TakharKuljit_pset5

March 10, 2025

```
<h2 style ="text-align:center; padding-top:5px;"> CS 101 - Foundation of Data Science and Engi
<p style="text-align:center;padding:5px; fontt-size:14px"><b> PSET-5 - Exploratory Data Analys
```

1 This is an individual assignment. No collaboration is allowed!

1.0.1 Note:

You are allowed to use all Python built-in functions and other Import features covered in class. Ensure your code is organized, well-commented, and follows the best practices we discussed. Remember, the key is not just to write a working program but to produce a solution that follows SPECS, is easy to debug, and is easy to maintain.

1.0.2 Assignment Question List:

Question 1 : Data Extraction and cleaning (1 pts)

Question 2 : Column Cleaning Function (15 pts)

Question 3 : Data Cleaning (9 pts)

Question 4 : Data Aggregation (30 pts)

Question 5 : Merging Data (10 pts)

Question 6 : Filtering Data (30 pts)

Coding Style : (5 pts)

1.1 Objective:

You are tasked with performing detailed exploratory data analysis on various system datasets: CPU, Disk, and Memory. Utilize Python functions to streamline data extraction, cleaning, and aggregation.

1.1.1 Datasets:

1. **CPU Dataset:** `cpu.csv`
   - **Columns:** 'Image', 'PID', 'Description', 'Status', 'Threads', 'CPU', 'Average CPU'
2. **Disk Dataset:** `disk.csv`
   - **Columns:** 'Image', 'PID', 'Model', 'Read Byte Per Second', 'Write Byte Per Second', 'Delay', 'Total Byte Per Second'
3. **Memory Dataset:** `memory.csv`

- **Columns:** 'Image', 'PID', 'Hard Faults/sec', 'Commit KB', 'Working Set KB', 'Shareable KB', 'Private KB'

2 Your Tasks

2.0.1 1. Data Extraction (1 pts):

a. **Load `cpu.csv` into a Pandas DataFrame named `df_cpu`. Extract only these columns:** 'Image', 'PID', 'Description', 'Status', 'Threads', 'Average CPU'

b. **Load `disk.csv` into a DataFrame named `df_disk`. Extract only these columns:** 'Image', 'PID', 'Total Byte Per Second'

c. **Load `memory.csv` into a DataFrame named `df_memory`. Extract only these columns:** 'Image', 'PID', 'Working Set KB'

d. **For each of the above data frame show the shape, and info(example : df_cpu.info())**

[40]:
```
import pandas as pd

# Load the CSV file into a Pandas DataFrame
file_path = "cpu.csv"
df_cpu = pd.read_csv(file_path)

# Extract the specified columns (Image, PID, Total Byte
df_cpu = df_cpu[['Image', 'PID', 'Description', 'Status', 'Threads', 'Average␣
↪CPU']]
```

[42]:
```
# Load disk.csv into a DataFrame and extract specified columns (Image, PID,␣
↪Total Byte Per Second)
df_disk = pd.read_csv("disk.csv")
df_disk = df_disk[['Image', 'PID', 'Total Byte Per Second']]
```

[44]:
```
# Load memory.csv into a DataFrame and extract specified columns (Image, PID,␣
↪Working Set KB)
df_memory = pd.read_csv("memory.csv")
df_memory = df_memory[['Image', 'PID', 'Working Set KB']]
```

[46]:
```
# Display shape and info for each DataFrame
print("CPU DataFrame:")
print(df_cpu.shape)
df_cpu.info()
print("\n")

print("Disk DataFrame:")
print(df_disk.shape)
df_disk.info()
print("\n")
```

```
print("Memory DataFrame:")
print(df_memory.shape)
df_memory.info()
```

```
CPU DataFrame:
(306, 6)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 306 entries, 0 to 305
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Image        306 non-null    object
 1   PID          306 non-null    object
 2   Description  290 non-null    object
 3   Status       306 non-null    object
 4   Threads      306 non-null    object
 5   Average CPU  306 non-null    float64
dtypes: float64(1), object(5)
memory usage: 14.5+ KB


Disk DataFrame:
(14, 3)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 3 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Image                  14 non-null     object
 1   PID                    14 non-null     int64
 2   Total Byte Per Second  14 non-null     object
dtypes: int64(1), object(2)
memory usage: 468.0+ bytes


Memory DataFrame:
(306, 3)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 306 entries, 0 to 305
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Image           306 non-null    object
 1   PID             306 non-null    int64
 2   Working Set KB  306 non-null    object
dtypes: int64(1), object(2)
memory usage: 7.3+ KB
```

