**Importing Libraries**

* The required libraries are imported for data manipulation, visualization, and database connectivity
* Libraries
* Pandas
* Numpy
* Matplotlib
* Seaborn
* Datetime
* Pyodbc
* Urllib
* Sqlalchemy
* math

**Defining Connection**

* The connection details for the database are defined using the ODBC Driver and connection parameters.
* An engine is created using the defined connection details for executing SQL queries.

**Demand Estimation**

**Description:**

 The **Tape Demand Calculator** class represents a tape demand calculator. It provides methods to read and process fabric recipe data, weaving planning demand data, master fabric and tape data, calculate tape demand, and calculate tape demand data frame. The class also includes methods to read extended master tape data, extended master fabric data, and master tape recipe data. The class also reads and adds buffer plan data to the actual demand.

Here's a detailed breakdown of the class methods and their functionalities:

**Constructor: `\_\_init\_\_(self, engine)`**

Purpose: Initializes the `TapeDemandCalculator` class.

Parameters:

`engine`: A database engine used for making database connections.

Functionality: Stores the database engine and creates a connection to the database.

**Method: `read\_fabric\_recipe(self)`**

Purpose: Reads fabric recipe data from the database.

Functionality: Executes a SQL query to retrieve all records from the `FabricRecipe` table.

**Method: `process\_fabric\_recipe2(self, fab\_recp)`**

Purpose: Processes fabric recipe data, specifically separating warp, weft, and warp R/F data.

Parameters:- `fab\_recp`: DataFrame containing fabric recipe data.

Functionality: Splits the fabric recipe data into warp, weft, and warp R/F (Reinforced) categories, and merges them based on `FabricId`.

**Method: `process\_fabric\_recipe(self, fab\_recp)`**

Purpose: An alternative method for processing fabric recipe data.

Functionality: Filters and pivots the data for 'Warp', 'Weft', and 'Warp R/F', and resets the DataFrame index for clarity.

**Method: `read\_weaving\_planning(self)`**

Purpose: Reads weaving planning data from the database.

Functionality: Retrieves relevant weaving planning demand information from the database.

**Method: `runningFabrics(self, df\_demand)`**

Purpose: Segregates running and non-running fabrics based on the demand DataFrame.

Parameters:

`df\_demand`: DataFrame containing demand data.

Functionality: Filters the DataFrame to separate running fabrics from others.

**Method: `freeze\_fabrics(self, df\_demand)`**

Purpose: Identifies fabrics that are frozen in the schedule.

Parameters:

- `df\_demand`: DataFrame containing demand data.

Functionality: Merges freeze fabric information with the demand data to identify frozen fabrics.

**Method: `final\_demand(self, df\_running, df\_freeze)`**

Purpose: Combines running and frozen fabrics to finalize the demand.

Parameters:

- `df\_running`: DataFrame of running fabrics.

- `df\_freeze`: DataFrame of frozen fabrics.

Functionality: Concatenates the running and frozen fabric data.

**Method: `calculate\_no\_of\_looms(self, df\_demand)`**

Purpose: Calculates the number of looms for each fabric.

Parameters:

- `df\_demand`: DataFrame containing demand data.

Functionality : Determines the unique number of looms used for each fabric.

**Method: `read\_master\_fabric\_data(self)`**

Purpose: Reads master fabric data from the database.

Functionality: Retrieves master fabric information, including fabric width, GSM, mesh, and type.

**Method: `read\_master\_tape\_data(self, df)`**

Purpose: Reads and merges master tape data with the provided DataFrame.

Parameters:

- `df`: DataFrame to merge with tape data.

Functionality: Retrieves tape denier information and merges it with the provided DataFrame.

**Method: `days\_demand(self, df1, confg\_days)`**

Purpose: Calculates the demand for upcoming days.

Parameters:

- `df1`: DataFrame containing demand information.

- `confg\_days`: Configuration for the number of days to consider.

Functionality: Filters the demand according to configured days and Adjusts the fabric balance to make based on the configured number of days.

