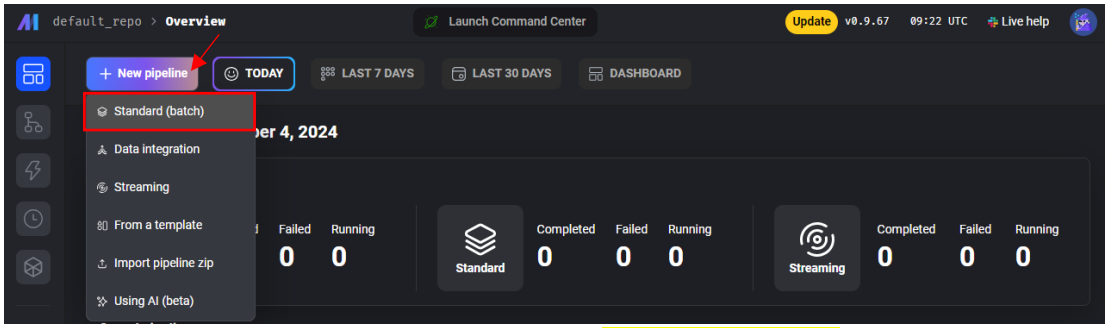
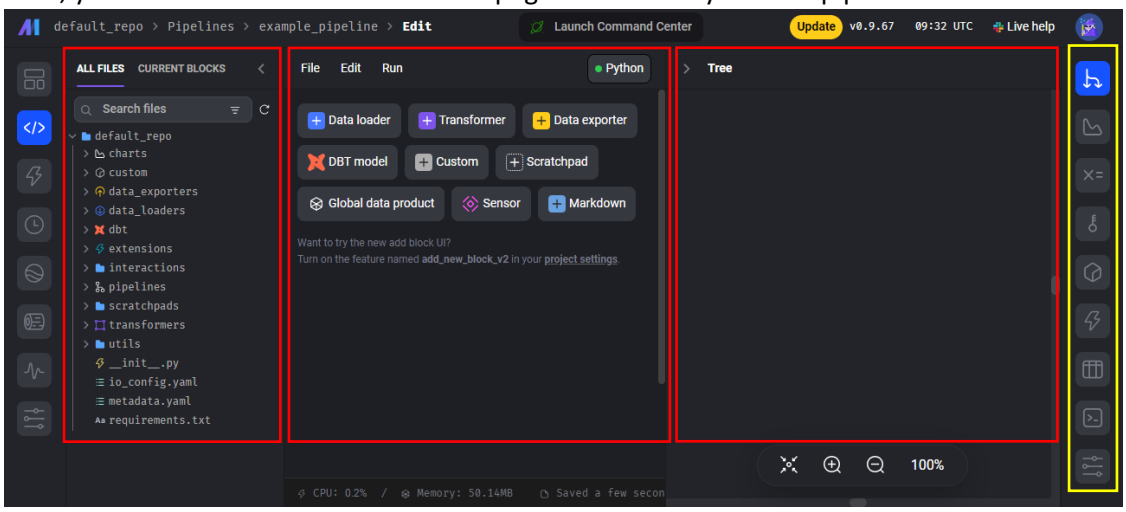
In this tutorial, we'll focus on the "**Transform**" and "**Load**" steps in the ETL process using Mage.ai and Python. Don't worry if you're not familiar with them—this guide is designed to be easy to follow! 😊

