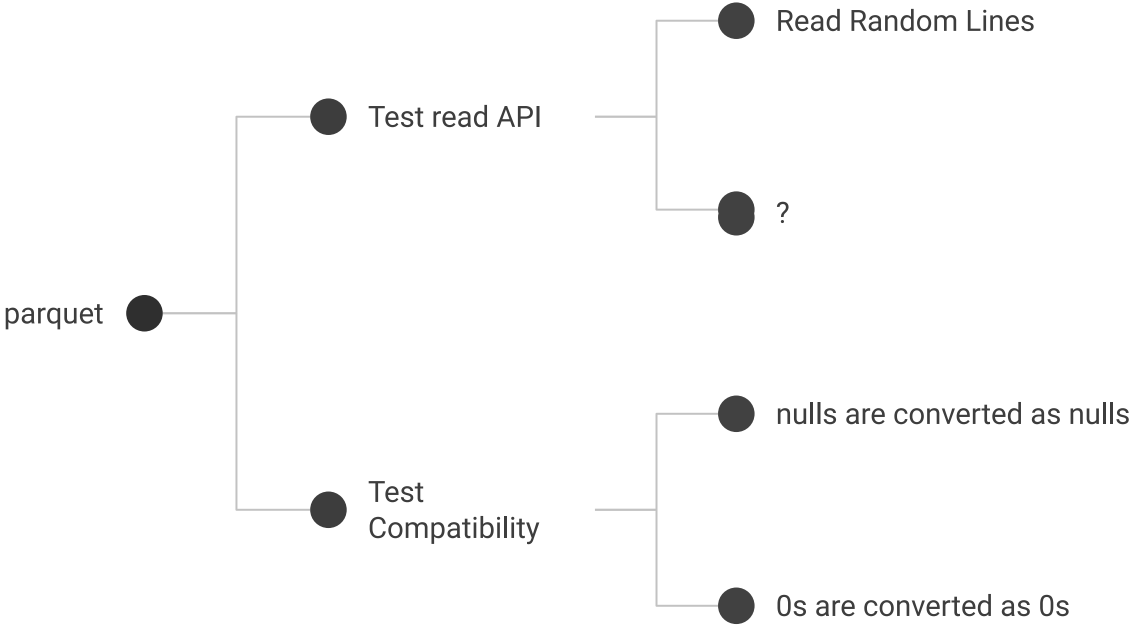
Data
/**P**rocessing
/**C**leaning
/**I**mputation
Pipeline