**Method: `calculate\_tape\_demand2(self, df)`**

Purpose: Calculates the demand for different types of tapes.

Parameters:

-`df`: DataFrame containing fabric and demand information.

- Functionality: Performs calculations to estimate the demand for warp, weft, and warp R/F tapes.As per following:

* 1. Fabric in KG = (Width\*fabric\_quantity(meter)\*gsm)/1000
  2. Warp tape = Fabric in KG \* Warp%/100
  3. Warp rf tape = Fabric in KG \* Warp rf %/100
  4. Weft tape = Fabric in KG \* Weft%/100

**Method: `calculate\_load\_unload(self, dfk)`**

Purpose: Calculates the loading and unloading requirements for tapes.

Parameters:

- `dfk`: DataFrame with fabric and tape information.

-Functionality: Determines the tape load and unload requirements based on changes in warp and weft specifications.

1. Loading/unloading will happen in only warp and warp rf tapes
2. If there is a change in warp id of current and previous production then:
   1. Warp load = warp tapes\*1.5\*warp layers
   2. Warp unload = warp tapes\*0.6\*warp layers
   3. Previous warp unload = previous warp tapes \* 0.6\* previous warp layers
3. If there is no change current warp id and prev warp id then:
   1. If current warp tapes> prev warp tapes:
      1. Warp load = (current warp tapes - prev warp tapes)\*1.5\*current warp layers
      2. Warp unload = current warp layers\*0.6\*current warp layers
      3. Prev warp unload = 0
   2. If current warp tapes < pre warp tapes;
      1. Warp load = 0
      2. Warp unload = current warp tapes\*0.6\*current warp layers
      3. Prev warp unload = (prev warp tapes - current warp tapes)\*prev warp layers
   3. If there is no change in ids:
      1. Warp load =0
      2. Warp unload = 0
      3. Prev warp unload = prev warp layers \* 0.6 \* prev warp layers
4. Same for warp rf tapes
5. After loading unloading:
   1. Total warp demand = warp tape + warp load
   2. Total weft demand = weft tape
   3. Total warp rf demand = warp rf tape + warp rf load

**Method: `total\_demand(self, df)`**

Purpose: Calculates the total demand for warp, weft, and warp R/F tapes.

Parameters:

- `df`: DataFrame with calculated tape demands.

Functionality: Summarizes the total tape demand by adding the load requirements to the calculated tape demands.

**Method: `invent\_df(self, df\_invent\_GR,df\_invent\_D19, demand\_df)`**

Purpose: Adjusts demand based on available inventory.

Parameters:

- `df\_invent\_GR`: Inventory data for Gajner Road location.

- `df\_invent\_D19`: Inventory data for D-19 location.

- `demand\_df`: DataFrame containing demand data.

Functionality: Adjusts the demand calculations based on inventory available at different locations.

**Method: `calculate\_demand\_df(self, demand\_df)`**

Purpose: Finalizes the demand DataFrame.

Parameters:

- `demand\_df`: DataFrame containing demand data.

Functionality: Splits and groups demand data for weft, warp, and warp R/F tapes, and then consolidates it.

**Method: add\_buffer\_demand(df\_demand):**

Purpose :To read and add the buffer plan demand to the weaving planning demand

Parameters : df\_demand : original demand

Functionality : reads the buffer plan demand from the buffer plan table in DB, adds tape specs to the demand like color, denier, uv, rp , filler. Merges buffer demand on df\_demand to get final demand.

**Method: `read\_master\_tape\_data\_extended(self)`**

Purpose: Reads extended master tape data from the database.

Functionality: Retrieves additional tape details, including width, color, denier, and marking.

**Method: `read\_master\_fabric\_data\_extended(self)`**

Purpose: Reads extended master fabric data from the database.

Functionality: Retrieves additional fabric details like warp mesh.

**Method: `read\_master\_tape\_recipe(self)`**

Purpose: Reads master tape recipe data from the database.

Functionality: Retrieves tape recipe information, focusing on the material composition of tapes.

**Method: `process\_demand\_df2(self, demand\_df, dfkk, dfk)`**

Purpose: Processes the demand DataFrame with additional master data.

