             2024-01-15 02:30:00 BMW_LARGE_WHEEL 2024-01-15 02:30:00
 3 uid_11287             2024-01-15 03:00:00 BMW_LARGE_WHEEL 2024-01-15 03:00:00
 4 uid_11288             2024-01-15 03:30:00 BMW_LARGE_WHEEL 2024-01-15 03:30:00
 5 uid_11289             2024-01-15 04:00:00 BMW_LARGE_WHEEL 2024-01-15 04:00:00
 6 uid_11290             2024-01-15 04:30:00 BMW_LARGE_WHEEL 2024-01-15 04:30:00
 7 uid_11291             2024-01-15 05:00:00 BMW_LARGE_WHEEL 2024-01-15 05:00:00
 8 uid_11292             2024-01-15 05:30:00 BMW_LARGE_WHEEL 2024-01-15 05:30:00
 9 uid_11293             2024-01-15 06:00:00 BMW_LARGE_WHEEL 2024-01-15 06:00:00
10 uid_11294             2024-01-15 06:30:00 BMW_LARGE_WHEEL 2024-01-15 06:30:00
# i 39 more rows
# i 2 more variables: max_pressure <dbl>, time_to_max <drtn>
```

Python

```
import pandas as pd

production_joined_df = pd.merge(production_df_clean,
pressure_clean,left_on='timestamp', right_on='cycle_start_time', how='left')

print(production_joined_df.head(10))
```

```
  unique_part_identifer         cycle_start  ... max_pressure time_to_max
0             uid_11285 2024-01-15 00:00:00  ...    51.507711   17.666667
1             uid_11286 2024-01-15 00:30:00  ...    52.036447   20.166667
2             uid_11287 2024-01-15 01:00:00  ...    50.503723   21.833333
3             uid_11288 2024-01-15 01:30:00  ...    50.345908   17.500000
4             uid_11289 2024-01-15 02:00:00  ...    50.826550   16.666667
5             uid_11290 2024-01-15 02:30:00  ...    51.776302   20.666667
6             uid_11291 2024-01-15 03:00:00  ...    51.331237   20.333333
7             uid_11292 2024-01-15 03:30:00  ...    50.800816   16.666667
8             uid_11293 2024-01-15 04:00:00  ...    53.321533   20.000000
9             uid_11294 2024-01-15 04:30:00  ...    50.282106   20.333333

[10 rows x 7 columns]
```

2.3. Block 3: Join Casting to Joined_data

Here we will just perform a join and keep relevant factor.

> **Tip**
>
> The variables onto which we must do the join, on the contrary of previous join, are not exactly the same. For example timestamp from production is 02:00:00 and timestamp from casting is 02:00:13. The pattern repeats itself with the second being equal or slightly higher than the second.

R

```
production_joined_cast <-
  production_joined %>%
  left_join(casting_df, join_by(closest(timestamp <= timestamp)),
            suffix = c(".exact", ".approximative")) %>%
```

```
  select(unique_part_identifer,part_type,cycle_start, max_pressure, time_to_max,
         casting_temperature, timestamp.exact, timestamp.approximative)

production_joined_cast
```

```
# A tibble: 49 x 8
   unique_part_identifer part_type  cycle_start         max_pressure time_to_max
   <chr>                 <chr>      <dttm>                     <dbl> <drtn>
 1 uid_11285             BMW_LARGE~ 2024-01-15 02:00:00         51.5 17.66667 m~
 2 uid_11286             BMW_LARGE~ 2024-01-15 02:30:00         52.0 20.16667 m~
 3 uid_11287             BMW_LARGE~ 2024-01-15 03:00:00         50.5 21.83333 m~
 4 uid_11288             BMW_LARGE~ 2024-01-15 03:30:00         50.3 17.50000 m~
 5 uid_11289             BMW_LARGE~ 2024-01-15 04:00:00         50.8 16.66667 m~
 6 uid_11290             BMW_LARGE~ 2024-01-15 04:30:00         51.8 20.66667 m~
 7 uid_11291             BMW_LARGE~ 2024-01-15 05:00:00         51.3 20.33333 m~
 8 uid_11292             BMW_LARGE~ 2024-01-15 05:30:00         50.8 16.66667 m~
 9 uid_11293             BMW_LARGE~ 2024-01-15 06:00:00         53.3 20.00000 m~
10 uid_11294             BMW_LARGE~ 2024-01-15 06:30:00         50.3 20.33333 m~
# i 39 more rows
# i 3 more variables: casting_temperature <dbl>, timestamp.exact <dttm>,
#   timestamp.approximative <dttm>
```

Python

```
import pandas as pd

casting_df = r.casting_df

production_joined_cast_df = pd.merge_asof(
    production_joined_df,
    casting_df,
    on='timestamp',
    direction='forward',
    suffixes=('.exact', '.approximative')
)


print(production_joined_cast_df.head(10))
```

```
  unique_part_identifer         cycle_start  ... time_to_max casting_temperature
```

```
0             uid_11285 2024-01-15 00:00:00  ...   17.666667          710.784843
1             uid_11286 2024-01-15 00:30:00  ...   20.166667          711.426896
2             uid_11287 2024-01-15 01:00:00  ...   21.833333          710.703687
3             uid_11288 2024-01-15 01:30:00  ...   17.500000          711.343364
4             uid_11289 2024-01-15 02:00:00  ...   16.666667          710.481132
5             uid_11290 2024-01-15 02:30:00  ...   20.666667          711.396336
6             uid_11291 2024-01-15 03:00:00  ...   20.333333                 NaN
7             uid_11292 2024-01-15 03:30:00  ...   16.666667          711.505411
8             uid_11293 2024-01-15 04:00:00  ...   20.000000          712.290626
9             uid_11294 2024-01-15 04:30:00  ...   20.333333          712.083290