1. Click on "**+ New pipeline**" button and select "**Standard (batch)**".

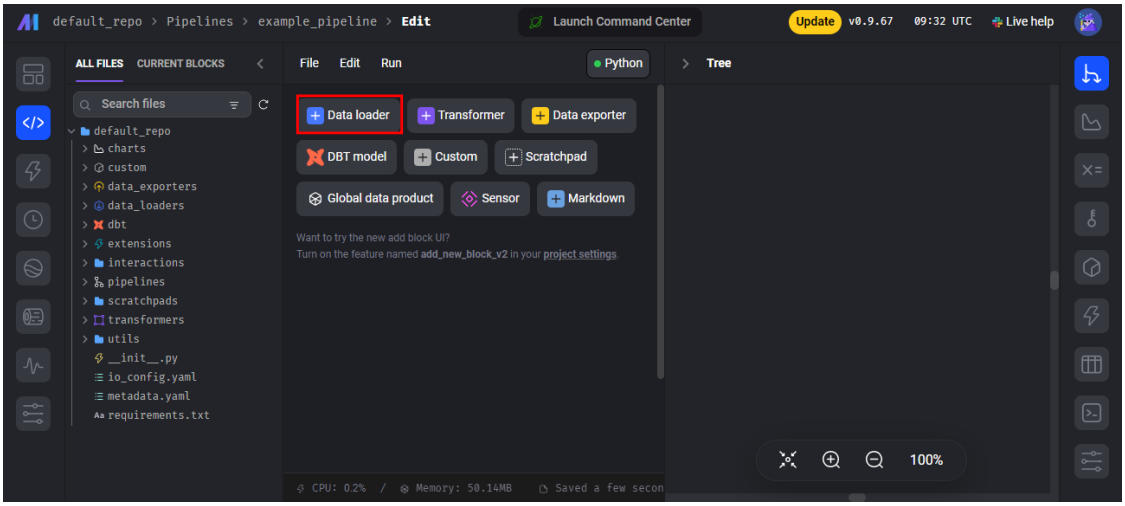


   a. Mage.ai offers different types of pipelines (TODO: Add usages):
      i. Standard (batch),
      ii. Data Integration,
      iii. Streaming,
      iv. From a template,
      v. Import pipeline zip, and
      vi. Using AI.
   b. You may change/add the pipeline's
      i. Name,
      ii. Description, and/or
      iii. Tags to this new pipeline.
2. Next, you will be redirected to the "**Edit**" page for the newly created pipeline.