2.0.2 2. Column Cleaning Function (15 pts):

a. **Write a Python function, `CleanColumnHeading(dfx)`, to clean the column headers of any Pandas dataframe. The function should be dynamic enough to be able to process any datasets. The function should:**

i. Convert all column names to lowercase.

ii. Replace spaces in column names with underscores _.

iii. Apply the `CleanColumnHeading` function to `df_cpu`, `df_disk`, and `df_memory`.

iv. Show examples of the changes.

Example:

Function header: "'python def CleanColumnHeading(dfx): # Your code to convert all column names to lowercase # Your code to change all spaces in column names to underscores _ return dfx

[49]:
```
# Function to clean column names
def CleanColumnHeading(dfx):
    """
    Cleans the column names:
    - Converts all column names to lowercase.
    - Replaces spaces with underscores.
    """
    dfx.columns = dfx.columns.str.lower().str.replace(' ', '_')
    return dfx
```

[51]:
```
# Load datasets
df_cpu = pd.read_csv("cpu.csv")
df_disk = pd.read_csv("disk.csv")
df_memory = pd.read_csv("memory.csv")
```

[53]:
```
# Apply the function to each data frame
df_cpu = CleanColumnHeading(df_cpu)
df_disk = CleanColumnHeading(df_disk)
df_memory = CleanColumnHeading(df_memory)
```

[55]:
```
# Examples of changes
print("CPU DataFrame columns:", df_cpu.columns)
print("Disk DataFrame columns:", df_disk.columns)
print("Memory DataFrame columns:", df_memory.columns)
```

```
CPU DataFrame columns: Index(['image', 'pid', 'description', 'status',
'threads', 'cpu',
       'average_cpu'],
      dtype='object')
Disk DataFrame columns: Index(['image', 'pid', 'model', 'read_byte_per_second',
       'write_byte_per_second', 'delay', 'total_byte_per_second'],
      dtype='object')
Memory DataFrame columns: Index(['image', 'pid', 'hard_faults/sec', 'commit_kb',
```

```
'working_set_kb',
       'shareable_kb', 'private_kb'],
      dtype='object')
```

2.0.3 3. Data cleaning (9 pts)

Examine the columns 'threads', 'total_byte_per_second', 'working_set_kb' from the dataframes df_cpu, df_disk, and df_memory. We are going to work with these columns Question 4-6. Ensure that they have the correct data type, fix the values if required so. If there are invalid values drop them. At the end of this step you should not have any invalid values, and the correct data type set for the 3 columns. You are not required to do data cleaning on other columns. If you choose to do so, please ensure that no rows are dropped while cleaning these columns

Hint: values such as 71,807 are not invalid. They are simply string representation of the number 71807. Fix it so they are stored as 71807, and the column datatype should either be a float or int.

For each of the columns modified , show example rows of what was modified. Also call the info() method on each of the dataframes.

[62]:
```
# Function to clean numeric columns
def clean_numeric_column(dfx, column_name):
    """
    Converts a column to numeric by:
    - Removing commas (e.g., "71,807" → "71807")
    - Converting to float or int
    - Dropping invalid values
    """
    dfx[column_name] = dfx[column_name].astype(str).str.replace(',', '')  #␣
 ↪Remove commas
    dfx[column_name] = pd.to_numeric(dfx[column_name], errors='coerce')  #␣
 ↪Convert to number
    before_drop = len(dfx)
    dfx = dfx.dropna(subset=[column_name])  # Drop rows with invalid values
    after_drop = len(dfx)

  # Show examples of modified rows (if any rows were dropped)
    if before_drop != after_drop:
        print(f"Modified rows in {column_name} (before dropping NaN values):")
        print(dfx[[column_name]].head())

    return dfx
```

[64]:
```
# Clean 'threads' column in df_cpu
df_cpu = clean_numeric_column(df_cpu, 'threads')
```

```
Modified rows in threads (before dropping NaN values):
   threads
```

```
1     68.0
2     16.0
3     13.0
4     12.0
5     18.0
```