Parameters:

- `demand\_df`: DataFrame containing demand data.

- `dfkk`: DataFrame containing extended master tape data.

- `dfk`: DataFrame containing extended master fabric data.

Functionality: Merges and processes the demand data with extended master data to extract additional insights like tape color, width, and UV properties.

**Demand Scheduling**

**Description:**

The estimated demand from tape demand calculator is passed to the tape demand scheduler to generate the schedule of estimated demand based on various factors such as loom velocity, changeover time, cost and tape specs such as tape denier, tape color, width, RP, Filler, Tape Type etc. All the scheduling is done on the basis of configured machine rules and constraints. The tape scheduling process is achieved by 3 classes class production, bestmachines and tape scheduler class.

**Class: `Production`**

**Constructor: `\_\_init\_\_(self, prod\_data)`**

Purpose: Initializes the `Production` class.

Parameters:

- `prod\_data`: A DataFrame containing production data.

Functionality: Initializes the class with the production data and sets up an empty DataFrame `prod\_final` for processed results.

**Method: `process\_production(self)`**

Purpose: Filters and combines production data based on specific criteria for each machine.

Functionality: Applies production per hour filters to each tape plant (e.g., Tape plant 1, Tape plant 2, etc.) and then concatenates the filtered data into the `prod\_final` DataFrame.

**Class: `ProductionAnalyzer`**

This class provides functionalities to analyze production data, such as calculating production velocity and analyzing production characteristics of different machines.

**Constructor: `\_\_init\_\_(self, prod\_df)`**

Purpose: Initializes the `ProductionAnalyzer` class.

Parameters:

- `prod\_df`: A DataFrame containing production data.

Functionality: Initializes the class with the production DataFrame.

**Method: `calculate\_velocity(self)`**

Purpose: Calculates the mean production per hour for each combination of machine and tape specifications.

Functionality:

- Groups the production data by `MachineName`, `TapeDenier`, `TapeWidth`, `TapeColour`, `TapeFiller`, and `TapeRP`.

- Calculates the mean production per hour and count of records for each group.

- Categorizes tape width and denier into 'Narrow', 'Standard', 'Wider' and 'Low', 'High', respectively.

- Creates a `TapeSpecs` column combining denier, width, color, filler, and RP for easy reference.

**Method: `used\_machine\_velocity\_by\_tape\_spec(self, Tape\_spec)`**

Purpose: Analyzes which machines are most frequently used for a given tape specification and their average production velocity.

Parameters:

- `Tape\_spec`: A list representing a specific tape specification.

Functionality:

- Converts `Tape\_spec` into a string format and filters the `velocity\_df` DataFrame for matching specifications.

- Sorts the machines based on their mean production velocity for the given tape specification.

- Returns a list of machines and their velocities for the specified tape.

**Method: `last\_production(self, tape\_prod)`**

Purpose: Retrieves the last production record for each tape plant.

Parameters:

- `tape\_prod`: A DataFrame containing tape production data.

Functionality:

- Iterates through tape plants (excluding Tape plant 3) and filters the `tape\_prod` DataFrame for the latest production record.

- Extracts relevant information from the latest production record, including tape ID, name, denier, color, UV, RP, filler, production end time, and width.

- Processes the tape color to extract a simplified color name.

- Returns a list of tuples containing the latest production details for each tape plant.

**Class: `Bestmachines`**

The `Bestmachines` class is designed to schedule tape production based on various constraints and machine availability. It aims to optimize the allocation of tapes to different machines while considering their specifications and production capacities. Here's a breakdown of the class and its methods:

**Constructor: `\_\_init\_\_(self, demand\_df, tape\_data\_df, last\_production, analyzer, plant)`**

Purpose: Initializes the `Bestmachines` class with required data.

Parameters:

- `demand\_df`: DataFrame containing tape demand information.

- `tape\_data\_df`: DataFrame containing detailed tape data.

- `last\_production`: List containing the last production data for each machine.