[10 rows x 8 columns]
```

2.4. Block 4 : Join Furnace to Joined_data

We perform again a join, but we keep in mind that here, it is the first timestamp that is higher or equal to the second.

R

```r
production_final <- production_joined_cast %>%
  left_join(furnace_df, join_by(closest(timestamp.exact >= timestamp)),
            suffix = c(".init", ".app"))

production_final
```

```
# A tibble: 49 x 10
   unique_part_identifer part_type  cycle_start         max_pressure time_to_max
   <chr>                 <chr>      <dttm>                     <dbl> <drtn>
 1 uid_11285             BMW_LARGE~ 2024-01-15 02:00:00         51.5 17.66667 m~
 2 uid_11286             BMW_LARGE~ 2024-01-15 02:30:00         52.0 20.16667 m~
 3 uid_11287             BMW_LARGE~ 2024-01-15 03:00:00         50.5 21.83333 m~
 4 uid_11288             BMW_LARGE~ 2024-01-15 03:30:00         50.3 17.50000 m~
 5 uid_11289             BMW_LARGE~ 2024-01-15 04:00:00         50.8 16.66667 m~
 6 uid_11290             BMW_LARGE~ 2024-01-15 04:30:00         51.8 20.66667 m~
 7 uid_11291             BMW_LARGE~ 2024-01-15 05:00:00         51.3 20.33333 m~
 8 uid_11292             BMW_LARGE~ 2024-01-15 05:30:00         50.8 16.66667 m~
 9 uid_11293             BMW_LARGE~ 2024-01-15 06:00:00         53.3 20.00000 m~
10 uid_11294             BMW_LARGE~ 2024-01-15 06:30:00         50.3 20.33333 m~
# i 39 more rows
```

```
# i 5 more variables: casting_temperature <dbl>, timestamp.exact <dttm>,
#   timestamp.approximative <dttm>, furnace_silicon_content <dbl>,
#   timestamp <dttm>
```

Python

```
import pandas as pd

furnace_df = r.furnace_df


production_final_df = pd.merge_asof(
    production_joined_cast_df,
    furnace_df,
    on='timestamp',
    direction='backward',
    suffixes=('.init', '.appro')
)

print(production_final_df.head(10))
```

```
  unique_part_identifer  ... furnace_silicon_content
0             uid_11285  ...                     1.2
1             uid_11286  ...                     1.2
2             uid_11287  ...                     1.2
3             uid_11288  ...                     1.2
4             uid_11289  ...                     1.2
5             uid_11290  ...                     1.2
6             uid_11291  ...                     1.2
7             uid_11292  ...                     1.2
8             uid_11293  ...                     1.4
9             uid_11294  ...                     1.4

[10 rows x 9 columns]
```

2.5. Block 5 : Critical review (Optimization)

Here we review the whole process and fix some errors if any.

> Based on the observation, we just have to fill in the accurate values for NA in the casting variable.

R

```
production_final
```

```
# A tibble: 49 x 10
   unique_part_identifer part_type  cycle_start         max_pressure time_to_max
   <chr>                 <chr>      <dttm>                     <dbl> <drtn>
 1 uid_11285             BMW_LARGE~ 2024-01-15 02:00:00         51.5 17.66667 m~
 2 uid_11286             BMW_LARGE~ 2024-01-15 02:30:00         52.0 20.16667 m~
 3 uid_11287             BMW_LARGE~ 2024-01-15 03:00:00         50.5 21.83333 m~
 4 uid_11288             BMW_LARGE~ 2024-01-15 03:30:00         50.3 17.50000 m~
 5 uid_11289             BMW_LARGE~ 2024-01-15 04:00:00         50.8 16.66667 m~
 6 uid_11290             BMW_LARGE~ 2024-01-15 04:30:00         51.8 20.66667 m~
 7 uid_11291             BMW_LARGE~ 2024-01-15 05:00:00         51.3 20.33333 m~
 8 uid_11292             BMW_LARGE~ 2024-01-15 05:30:00         50.8 16.66667 m~
 9 uid_11293             BMW_LARGE~ 2024-01-15 06:00:00         53.3 20.00000 m~
10 uid_11294             BMW_LARGE~ 2024-01-15 06:30:00         50.3 20.33333 m~
# i 39 more rows
# i 5 more variables: casting_temperature <dbl>, timestamp.exact <dttm>,
#   timestamp.approximative <dttm>, furnace_silicon_content <dbl>,
#   timestamp <dttm>
```

Python

```
import pandas as pd

print(production_final_df.head(10))
```

```
  unique_part_identifer  ... furnace_silicon_content
0             uid_11285  ...                     1.2
1             uid_11286  ...                     1.2
2             uid_11287  ...                     1.2
3             uid_11288  ...                     1.2
4             uid_11289  ...                     1.2
5             uid_11290  ...                     1.2
6             uid_11291  ...                     1.2
7             uid_11292  ...                     1.2
8             uid_11293  ...                     1.4
9             uid_11294  ...                     1.4

[10 rows x 9 columns]
```

2.6. Function : build_unified_view()

FINAL FUNCTION

R

```
## Build the function by putting things together
```

Python

```
### Build here
```