[66]:
```
# Clean 'total_byte_per_second' column in df_disk
df_disk = clean_numeric_column(df_disk, 'total_byte_per_second')
```

[68]:
```
# Clean 'working_set_kb' column in df_memory
df_memory = clean_numeric_column(df_memory, 'working_set_kb')
```

[70]:
```
# Display updated DataFrame info
print("\nCPU DataFrame Info:")
df_cpu.info()

print("\nDisk DataFrame Info:")
df_disk.info()

print("\nMemory DataFrame Info:")
df_memory.info()
```

```
CPU DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
Index: 304 entries, 1 to 305
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   image        304 non-null    object
 1   pid          304 non-null    object
 2   description  289 non-null    object
 3   status       304 non-null    object
 4   threads      304 non-null    float64
 5   cpu          304 non-null    int64
 6   average_cpu  304 non-null    float64
dtypes: float64(2), int64(1), object(4)
memory usage: 19.0+ KB

Disk DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 7 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   image                 14 non-null     object
 1   pid                   14 non-null     int64
 2   model                 14 non-null     object
```

```
 3   read_byte_per_second   14 non-null     object
 4   write_byte_per_second  14 non-null     object
 5   delay                  14 non-null     int64
 6   total_byte_per_second  14 non-null     int64
dtypes: int64(3), object(4)
memory usage: 916.0+ bytes

Memory DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 306 entries, 0 to 305
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   image            306 non-null    object
 1   pid              306 non-null    int64
 2   hard_faults/sec  306 non-null    int64
 3   commit_kb        306 non-null    object
 4   working_set_kb   306 non-null    int64
 5   shareable_kb     306 non-null    object
 6   private_kb       306 non-null    object
dtypes: int64(3), object(4)
memory usage: 16.9+ KB
```

2.0.4 4. Data Aggregation (30 pts):

a. **Develop an `aggregate_data(dfx)` function that:**

- Accepts one of the dataframes (`df_cpu`, `df_disk`, or `df_memory`).
- Dynamically identifies the dataset based on its columns.
- Aggregates the data based on the specific dataset requirements provided below and returns an aggregated dataframe. You can name this new aggregated dataframe whatever name works (examples given).

b. **Aggregation Requirements:**

i. For `df_cpu`:

```
1. Group by 'image' and aggregate the 'threads' column. Use the sum() function for agg
2. The resultant dataframe should contain: 'image_name', 'process_qty', 'threads_sum'
3. Reset the index after aggregation.
4. Assign the new aggregated dataframe values to df_cpu_sum.
```

ii. For `df_disk`:

1. Group by 'image' and aggregate the 'total_byte_per_second' column.Use the sum() function for aggregation.
2. The resultant dataframe should contain: 'image_name', 'process_qty', 'total_byte_sum'.
3. Reset the index after aggregation.
4. Assign the new aggregated dataframe values to df_disk_sum.

iii. For `df_memory`:

1. Group by 'image' and aggregate the 'working_set_kb' column. Use the sum() function for aggregation.
2. The resultant dataframe should contain: 'image_name', 'process_qty', 'working_set_sum'.
3. Reset the index after aggregation.
4. Assign the new aggregated dataframe values to df_memory_sum.

Note: 'process_qty' is defined by grouping by 'image' and determining the number of times a particular image name occurs. Use the size() function for aggregation

c. **Run `aggregate_data()` for each of your dataframes and save them into new dataframes. Print the head(10) and tail(10) for each of the new dataframes.**

Example:

Function header:

```
def aggregate_data(dfx):
    # Your code aggregates the data based on the requirements provided.
    return df_combo  # Name this resultant dataframe whatever name that works
```

Calling function and assigning the resultant dataframe to `df_cpu_sum`, `df_disk_sum`, `df_memory_sum`:

```
df_cpu_sum = aggregate_data(df_cpu)
df_disk_sum = aggregate_data(df_disk)
df_memory_sum = aggregate_data(df_memory)
```