- `analyzer`: An instance of `ProductionAnalyzer` used for velocity calculations.

- `plant`: DataFrame containing plant details.

Functionality: Initializes class variables, sets up schedules, and identifies machines available for colored tape production.

**Method: `get\_machine\_by\_denier\_width(self, denier, width)`**

Purpose: Determines the best machines for a given tape denier and width.

Parameters:

- `denier`: The denier of the tape.

- `width`: The width of the tape.

Functionality: Filters plant details to find machines suitable for the given tape specifications and sorts them by production velocity.

**Method: `sort\_tape\_ids\_by\_nearest\_width(self, target\_id)`**

Purpose: Sorts tape IDs based on their closeness in width to a target tape ID.

Parameters:

- `target\_id`: The target tape ID.

Functionality: Orders tape IDs by the similarity of their widths to the target tape's width.

**Method: `width\_can\_be\_used(self, width, denier)`**

Purpose: Determines suitable machines based on tape width and denier.

Parameters:

- `width`: The width category of the tape ('Narrow', 'Standard', or 'Wider').

- `denier`: The denier of the tape.

Functionality: Suggests appropriate machines for a given tape width and denier category.

**Method: `schedule\_tapes(self, analyzer)`**

Purpose: Schedules tapes on the best-suited machines based on various criteria.

Parameters:

- `analyzer`: An instance of `ProductionAnalyzer`.

Functionality:

- Iterates through the demand DataFrame to schedule each tape.

- Applies constraints based on tape specifications and machine capabilities.

- Selects machines that can produce the required tape as quickly as possible without exceeding capacity limits.

- Updates the machine availability time based on the scheduling.

- Stores scheduled and not-scheduled tape details.

**Key Functional Points**

1. **Scheduling Algorithm**: The class iteratively schedules each tape based on its specifications and the capabilities of available machines. It considers factors like tape color, width, denier, filler, and RP (Reinforced Plastic) content.

2. **Machine Availability**: The algorithm updates the availability of each machine after scheduling a tape, ensuring that the machine capacity is not exceeded.

3. **Handling Special Cases**: The class has specific rules for handling tapes with unique requirements, such as high denier or specific colors.

4. **Optimization Criteria**: The scheduling is optimized based on the production velocity of each machine and the total load on the machines.

5. **Not Scheduled Tapes**: Tapes that cannot be scheduled due to various constraints are tracked separately with reasons for non-scheduling.

**Class TapeScheduler**

The `TapeScheduler` class is designed to create an optimized schedule for tape production, taking into account various parameters and constraints. Here's a detailed explanation of its components and functionalities:

**Constructor: `\_\_init\_\_(self, schedule\_df, tape\_data\_df, changeover\_df, last\_production, denier\_step\_data, planned\_stop\_data, recipe\_change\_data, regular\_tapes)`**

Purpose: Initializes the `TapeScheduler` class with necessary data.

Parameters:

- `schedule\_df`: DataFrame containing the schedule data.

- `tape\_data\_df`: DataFrame with detailed information about tapes.

- `changeover\_df`: DataFrame containing changeover data for various tape types.

- `last\_production`: List containing the last production data for each machine.

- `denier\_step\_data`: DataFrame with information about denier change steps.

- `planned\_stop\_data`: DataFrame containing data about planned stops.

- `recipe\_change\_data`: DataFrame with details on recipe changes.

- `regular\_tapes`: List of regular tapes.

Functionality: Sets up the class with initial data and prepares the scheduling structure.

**Method: `get\_changeover\_data(self, prev\_tape\_id, tape\_id, machine)`**

Purpose: Determines the required changeover data based on the previous and current tape IDs and the machine.

Parameters:

- `prev\_tape\_id`: The ID of the previous tape.

- `tape\_id`: The ID of the current tape.

- `machine`: The machine on which the tape is to be produced.

Functionality: Calculates the type of changeover needed and retrieves relevant data such as time and resource requirements.

**Method: `filter\_with\_danier(self, tape\_denier, machine\_can\_be\_used)`**

Purpose: Filters machines based on their suitability for a given tape denier.