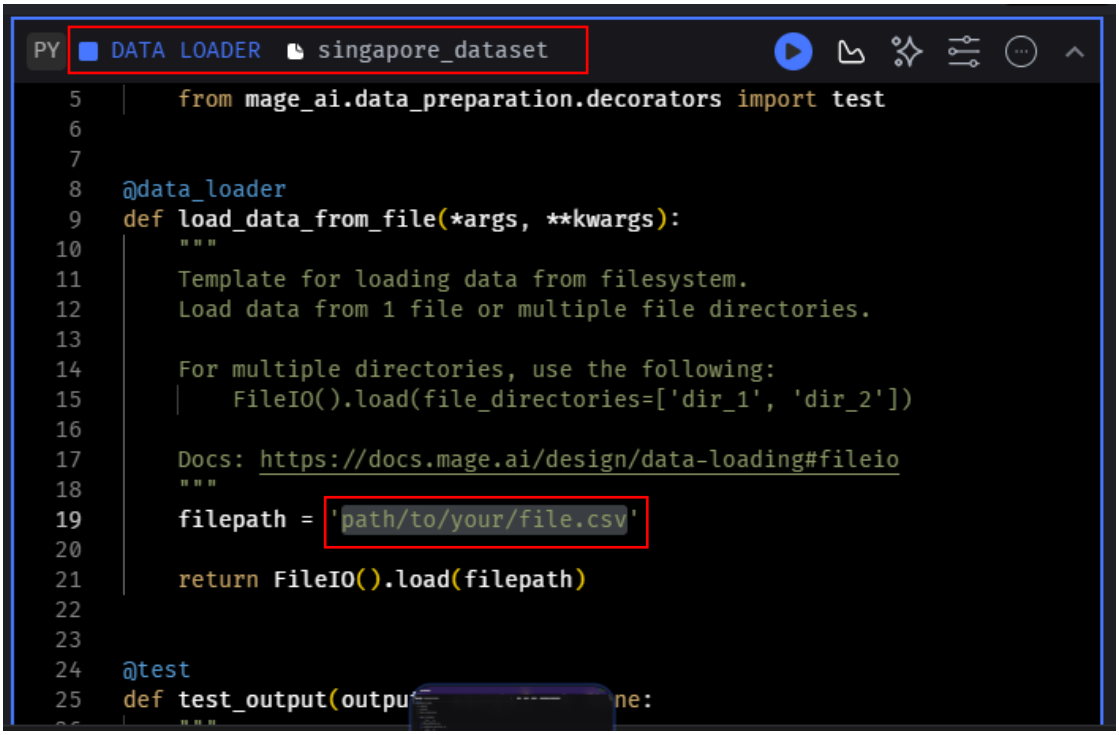


   a. Left pane (Directory pane): Directory for "**/home/src/default_repo**".
   b. Middle pane: Allows you to choose any to add a block to your pipeline and serves as a code editor.
   c. Right pane: To view the items relevant to the icons on the extreme right that is boxed up in yellow.
3. Firstly, we will load the data from the file to Mage.ai:
   Select "**+ Data loader**" > "**Python**" > "**Local file**". Change the name of this block to "**singapore_dataset**".

For more information regarding Mage.ai's documentation, please refer to https://docs.mage.ai/.



a. A code editor will be added to the middle pane.
b. In the code editor, note that you're viewing the code for *Data Loader* that is named "**singapore_dataset**".
c. The function "**load_data_from_file**" will be executed when this block runs. It will read in the file from the specified filepath. Replace "**path/to/your/file.csv**" with "**/home/src/default_repo/datasets/**".



d. Then, upload the "**singapore.csv**" file to "**/home/src/default_repo/datasets/**": Right click on "**default_repo**" on the directory pane > Select "**New folder**" > Enter "**datasets**" > Select "**Create**" > Right click on "**datasets**" on the directory pane > Select "**Upload files**" > Drag and drop or Browse and select "**singapore.csv**" > "**Close**".

4. Secondly, we will **transform** the data based on the plan we ideated previously.
   a. Get the "**year**" from "**epi_week**".
      i. Scroll to the bottom of the middle pane and add a *Transformer*: Select "**Transformer**" block > "**Python**" > "**Generic (no template)**".

ii. The function "**transform**" will be executed when this block runs. The first parameter that this function takes in ("**df**") is the output from the "**singapore_dataset**" *Data Loader* block, which is the dataset itself in Python's pandas' DataFrame type.

iii. To extract "**year**" from "**epi_week**", replace the code in "**transform**" function with the following (the "**df**" in the code refers to the "**data**" variable being passed in as parameter):

```
# Extracts the first 4 characters of each value and store it under the "year" column
df['year'] = df['epi_week'].str.slice(0, 4)
# Removes "epi_week" column from the dataframe
df.drop(columns=['epi_week'], inplace=True)
# Returns this updated dataframe to pass it to the next block
return df
```

b. Get the new number of cases based on year and disease:

i. Add another *Transformer* block by going to the end of the middle pane and select "**Transformer**" > "**Python**" > "**Aggregate**" > "**Aggregate by sum of values**".

ii. Similar to the "**transform**" function mentioned in Section 4.a.ii (TODO: Add link), the first parameter taken in by the function "**execute_transformer_action**" in this newly created *Transformer* is the dataframe returned from the previous *Transformer*'s returned dataframe.

iii. To get the new number of cases, replace the code block in the function "**execute_transformer_action**" with the following code below. This code will return a the original dataframe with a new column called "**no_of_cases**" and the value in it will be based on its disease and year values for that row.

```
# Groups data by disease and year to get the summed no_of_cases value
  get_no_of_cases = build_transformer_action(
    df,
    action_type=ActionType.SUM,
    arguments=['no._of_cases'],  # Enter the columns to compute aggregate over
    axis=Axis.COLUMN,
    options={'groupby_columns': ['disease', 'year']},  # Enter columns to group by
    outputs=[
      # The number of outputs below must match the number of arguments
      {'uuid': 'no_of_cases', 'column_type': 'number'},
    ],
  )
  return BaseAction(get_no_of_cases).execute(df)
```

c. Remove old number of cases

i. Based on Section 4.b.iii (TODO: Add link), a new column for the number of cases was created. Hence, we will remove the old column for the number of cases which is now obsolete as we are more interested in the number of cases related to certain disease in a certain year.

ii. To remove a column, we can add this *Transformer* from the bottom of the middle pane: Select "**Transformer**" > "**Python**" > "**Column Removal**" > "**Remove Column(s)**".

iii. Then, in the function "**execute_transformer_action**", specify "**no._of_cases**" in the "**arguments**" parameter being passed to the function "**build_transformer_action**" and the code in your function should look like this:

```
# Removes the initial column for no._of_cases
  remove_old_no_of_cases = build_transformer_action(
    df,
    action_type=ActionType.REMOVE,
    arguments=['no._of_cases'],  # Specify the column(s) you want to remove
    axis=Axis.COLUMN,
  )
  return BaseAction(remove_old_no_of_cases).execute(df)
```

d. Drop duplicate rows
   i. Based on Section 4.b.iii (TODO: Add link), there will be many duplicated rows in the dataframe (i.e. same year, disease, and number of cases).
   ii. To drop duplicates, we can add this *Transformer* from the bottom of the middle pane: Select "**Transformer**" > "**Python**" > "**Row Actions**" > "**Drop duplicates**".
   iii. For this transformer step, we don't need to modify the code as we want to drop rows with the same values in all columns and it doesn't matter if we keep the first or last duplicate row since they are the same.

e. Add "**country**" column
   i. Now that our data is ready, we want to add a new column called "**country**" and the values in this column will all be "**singapore**". This allows us to add other countries' similar health data in the future.
   ii. To do so, add a generic *Transformer* block like in Section 4.a.i (TODO: Add link) and replaced the function "**transform**" code with the following (the "**df**" in the code refers to the "**data**" variable being passed in as parameter):

```
# Adds new column with 'singapore' as value
  df['country'] = 'singapore'
  return df
```

5. Lastly, let's **load** this data into your database.
   a. Let's add a *Data Exporter* block at the end of the middle pane: Select "**Data Exporter**" > "**Python**" > "**PostgreSQL**".
   b. In the function "**export_data_to_postgres**", update the values in the variables "**schema_name**" to "**public**" and "**table_name**" to "**healthcare**".
   c. To let Mage.ai know how to connect to your database, navigate to "**./fyp/utils/io_config.yaml**" on the directory pane. Search for "**# PostgresSQL**" in Line 82. From Lines 83 to 89, update the following with the database that we have allocated for you: (TODO: Update this when we are able to spin up for each user a postgresql container. HOST and PORT is based on the <name>: in compose file and

port number running in container and not local machine since they are running in the same compose file)

```
# PostgresSQL
POSTGRES_CONNECT_TIMEOUT: 10
POSTGRES_DBNAME: db
POSTGRES_SCHEMA: public # Optional
POSTGRES_USER: postgres
POSTGRES_PASSWORD: password
POSTGRES_HOST: db
POSTGRES_PORT: 5432
```

6. Now that our pipeline is ready, we can trigger it to execute all the blocks such that the raw dataset will be transformed and then loaded to our database.
    a. On the navigation pane, to "**Triggers**" > "**Run@once**". Click into the newly created trigger to see if the pipeline block runs are successful or not. To look into more details, you can click on the number under "**Block runs**" or icon under "**Logs**". To rerun, simply click on "**Run@once**" again.
7. When you're done with this tutorial, click on the next button at the bottom right of this page.