[72]:
```
# Function to aggregate data dynamically based on dataset type
def aggregate_data(dfx):
    """
    Aggregates the given dataframe based on its dataset type:
    - df_cpu: Groups by 'image', sums 'threads', and counts occurrences.
    - df_disk: Groups by 'image', sums 'total_byte_per_second', and counts␣
 ↪occurrences.
    - df_memory: Groups by 'image', sums 'working_set_kb', and counts␣
 ↪occurrences.
    """
    # Identify dataset type based on columns
    if 'threads' in dfx.columns:
        df_combo = dfx.groupby('image').agg(
            process_qty=('image', 'size'),
            threads_sum=('threads', 'sum')
        ).reset_index()
        df_combo.rename(columns={'image': 'image_name'}, inplace=True)

    elif 'total_byte_per_second' in dfx.columns:
        df_combo = dfx.groupby('image').agg(
```

```
            process_qty=('image', 'size'),
            total_byte_sum=('total_byte_per_second', 'sum')
        ).reset_index()
        df_combo.rename(columns={'image': 'image_name'}, inplace=True)

    elif 'working_set_kb' in dfx.columns:
        df_combo = dfx.groupby('image').agg(
            process_qty=('image', 'size'),
            working_set_sum=('working_set_kb', 'sum')
        ).reset_index()
        df_combo.rename(columns={'image': 'image_name'}, inplace=True)

    else:
        raise ValueError("Unknown dataset structure. Unable to aggregate.")

    return df_combo
```

[74]:
```
# Apply the aggregation function to each dataset
df_cpu_sum = aggregate_data(df_cpu)
df_disk_sum = aggregate_data(df_disk)
df_memory_sum = aggregate_data(df_memory)
```

[76]:
```
# Print head(10) and tail(10) for each aggregated dataframe
print("\nCPU Aggregated Data:")
print(df_cpu_sum.head(10))
print(df_cpu_sum.tail(10))

print("\nDisk Aggregated Data:")
print(df_disk_sum.head(10))
print(df_disk_sum.tail(10))

print("\nMemory Aggregated Data:")
print(df_memory_sum.head(10))
print(df_memory_sum.tail(10))
```

```
CPU Aggregated Data:
                      image_name  process_qty  threads_sum
0  AcrobatNotificationClient.exe            1         13.0
1      Adobe Crash Processor.exe            1          4.0
2      Adobe Desktop Service.exe            1         47.0
3            AdobeCollabSync.exe            2         24.0
4             AdobeIPCBroker.exe            1         26.0
5    AdobeNotificationClient.exe            1         12.0
6         AdobeUpdateService.exe            1          5.0
7             AggregatorHost.exe            1          1.0
8       ApplicationFrameHost.exe            1          6.0
9                  CCLibrary.exe            1          1.0
```

	image_name	process_qty	threads_sum
140	svchost.exe (osprivacy -p)	1	4.0
141	svchost.exe (smphost)	1	6.0
142	svchost.exe (utcsvc -p)	1	12.0
143	taskhostw.exe	1	8.0
144	uihost.exe	1	63.0
145	unsecapp.exe	1	2.0
146	vpnagent.exe	1	7.0
147	wininit.exe	1	2.0
148	winlogon.exe	1	3.0
149	wlanext.exe	1	2.0

Disk Aggregated Data:

	image_name	process_qty	total_byte_sum
0	EXCEL.EXE	1	71807
1	Grammarly.Desktop.exe	1	6729
2	MsMpEng.exe	1	1457
3	OUTLOOK.EXE	1	1471
4	Registry	1	5523
5	System	1	232627
6	Teams.exe	1	547
7	WUDFHost.exe	1	24587
8	chrome.exe	2	11218
9	msedgewebview2.exe	4	20060

	image_name	process_qty	total_byte_sum
0	EXCEL.EXE	1	71807
1	Grammarly.Desktop.exe	1	6729
2	MsMpEng.exe	1	1457
3	OUTLOOK.EXE	1	1471
4	Registry	1	5523
5	System	1	232627
6	Teams.exe	1	547
7	WUDFHost.exe	1	24587
8	chrome.exe	2	11218
9	msedgewebview2.exe	4	20060