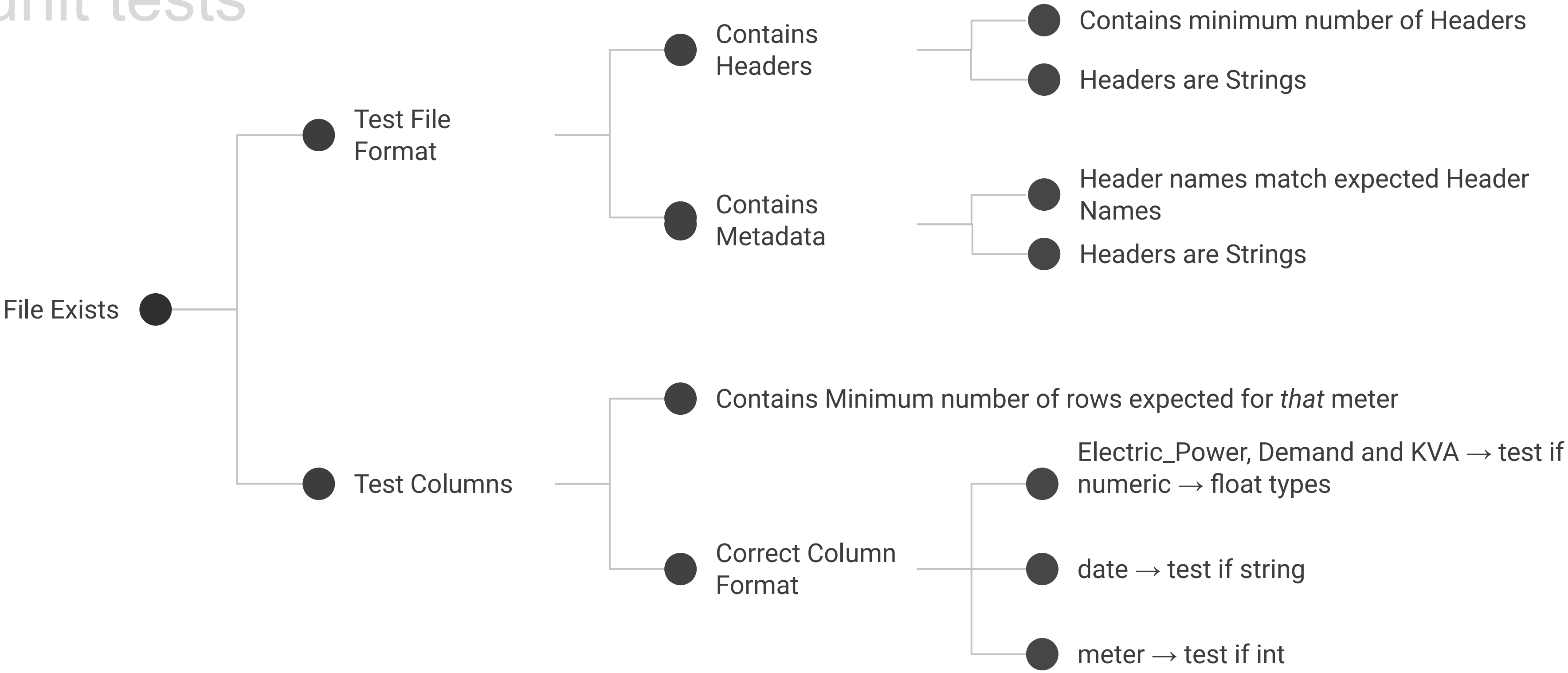
on Real-Time data

Sana Wajid
12/5/19
Nair and Associates LLC

unit tests

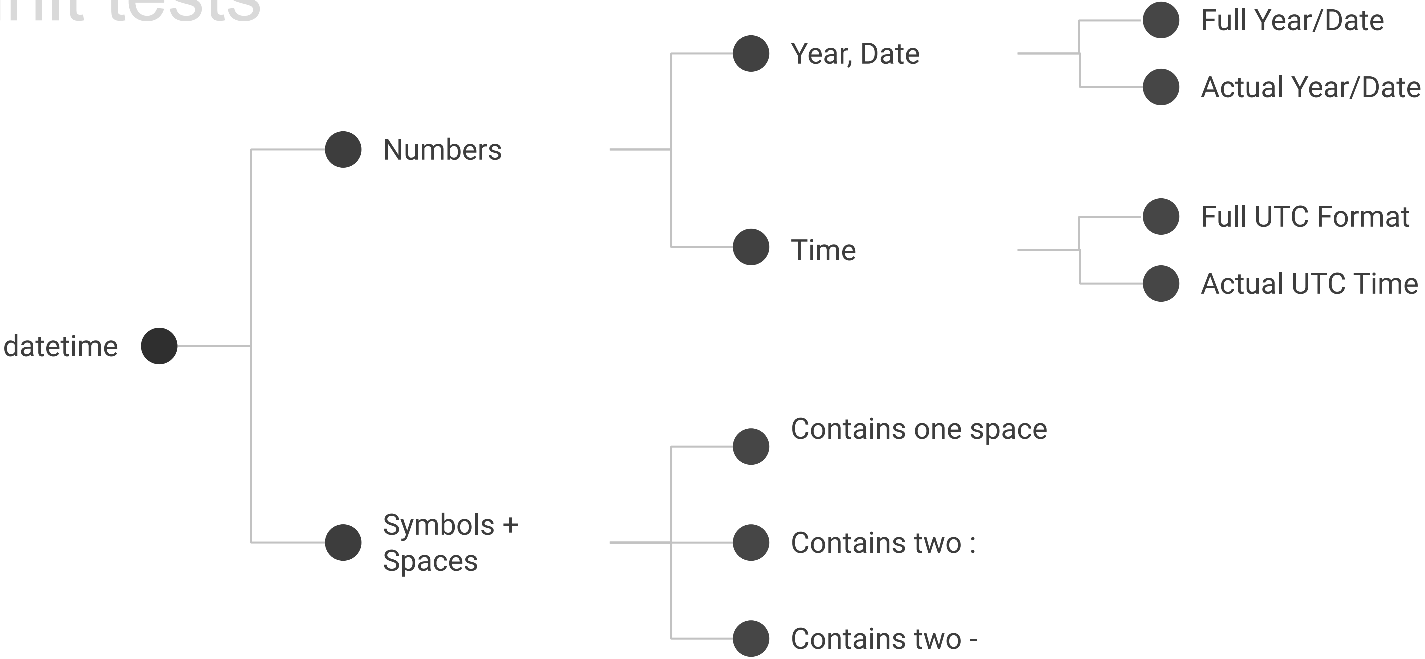

parquet
Test read API
Read Random Lines
?
Test Compatibility
nulls are converted as nulls
0s are converted as 0s

unit tests



- File Exists
  - Test File Format
    - Contains Headers
      - Contains minimum number of Headers
      - Headers are Strings
    - Contains Metadata
      - Header names match expected Header Names
      - Headers are Strings
  - Test Columns
    - Contains Minimum number of rows expected for *that* meter
    - Correct Column Format
      - Electric_Power, Demand and KVA → test if numeric → float types
      - date → test if string
      - meter → test if int

unit tests



- datetime
  - Numbers
    - Year, Date
      - Full Year/Date
      - Actual Year/Date
    - Time
      - Full UTC Format
      - Actual UTC Time
  - Symbols + Spaces
    - Contains one space
    - Contains two :
    - Contains two -

0P. Step

P = Processing
C = Cleaning
I = Imputation

Assumptions:

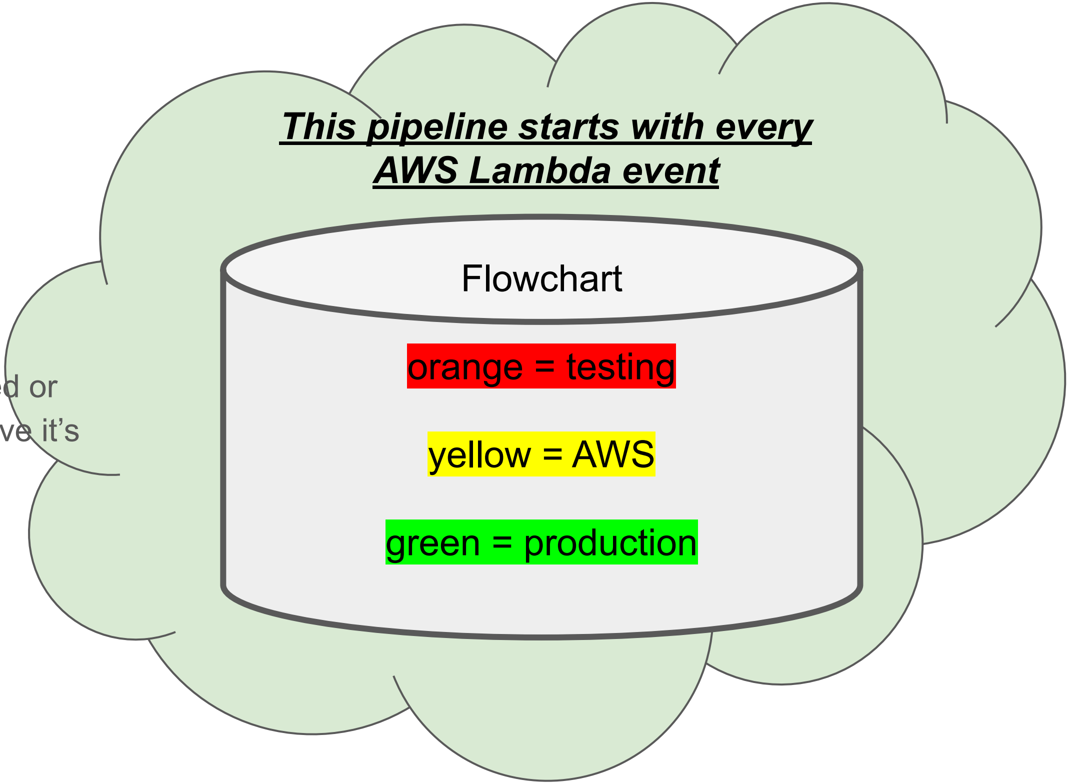
- For testing using csv files
- These are naive assumptions that can be added or removed. Please remember, a pipeline can move it's starting point upstream or downstream.