Parameters:

- `tape\_denier`: The denier of the tape.

- `machine\_can\_be\_used`: A list of machines that can potentially be used.

Functionality: Filters the list of machines based on whether they can handle the specified tape denier.

**Method: `get\_recipe\_changetime(self, machine, tape\_UV, tape\_Colour, tape\_Filler)`**

Purpose: Calculates the time required for a recipe change based on tape properties.

Parameters:

- `machine`: The machine on which the change is to be made.

- `tape\_UV`: UV property of the tape.

- `tape\_Colour`: Colour of the tape.

- `tape\_Filler`: Filler content of the tape.

Functionality: Determines the total time needed for recipe changes based on the difference between the new tape properties and the properties of the tape previously run on the machine.

**Method: `get\_denier\_changetime\_step(self, machine, tapedenier, width)`**

Purpose: Calculates the time required for a denier change based on tape properties.

Parameters:

- `machine`: The machine on which the change is to be made.

- `tapedenier`: The denier of the tape.

- `width`: The width category of the tape.

Functionality: Determines the time needed for denier changes, factoring in the denier change steps and the difference between the new tape denier and the previous one.

**Method: `find\_nearest\_denier(self, prev\_value, values\_list)`**

Purpose: Finds the nearest denier values in sequence from a given starting point.

Parameters:

- `prev\_value`: The starting denier value.

- `values\_list`: A list of denier values to choose from.

Functionality: Sorts the denier values in the order of their closeness to the starting value.

**Method: `get\_width\_changetime(self, prev\_tape\_width, tape\_width)`**

Purpose: Calculates the time required for a change in tape width.

Parameters:

- `prev\_tape\_width`: The width of the previous tape.

- `tape\_width`: The width of the current tape.

Functionality: Determines the time needed for width changes based on the difference between the new width and the previous width.

**Method: `schedule\_tapes(self, last\_production)`**

Purpose: To schedule tapes across different machines in an optimized manner, based on a variety of criteria including color, denier, UV properties, and machine availability.

Parameters:

- `last\_production`: A list containing the last production data for each machine.

Functionality:

- Iterates through each tape plant considering planned stop times and the current color being produced on each machine.

- Filters tapes based on their completion dates, color, and denier, aiming to minimize changeovers and optimize production flow.

- For each tape, calculates changeover times required for width, recipe, and denier changes. It schedules tapes in a way that these changeovers occur in the least disruptive manner.

- Updates the schedule and machine availability after scheduling each tape.

- Handles special cases like transitioning from colored to white tapes and vice versa, and managing denier steps for efficient changeovers.

- Accumulates the schedule in a structured format, detailing every aspect of the production plan, including start and end times for production and changeovers, as well as the nature of the changeovers.

**Key Components:**

1. Machine Initialization: Sets up the initial state of each machine based on the last production data.

2. Planned Stops Handling: Incorporates planned stop data to ensure that the schedule aligns with machine downtime.

3. Color-Based Scheduling: Prioritizes scheduling based on tape color to minimize changeovers due to color changes.

4. Denier Order Optimization: Orders the tapes to be scheduled based on their denier, aiming to minimize drastic changes and optimize transition times.

5. Changeover Calculations: For each tape, calculates the time required for various types of changeovers (width, recipe, denier) and plans the schedule accordingly.

6. Schedule Construction: Builds a comprehensive schedule that includes all relevant details for each tape production cycle on each machine.

The `schedule\_tapes` method effectively manages the complexities of tape production scheduling. It takes into account the nuances of tape manufacturing, such as the impact of changing tape properties on machine setup and production flow. The method's approach to scheduling by color and denier, along with its consideration of planned stops and changeover requirements, helps optimize production efficiency, reduce downtime, and ensure a smooth production process.

Overall Functionality:

The `TapeScheduler` class is a comprehensive system for scheduling tape production. It takes into account various factors like tape properties, machine capabilities, changeover requirements, and previous production data to create an optimized production schedule. This class helps to minimize downtime and maximize efficiency in tape production processes.