Memory Aggregated Data:

	image_name	process_qty	working_set_sum
0	AcrobatNotificationClient.exe	1	3632
1	Adobe Crash Processor.exe	1	15252
2	Adobe Desktop Service.exe	1	85504
3	AdobeCollabSync.exe	2	37012
4	AdobeIPCBroker.exe	1	10568
5	AdobeNotificationClient.exe	1	2836
6	AdobeUpdateService.exe	1	9580
7	AggregatorHost.exe	1	7420
8	ApplicationFrameHost.exe	1	35988
9	CCLibrary.exe	1	2948

```
                    image_name  process_qty  working_set_sum
141  svchost.exe (osprivacy -p)            1            13812
142       svchost.exe (smphost)            1            16624
143     svchost.exe (utcsvc -p)            1            30104
144                taskhostw.exe            1            19400
145                   uihost.exe            1            10472
146                 unsecapp.exe            1             7528
147                 vpnagent.exe            1            26192
148                  wininit.exe            1             6264
149                 winlogon.exe            1            12320
150                  wlanext.exe            1             6764
```

2.0.5 5. Merging Data (10 pts):

a. **Use the Pandas merge function to do an inner join on 'image_name' for dataframes `df_cpu_sum` and `df_disk_sum` created in the previous step. Store the result in a dataframe named `df_new`.**

b. **Further merge `df_new` with `df_memory_sum` using an inner join on 'image_name'.**

c. **Show the resulting dataframe in each of the steps above**

Please refer to the pandas merge method and its parameters here : https://pandas.pydata.org/docs/reference/api/pandas.merge.html

[78]:
```
# Merge df_cpu_sum and df_disk_sum on 'image_name' using an inner join
df_new = pd.merge(df_cpu_sum, df_disk_sum, on='image_name', how='inner')
```

[80]:
```
# Display the merged DataFrame after first merge
print("Merged CPU and Disk Data (first 10 rows):")
print(df_new.head(10))
print(df_new.info())  # Show DataFrame structure
```

```
Merged CPU and Disk Data (first 10 rows):
              image_name  process_qty_x  threads_sum  process_qty_y  \
0             EXCEL.EXE              1         64.0              1
1  Grammarly.Desktop.exe              1         48.0              1
2           MsMpEng.exe              1         66.0              1
3           OUTLOOK.EXE              1         89.0              1
4              Registry              1          4.0              1
5                System              1        342.0              1
6             Teams.exe              9        213.0              1
7          WUDFHost.exe              6         66.0              1
8            chrome.exe             12        236.0              2
9    msedgewebview2.exe             32        620.0              4

   total_byte_sum
0           71807
1            6729
2            1457
```

```
3            1471
4            5523
5          232627
6             547
7           24587
8           11218
9           20060
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   image_name      10 non-null     object
 1   process_qty_x   10 non-null     int64
 2   threads_sum     10 non-null     float64
 3   process_qty_y   10 non-null     int64
 4   total_byte_sum  10 non-null     int64
dtypes: float64(1), int64(3), object(1)
memory usage: 532.0+ bytes
None
```

[82]:
```
# Merge df_new with df_memory_sum on 'image_name' using an inner join
df_new = pd.merge(df_new, df_memory_sum, on='image_name', how='inner')
```

[84]:
```
# Display the final merged DataFrame
print("\nFinal Merged Data (first 10 rows):")
print(df_new.head(10))
print(df_new.info())  # Show DataFrame structure
```

```
Final Merged Data (first 10 rows):
              image_name  process_qty_x  threads_sum  process_qty_y  \
0             EXCEL.EXE              1         64.0              1
1  Grammarly.Desktop.exe              1         48.0              1
2           MsMpEng.exe              1         66.0              1
3           OUTLOOK.EXE              1         89.0              1
4              Registry              1          4.0              1
5                System              1        342.0              1
6            Teams.exe              9        213.0              1
7          WUDFHost.exe              6         66.0              1
8            chrome.exe             12        236.0              2
9    msedgewebview2.exe             32        620.0              4

   total_byte_sum  process_qty  working_set_sum
0           71807            1           298380
1            6729            1           273436
2            1457            1           210204
3            1471            1           447040
```

```
4            5523            1            40864
5          232627            1             3140
6             547           10          1117452
7           24587            6            69316
8           11218           12          1076912
9           20060           32          1002516
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   image_name       10 non-null     object
 1   process_qty_x    10 non-null     int64
 2   threads_sum      10 non-null     float64
 3   process_qty_y    10 non-null     int64
 4   total_byte_sum   10 non-null     int64
 5   process_qty      10 non-null     int64
 6   working_set_sum  10 non-null     int64
dtypes: float64(1), int64(5), object(1)
memory usage: 692.0+ bytes
None
```