Checks:

1. Listed in order of least to most processing time
2. Numbers will refer to function numbers in code comment or headers in Jupyter notebook
   a. e.g. 1P-2: File contains minimum number of headers

Libraries:

Libraries used in {pandas, scikit-learn} ∈ Python



This pipeline starts with every AWS Lambda event

Flowchart

🟥 orange = testing

🟨 yellow = AWS

🟩 green = production

1P. Input

Assumptions:

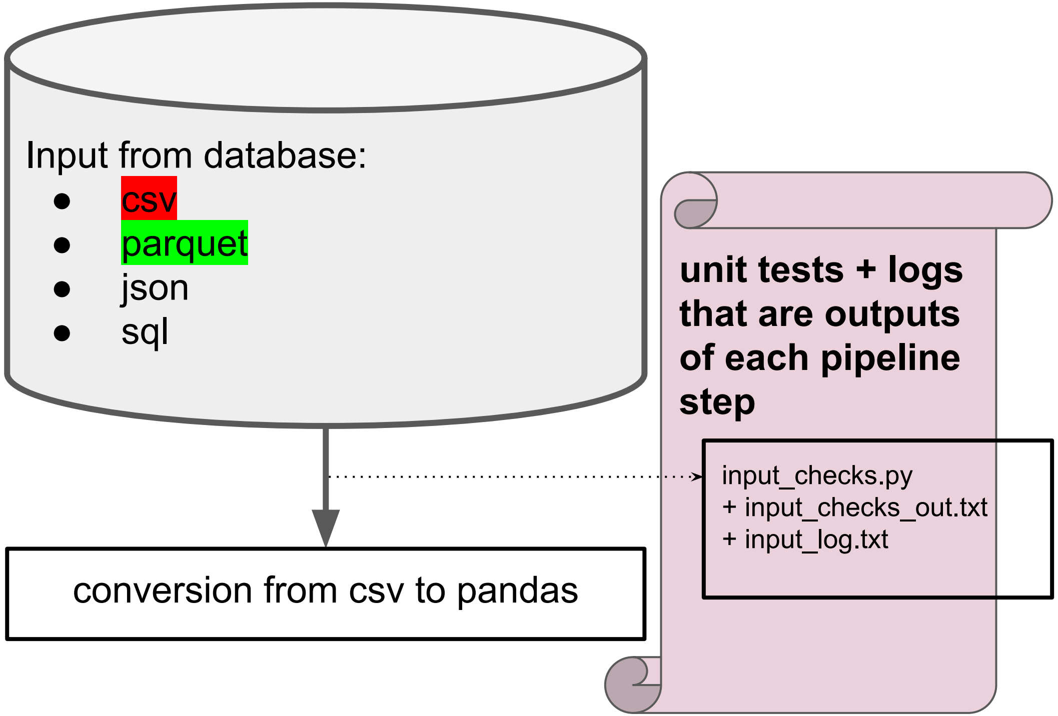
- There is a connection to the database
- Data is “raw data”
- Column names don’t contain spaces

Functions:

1. File exists
2. File contains minimum number of headers
3. File contains minimum number of rows
4. File contains correct headers

Libraries:

sed/awk or unix or base python



Input from database:

- csv
- parquet
- json
- sql

conversion from csv to pandas

unit tests + logs that are outputs of each pipeline step

input_checks.py
+ input_checks_out.txt
+ input_log.txt

2P. Raw data conversion to pandas

Assumptions:

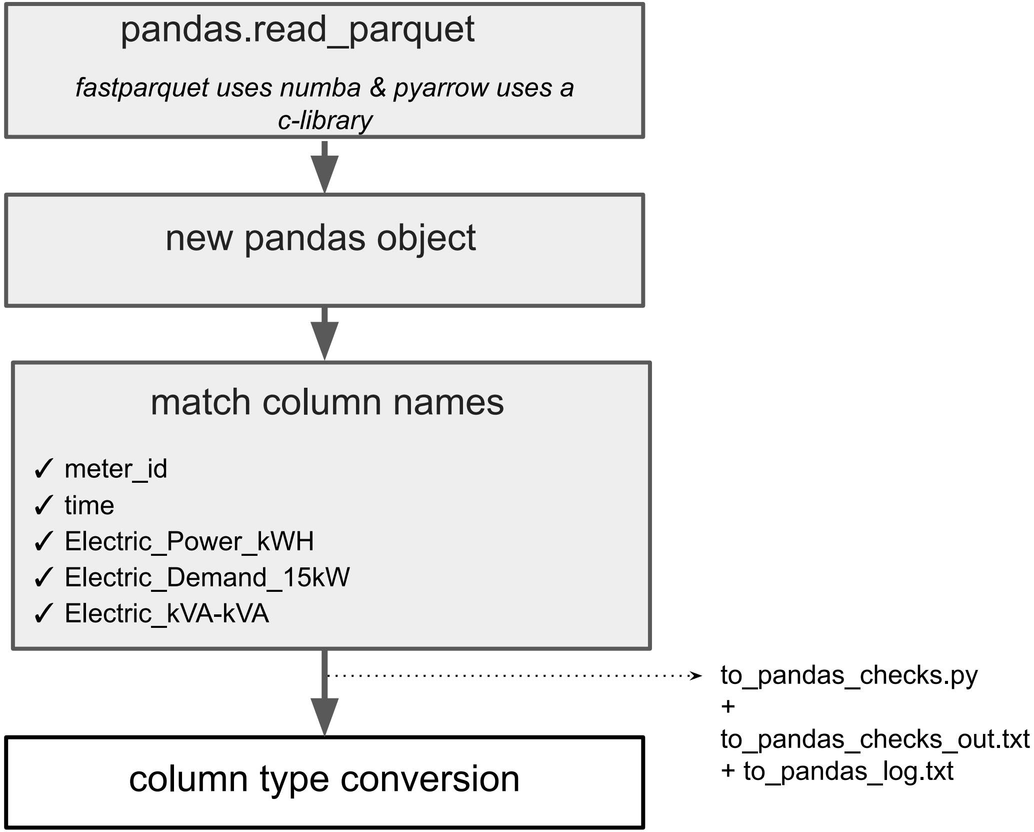
- File contains minimum number of headers
- File contains minimum number of rows
- File contains correct headers
- Column names don’t contain spaces

Functions:

1. Conversion to pandas

Libraries:

`read_parquet` ∈ pandas ∈ Python



pandas.read_parquet

fastparquet uses numba & pyarrow uses a c-library

new pandas object

match column names

✓ meter_id
✓ time
✓ Electric_Power_kWH
✓ Electric_Demand_15kW
✓ Electric_kVA-kVA

column type conversion

to_pandas_checks.py
+
to_pandas_checks_out.txt
+ to_pandas_log.txt

3C. datetime column type conversion

Assumptions:

- time column is formatted as:

YEAR-MONTH-DAY

space

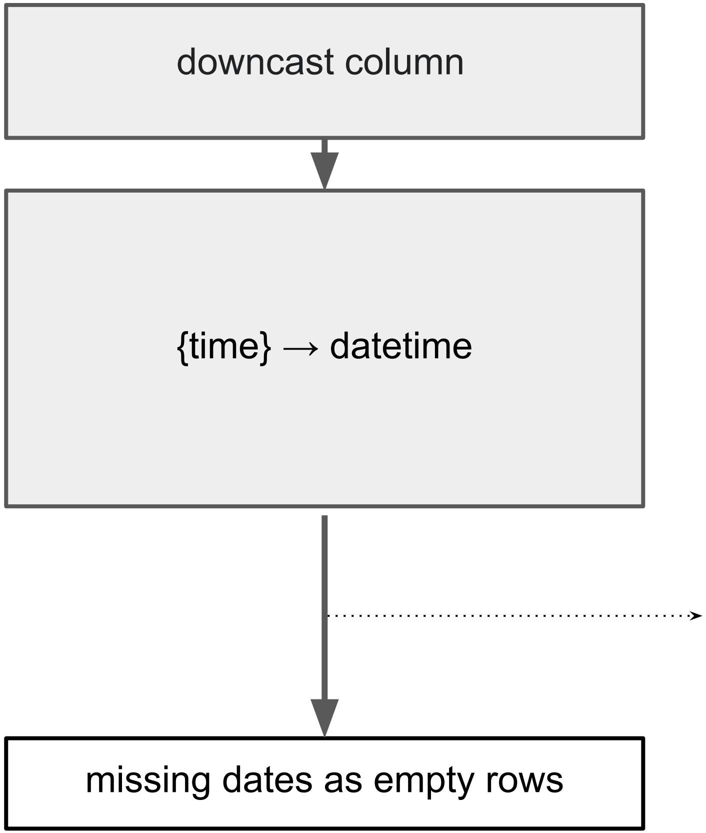
HOUR:MINUTE:SECOND

Functions:

```
1.   column_to_datetime
```

Libraries:

datetime ∈ Python



downcast column

{time} → datetime

missing dates as empty rows

to_col_int_float_checks.py
+ to_col_int_float_out.txt
+ to_col_int_float_log.txt

3C. Fill missing datetime *rows* as empty rows

Assumptions:

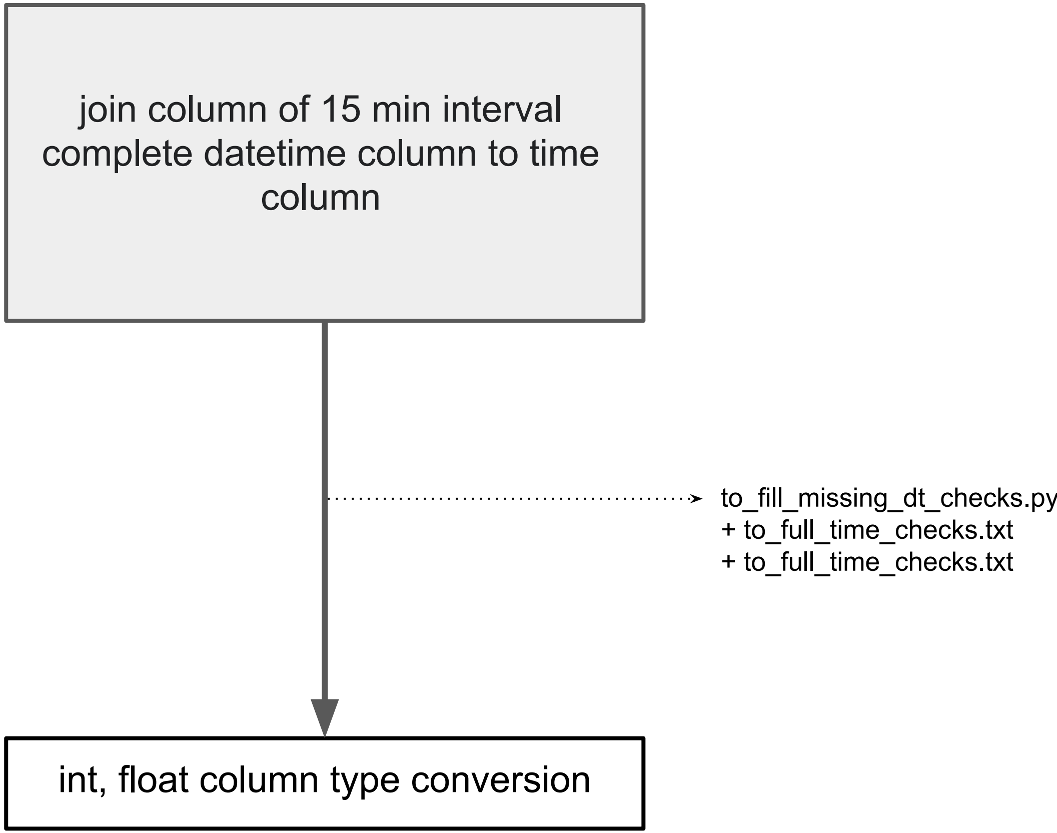
- time column is a series of datetime values

Functions:

1. `fill_missing_dates`

Libraries:

datetime ∈ Python



join column of 15 min interval complete datetime column to time column

to_fill_missing_dt_checks.py
+ to_full_time_checks.txt
+ to_full_time_checks.txt

int, float column type conversion

4C. int and float column type conversion

Assumptions:

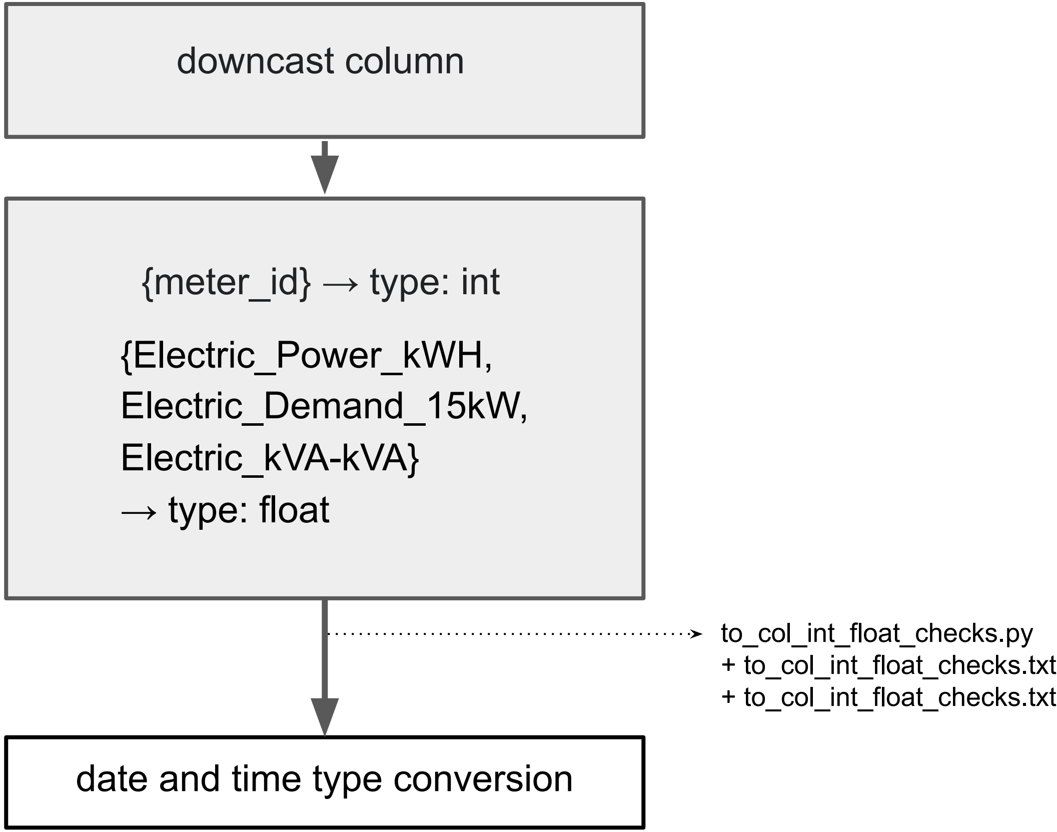
- Columns match minimum number of columns for this meter

Functions:

1. `column_to_int`
2. `column_to_float`

Libraries:

pandas ∈ Python



downcast column

{meter_id} → type: int

{Electric_Power_kWH, Electric_Demand_15kW, Electric_kVA-kVA} → type: float

to_col_int_float_checks.py
+ to_col_int_float_checks.txt
+ to_col_int_float_checks.txt

date and time type conversion

5I. Impute strategy: easy

Assumptions:

- time column is complete:

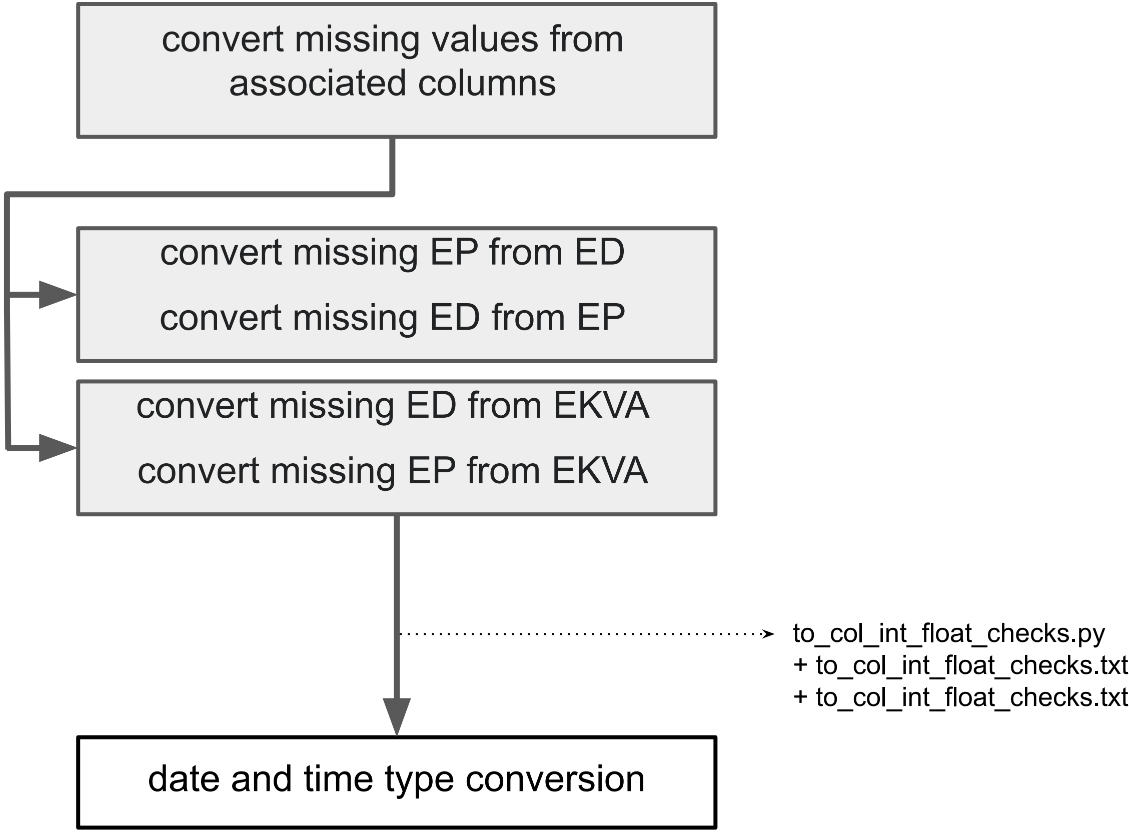
  every $t_0 : t_{end}$ has a row

Functions:

1. `convert_x_from_EP`
2. `convert_x_from_EP`
3. `convert_x_from_EKVA`

Libraries:

pandas ∈ Python



convert missing values from associated columns

convert missing EP from ED

convert missing ED from EP

convert missing ED from EKVA

convert missing EP from EKVA

to_col_int_float_checks.py
+ to_col_int_float_checks.txt
+ to_col_int_float_checks.txt

date and time type conversion

6aI. Impute strategy, medium: partition

Assumptions:

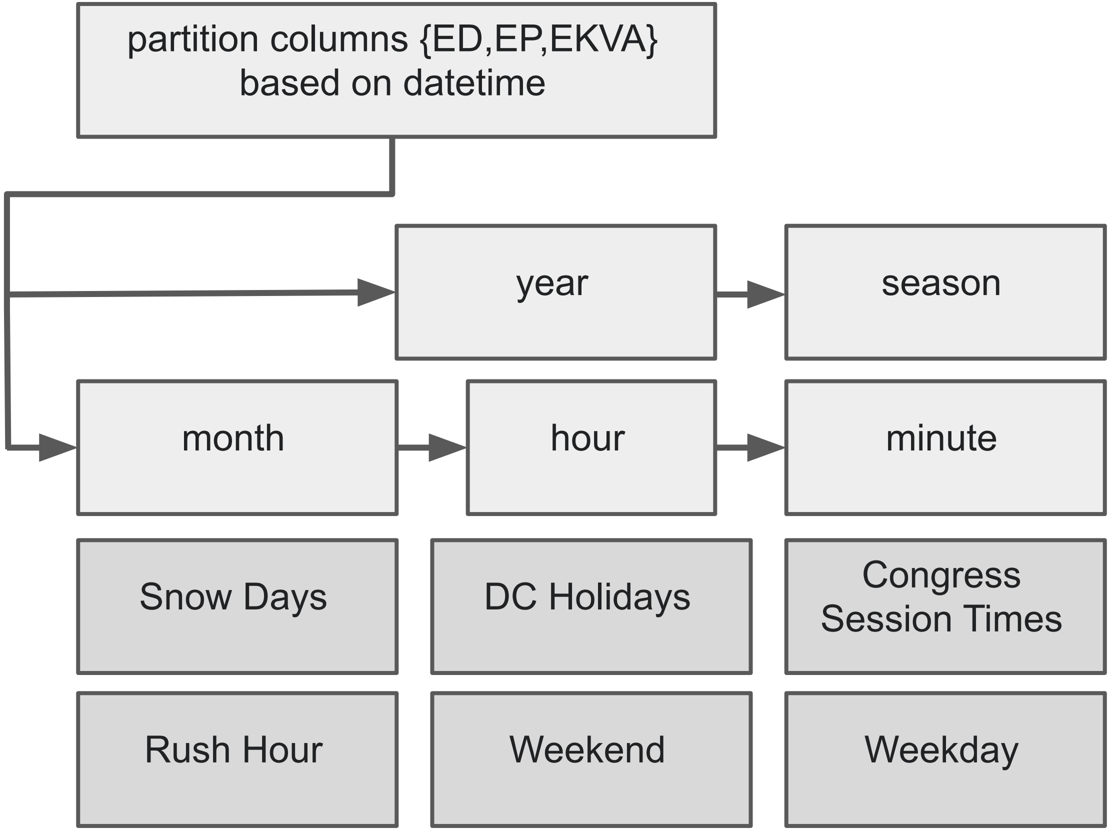
- time column is complete:

  $t_0 : t_{end}$ has a row

Functions:

accessor (get) functions will return dataframe

1. `get_year`(pandas type dataframe)
2. `get_season`(pandas type dataframe)
3. `get_month`(pandas type dataframe)
4. `get_hour`(pandas type dataframe)
5. `get_minute`(pandas type dataframe)
6. `get_Rush_Hour`(pandas type dataframe)
7. `get_Weekend`(pandas type dataframe)
8. `get_Weekday`(pandas type dataframe)



6aI. Impute strategy, medium: partition

Assumptions:

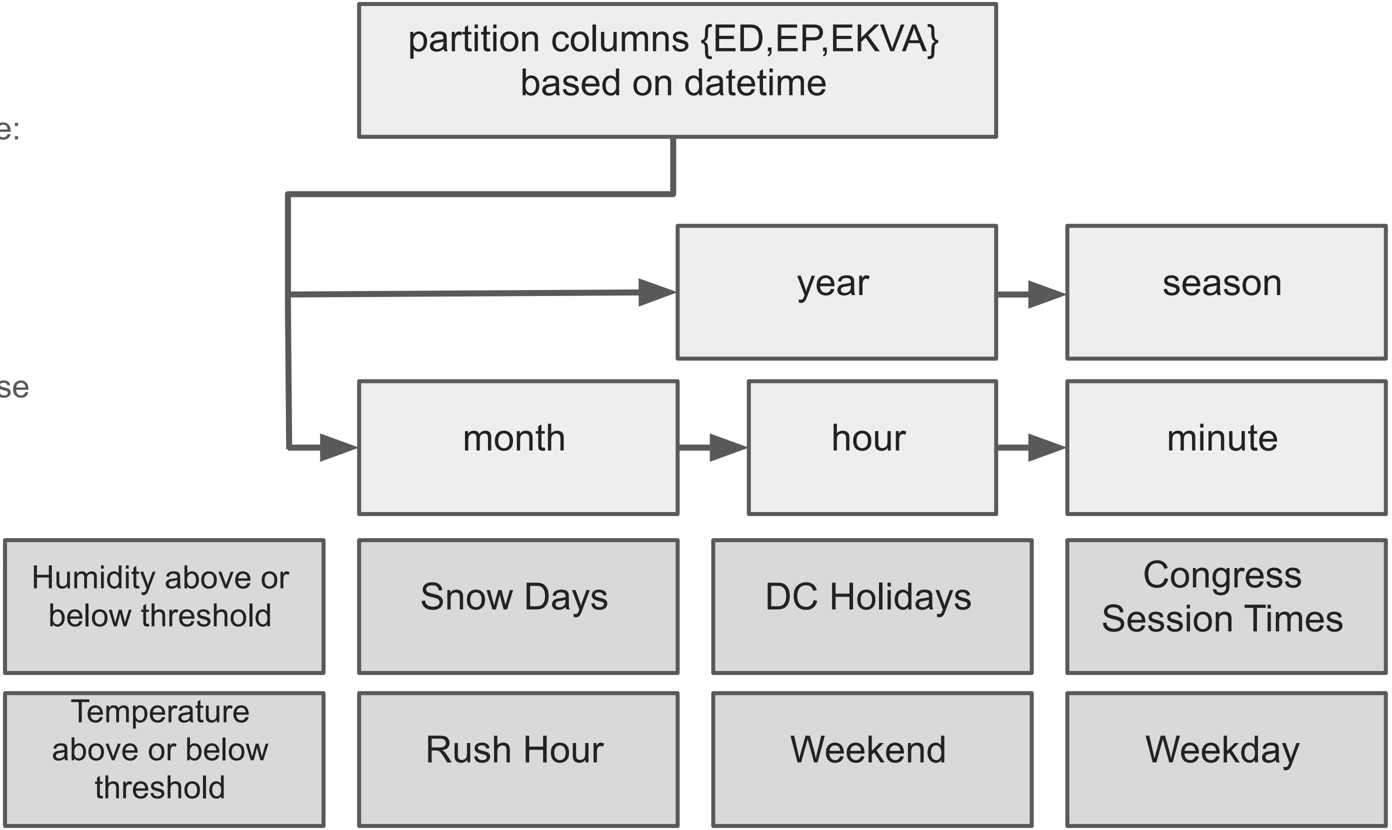
- time column is complete:

  t_0 : t_{end} has a row

Functions:

boolean will return True or False

1. `is_Snow_Day`
2. `is_DC_Holiday`
3. `is_Congress_in_Sess`
4. `is_Rush_Hour`
5. `is_Weekend`
6. `is_Weekday`
7. `is_high_temp`
8. `is_high_humidity`



partition columns {ED,EP,EKVA} based on datetime

year

season

month

hour

minute

Humidity above or below threshold

Snow Days

DC Holidays

Congress Session Times

Temperature above or below threshold

Rush Hour

Weekend

Weekday

6aI. Impute strategy, medium: identify null clusters

Assumptions:

- time column is complete:

  every $t_0 : t_{end}$ has a row

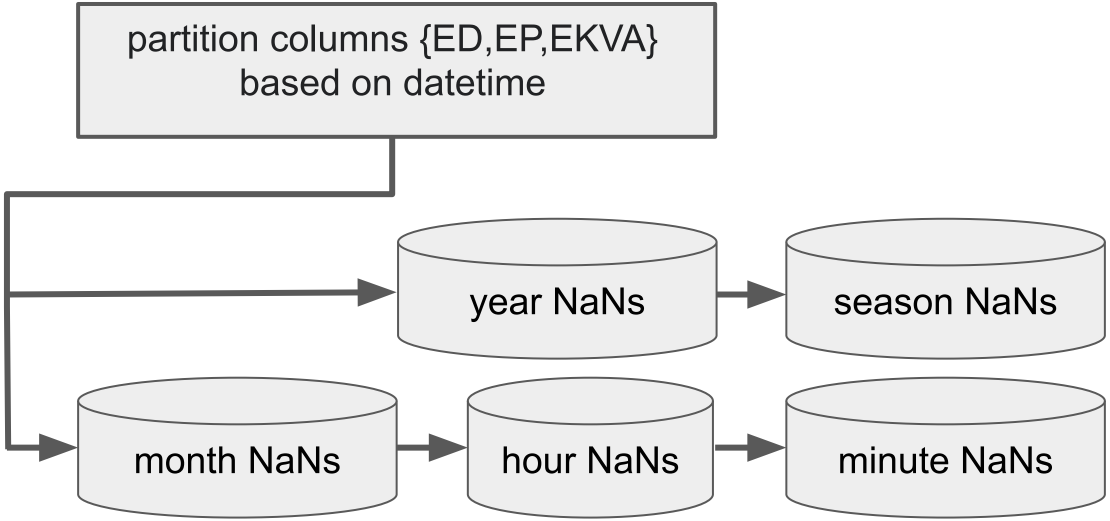
Functions:

accessor (get) functions will return dataframe

1. `get_null`(pandas type dataframe)
   a. output row of nulls given dataframe

filter bin size

1. `get_null_bin_size`



6bI. Impute strategy, medium: identify null clusters

Assumptions:

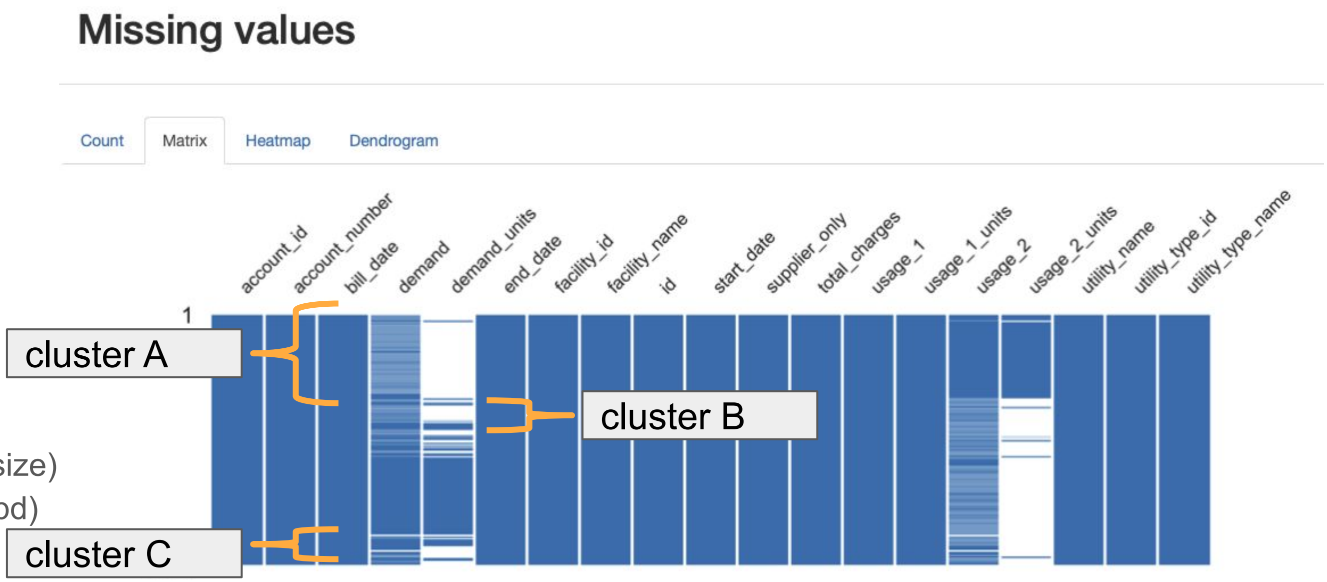
- time column is complete:

  every t_0 : t_{end} has a row

Functions:

boolean

3. `is_null_cluster`(pd, bin_size)
4. `flanking_null_clusters`(pd)



6cl. Impute strategy, medium: data integration

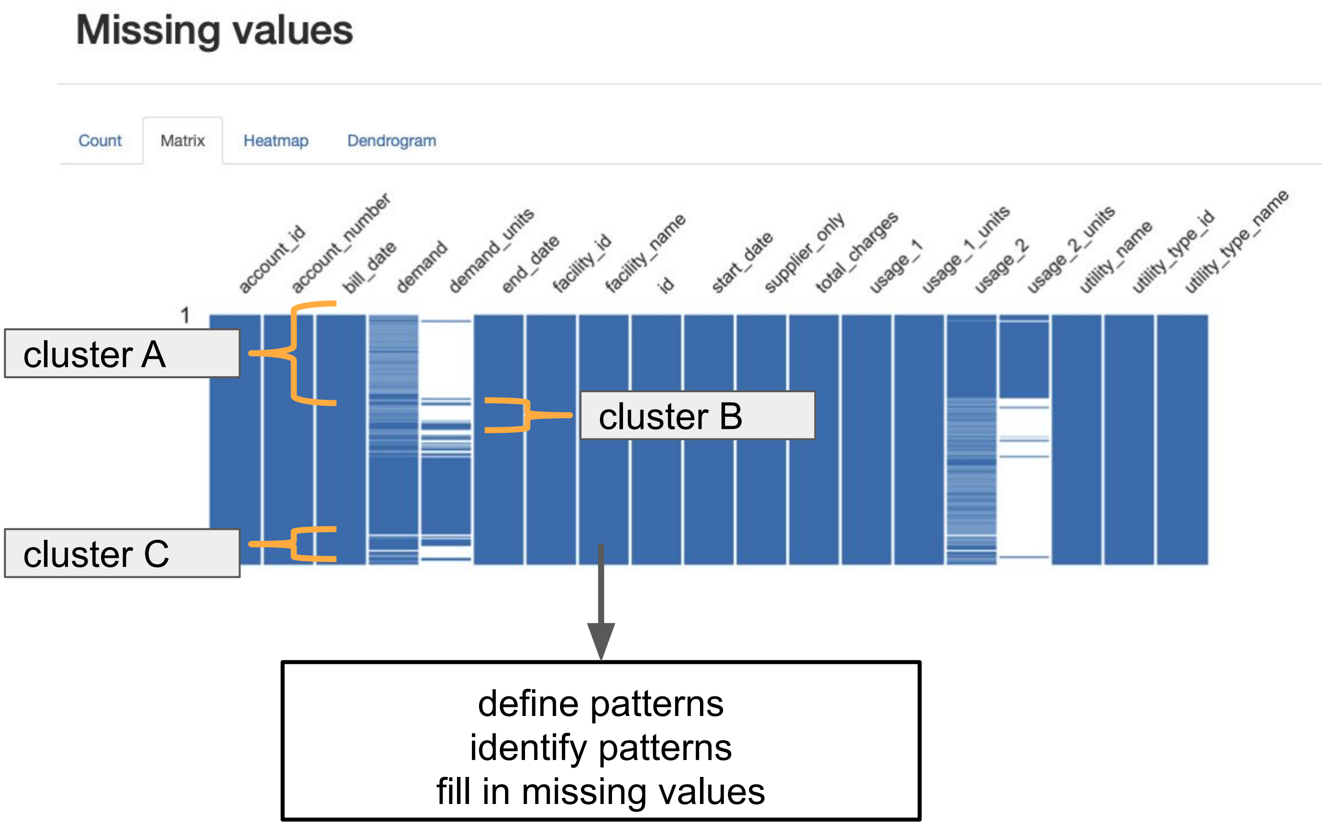
Assumptions:

- weather data is loaded

Functions:

merge functions

5. `merge_weather`
6. `merge_previous_year`
7. `merge_previous_season`
8. `merge_previous_month`



Missing values

define patterns
identify patterns
fill in missing values

7aI. Impute strategy, statistics: scikit-learn

Assumptions:

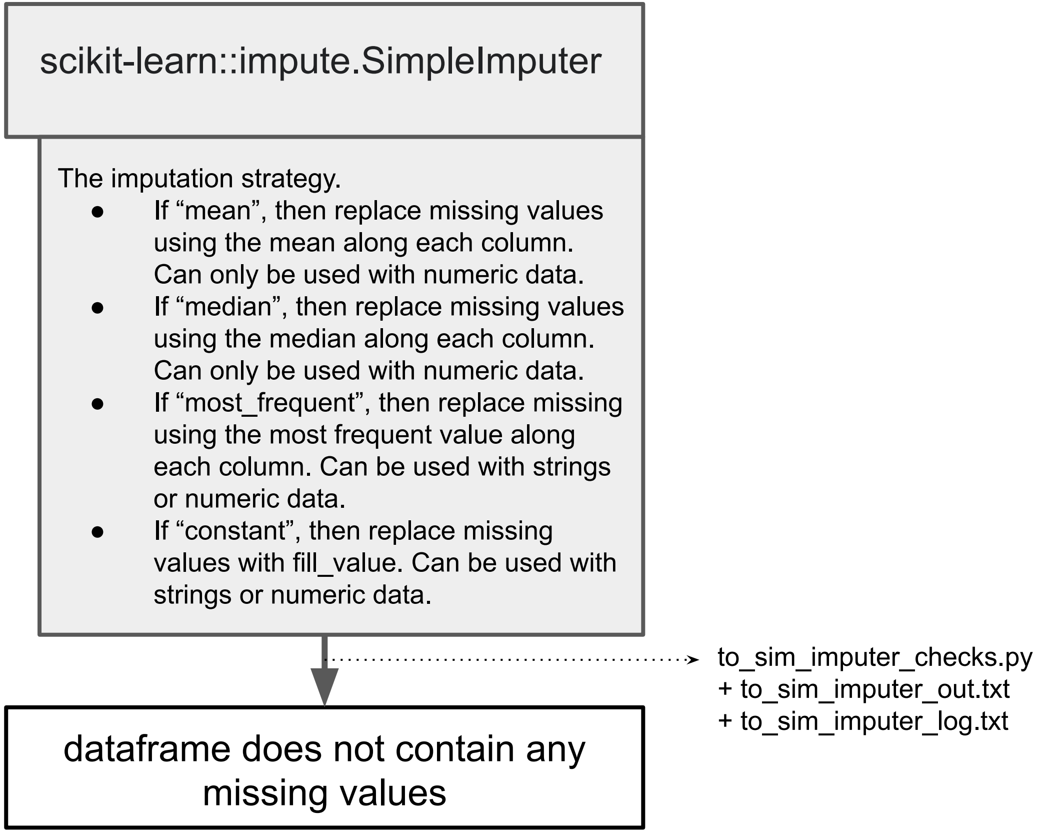
- fill value is defined per building and per meter

Functions:

1. `impute_cluster_by_mean`
2. `impute_cluster_by_median`
3. `impute_cluster_by_most_freq`
4. `impute_cluster_by_constant`

Libraries:

scikit-learn ∈ Python



scikit-learn::impute.SimpleImputer

The imputation strategy.

- If “mean”, then replace missing values using the mean along each column. Can only be used with numeric data.
- If “median”, then replace missing values using the median along each column. Can only be used with numeric data.
- If “most_frequent”, then replace missing using the most frequent value along each column. Can be used with strings or numeric data.
- If “constant”, then replace missing values with fill_value. Can be used with strings or numeric data.

to_sim_imputer_checks.py
+ to_sim_imputer_out.txt
+ to_sim_imputer_log.txt

dataframe does not contain any missing values etc, more will be added

Google’s Python Style Guide

http://google.github.io/styleguide/pyguide.html

References