2.0.6 6. Filtering Data (30 pts):

a. **Filter `df_new` to obtain:**

i. `image_name` that had 'working_set_sum' greater than 200,000. Store the result in a variable named 'high_memory_image'.

ii. `image_name` that had 'thread_sum' greater than 200. Store the result in a variable named 'high_thread_image'.

iii. `image_name` that had 'working_set_sum' greater than 200,000, 'thread_sum' less than 50, and 'total_byte_sum' less than 7,000. Store the result in a variable named 'hi_mem_low_thread_low_io'.

iv. Show each of the filtered dataframe in separate cells. Feel free to add new cells to this notebook.

[86]:
```
# Filter for 'working_set_sum' greater than 200,000
high_memory_image = df_new[df_new['working_set_sum'] > 200000]
```

[88]:
```
# Filter for 'threads_sum' greater than 200
high_thread_image = df_new[df_new['threads_sum'] > 200]
```

[90]:
```
# Filter for 'working_set_sum' > 200,000, 'threads_sum' < 50, and␣
↪'total_byte_sum' < 7,000
hi_mem_low_thread_low_io = df_new[
    (df_new['working_set_sum'] > 200000) &
    (df_new['threads_sum'] < 50) &
```

```
    (df_new['total_byte_sum'] < 7000)
]
```

[92]:
```
# Display the results
print("High Memory Image:")
print(high_memory_image)

print("\nHigh Thread Image:")
print(high_thread_image)

print("\nHigh Memory, Low Thread, Low IO:")
print(hi_mem_low_thread_low_io)
```

```
High Memory Image:
              image_name  process_qty_x  threads_sum  process_qty_y  \
0              EXCEL.EXE              1         64.0              1
1  Grammarly.Desktop.exe              1         48.0              1
2            MsMpEng.exe              1         66.0              1
3            OUTLOOK.EXE              1         89.0              1
6              Teams.exe              9        213.0              1
8             chrome.exe             12        236.0              2
9     msedgewebview2.exe             32        620.0              4

   total_byte_sum  process_qty  working_set_sum
0           71807            1           298380
1            6729            1           273436
2            1457            1           210204
3            1471            1           447040
6             547           10          1117452
8           11218           12          1076912
9           20060           32          1002516

High Thread Image:
           image_name  process_qty_x  threads_sum  process_qty_y  \
5              System              1        342.0              1
6           Teams.exe              9        213.0              1
8          chrome.exe             12        236.0              2
9  msedgewebview2.exe             32        620.0              4

   total_byte_sum  process_qty  working_set_sum
5          232627            1             3140
6             547           10          1117452
8           11218           12          1076912
9           20060           32          1002516

High Memory, Low Thread, Low IO:
              image_name  process_qty_x  threads_sum  process_qty_y  \
1  Grammarly.Desktop.exe              1         48.0              1
```

```
   total_byte_sum  process_qty  working_set_sum
1            6729            1           273436
```

2.0.7 Coding Style (5 pts)

Although we do not enforce a coding style such as PEP 8 (https://peps.python.org/pep-0008/) , please ensure that you have comments for each of the functions defined. Your code is readable, and includes only the code that is required by the assignment. Please remove any commented code, and experimental code that you may have tried. For each of the questions be sure to show some example rows of the dataframe that was modified or created.

2.1 Submission on Gradescope

Gradescope canvas left menu -> Gradescop -> PSET 5: Exploratory Data Analysis

Submission : Submit the jupyter notebook, and a pdf version of this notebook.

To create a pdf of this notebook : In your browser open print, and save as pdf. Name the pdf LastNameFirstName_pset5.pdf example: DoeJohn_pset5.pdf

Name this jupyter notebook with the same format LastNameFirstName_pset5.ipynb

Make sure that your notebook has been run before creating pdf. Any outputs from running the code needs to be clearly visible. We need both .ipynb, and pdf of this notebook to assign you grades.

Drop all the files in gradescope under PSET 5: Exploratory Data Analysis.

[ ]: