MTH 765P Mini-project

Thanh Trung Vu

January 15, 2024

1 Introduction

This dataset comprises a comprehensive collection of Solid State Drives (SSDs). It has been gathered by enthusiasts and serves as a research tool for consumers contemplating purchases.

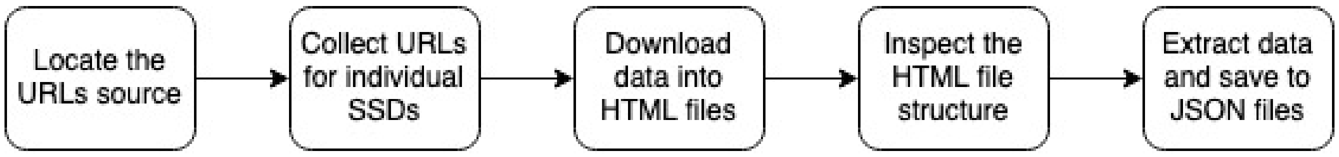
The analysis of this dataset can be beneficial for those seeking detailed information to guide their SSD purchasing decisions, especially when the market is saturated with various SSD options. This variety makes it very difficult to choose the right one as each individual's needs and preferences vary. Using this dataset, one can for example compare the latest models with older generations, which might reveal that older versions are still adequate, or perhaps even better in certain respects.

By analysing this dataset, we can learn some interesting information, such as the annual trends in SSD production, prices, DRAM cache sizes among others.

The bulk of this project lies in the process of collecting data by web scraping and then converting HTML files to JSON for storage. I will also briefly describe how the data is processed before it is used in some simple analysis and visualisation.

2 Data Collection

The process of collecting and storing data looks as follows:


Locate the URLs source
Collect URLs for individual SSDs
Download data into HTML files
Inspect the HTML file structure
Extract data and save to JSON files

Process of collecting and storing data

This can be summarised into two main steps:

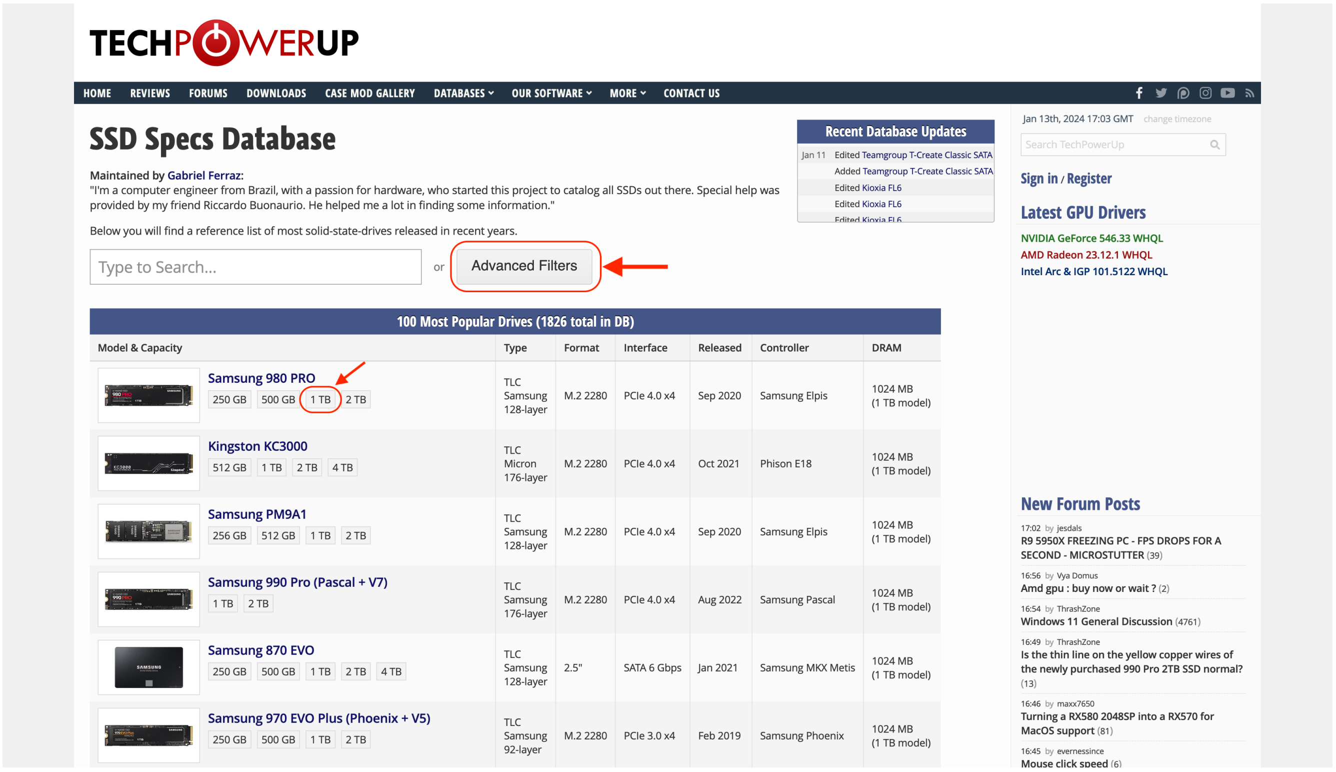
- Step 1: **Retrieving URLs for SSD Data Pages**

  This step involves identifying and collecting the web addresses (URLs) of the data pages for various SSDs. This is where specific links to each product's detailed information are gathered.

- Step 2: **Obtaining HTML Documents of the Data Pages**

  In this step, the actual HTML documents of the SSD data pages, accessible through the URLs obtained in Step 1, are downloaded. These documents contain the detailed information and specifications of each SSD, which will be further processed and analyzed.

2.1 Retrieving URLs for SSD Data Pages



SSD Specs Database

Maintained by Gabriel Ferraz:
"I'm a computer engineer from Brazil, with a passion for hardware, who started this project to catalog all SSDs out there. Special help was provided by my friend Riccardo Buonaurio. He helped me a lot in finding some information."

Below you will find a reference list of most solid-state-drives released in recent years.

Recent Database Updates

Jan 11 Edited Teamgroup T-Create Classic SATA
Added Teamgroup T-Create Classic SATA
Edited Kioxia FL6
Edited Kioxia FL6
Edited Kioxia FL6

100 Most Popular Drives (1826 total in DB)

Model & Capacity	Type	Format	Interface	Released	Controller	DRAM
Samsung 980 PRO 250 GB 500 GB 1 TB 2 TB	TLC Samsung 128-layer	M.2 2280	PCIe 4.0 x4	Sep 2020	Samsung Elpis	1024 MB (1 TB model)
Kingston KC3000 512 GB 1 TB 2 TB 4 TB	TLC Micron 176-layer	M.2 2280	PCIe 4.0 x4	Oct 2021	Phison E18	1024 MB (1 TB model)
Samsung PM9A1 256 GB 512 GB 1 TB 2 TB	TLC Samsung 128-layer	M.2 2280	PCIe 4.0 x4	Sep 2020	Samsung Elpis	1024 MB (1 TB model)
Samsung 990 Pro (Pascal + V7) 1 TB 2 TB	TLC Samsung 176-layer	M.2 2280	PCIe 4.0 x4	Aug 2022	Samsung Pascal	1024 MB (1 TB model)
Samsung 870 EVO 250 GB 500 GB 1 TB 2 TB 4 TB	TLC Samsung 128-layer	2.5"	SATA 6 Gbps	Jan 2021	Samsung MKX Metis	1024 MB (1 TB model)
Samsung 970 EVO Plus (Phoenix + V5) 250 GB 500 GB 1 TB 2 TB	TLC Samsung 92-layer	M.2 2280	PCIe 3.0 x4	Feb 2019	Samsung Phoenix	1024 MB (1 TB model)

Jan 13th, 2024 17:03 GMT

Sign in / Register

Latest GPU Drivers

NVIDIA GeForce 546.33 WHQL
AMD Radeon 23.12.1 WHQL
Intel Arc & IGP 101.5122 WHQL

New Forum Posts

17:02 by jesdals
R9 5950X FREEZING PC - FPS DROPS FOR A SECOND - MICROSTUTTER (39)

16:56 by Vya Domus
Amd gpu : buy now or wait ? (2)

16:54 by ThrashZone
Windows 11 General Discussion (4761)

16:49 by ThrashZone
Is the thin line on the yellow copper wires of the newly purchased 990 Pro 2TB SSD normal? (13)

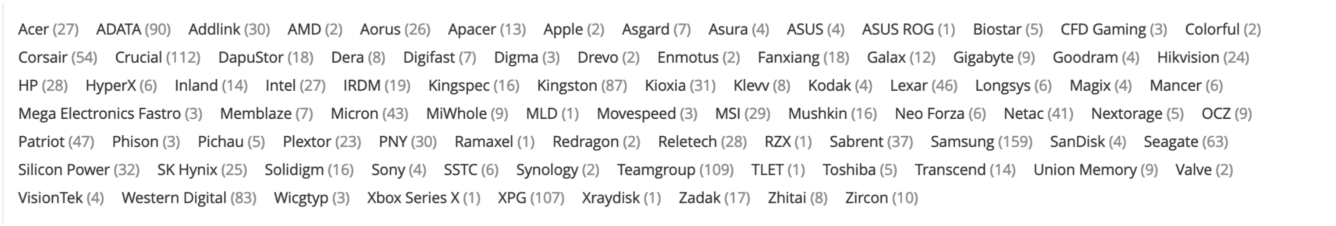
16:46 by maxx7650
Turning a RX580 2048SP into a RX570 for MacOS support (81)

16:45 by evernessince
Mouse click speed (6)

The SSD database page (the "root" for scraping)

I began the process by accessing the SSD database at **TechPowerUp** (*https://www.techpowerup.com/ssd-specs/*). Above is a screenshot of the SSD database page. It's important to note that the screenshot was taken after the data collection was completed, which may result in some discrepancies between the displayed information and the data used in this project.

This page contains a list of SSDs, from which I retrieved the URL for each drive's data page. Each of the small buttons with a drive capacity value contains the link to a data page. However, the page only shows at most 100 drives at a time. To obtain all the URLs, I had to make several queries, each targeting the drives produced by a specific manufacturer. This means that I first had to compile a list of URLs for each SSD brand.



Acer (27) ADATA (90) Addlink (30) AMD (2) Aorus (26) Apacer (13) Apple (2) Asgard (7) Asura (4) ASUS (4) ASUS ROG (1) Biostar (5) CFD Gaming (3) Colorful (2)
Corsair (54) Crucial (112) DapuStor (18) Dera (8) Digifast (7) Digma (3) Drevo (2) Enmotus (2) Fanxiang (18) Galax (12) Gigabyte (9) Goodram (4) Hikvision (24)
HP (28) HyperX (6) Inland (14) Intel (27) IRDM (19) Kingspec (16) Kingston (87) Kioxia (31) Klevv (8) Kodak (4) Lexar (46) Longsys (6) Magix (4) Mancer (6)
Mega Electronics Fastro (3) Memblaze (7) Micron (43) MiWhole (9) MLD (1) Movespeed (3) MSI (29) Mushkin (16) Neo Forza (6) Netac (41) Nextorage (5) OCZ (9)
Patriot (47) Phison (3) Pichau (5) Plextor (23) PNY (30) Ramaxel (1) Redragon (2) Reletech (28) RZX (1) Sabrent (37) Samsung (159) SanDisk (4) Seagate (63)
Silicon Power (32) SK Hynix (25) Solidigm (16) Sony (4) SSTC (6) Synology (2) Teamgroup (109) TLET (1) Toshiba (5) Transcend (14) Union Memory (9) Valve (2)
VisionTek (4) Western Digital (83) Wicgtyp (3) Xbox Series X (1) XPG (107) Xraydisk (1) Zadak (17) Zhitai (8) Zircon (10)

The manufacturer list in the Adanvanced Filters

To get the URLs of all SSD manufacturers, I accessed the Advanced Filters feature on the page. Here I found the full list of brands as shown above. You might notice a count next to each brand, representing the number of data pages, not the actual number of drives. The number of drives is smaller than the number of data pages since a drive can have multiple versions with different capacities. This discrepancy arises because a single drive model can have multiple versions, each with different storage capacities. For instance, the brand Samsung is listed with 159 drives, but this number includes all the different capacity versions of each model. However, the actual number of distinct drive models is fewer than 100, allowing us to easily obtain all URLs from the database page after applying the filter.

Below is a simplified excerpt from the database page where I found the URLs of the manufacturers.

```html
<html lang="en">
  <head><title>SSD Database | TechPowerUp</title></head>
  <body class="body ssddb"><h1 class="pagetitle">SSD Specs Database</h1>
  <div class="filter-list">
    <div class="filter-list-item" data-title="Manufacturer">
      <div class="filter-list-item-header"><span>Manufacturer</span></div>
      <div class="filter-list-item-entries">
        <a href="/ssd-specs/filter/?mfgr=Acer"
          >Acer <span class="filter-list-item-entry-count">(27)</span></a>
        <a href="/ssd-specs/filter/?mfgr=ADATA"
          >ADATA <span class="filter-list-item-entry-count">(90)</span></a>
        <a href="/ssd-specs/filter/?mfgr=Addlink"
          >Addlink <span class="filter-list-item-entry-count">(30)</span></a>
        ...
      </div>
    </div>
  </div>
  </body>
</html>
```

A simplied excerpt from the HTML document of the SSD database page

Within this HTML document, I searched for the `<div>` element with attribute `data-title="Manufacturer"`. This element represents the filtering-by-brand functionality, which contains the full list of manufacturers. Then I simply extracted the URLs from the attribute `href` in all the `<a>` elements. For example, the URL for the brand Acer is shown here as */ssd-specs/filter/?mfgr=Acer*. Note that this has to be joined with the home page *https://www.techpowerup.com* at the front to get the full working URL.

After filtering by brand, the next step is to extract the URLs of the individual data pages for the SSD drives from that particular brand.

2.2 Obtaining HTML Documents of the Data Pages

```html
<html lang="en">
  <head><title>SSD Database | TechPowerUp</title></head>
  <body class="body ssddb"><h1 class="pagetitle">SSD Specs Database</h1>
  <table class="drives-desktop-table">
    <tr>
      <td>
        <div class="drive-title">
          <a href="/ssd-specs/acer-fa100-1-tb.d333" class="drive-name"
            >Acer FA100 (Micron B27B)</a>
          <div class="drive-capacities">
            <a
              class="drive-capacity"
              href="/ssd-specs/acer-fa100-512-gb.d334"
              style="white-space: nowrap"
              >512 GB</a>
            <a
              class="drive-capacity"
              href="/ssd-specs/acer-fa100-1-tb.d333"
              style="white-space: nowrap"
              >1 TB</a>
            ...
          </div>
        </div>
      </td>
    </tr>
    ...
  </table>
  </body>
</html>
```

A simplied excerpt from the HTML document of the database page after filtered by brand

This simplied excerpt is taken from the HTML document from the URL *https://www.techpowerup.com/ssd-specs/filter/?mfgr=Acer*. This is basically the same as the database page, except that the SSD drive list now only contains SSD drives manufactured by Acer.

To get the URLs for the data pages for all the individual Acer drives (each with all of its variants with different capacities), I searched for all the `<a>` elements with attribute `class="drive-capacity"` and extracted the URL from the `href` attribute.

After gathering all the URLs, it was simply the matter of making a HTTP request to each URL to get all the HTML elements. This process took the largest amount of time by far. This is because I could not make too many requests in a short time. If I made too many requests, the site would return `HTTP 429` (Too Many Requests) errors. Therefore, I got to include a delay time between 60 and 90 seconds before each making a request. Since there are well over one thousand URLs to access, this step took multiple hours, including retries when something went wrong.

In the next section, I will describe the structure of the HTML files downloaded from the data pages and how I extracted and stored these data.

3 Data Storing

3.1 HTML Structure

```
<html lang="en">
  <head>
    <title>Acer FA100 1 TB Specs | TechPowerUp SSD Database</title>
    <meta property="og:url"
    content="https://www.techpowerup.com/ssd-specs/acer-fa100-1-tb.d333"/>
  </head>

  <body class="body ssddb-details">
    <h1 class="drivename">Acer FA100 1 TB (Micron B27B)</h1>

    <div class="sectioncontainer">
      <section class="details">
        <h1>Solid-State-Drive</h1>
        <table>
          <tr> <th>Capacity:</th> <td>1 TB (1024 GB)</td> </tr>
          <tr> <th>Production:</th> <td>Active</td> </tr>
          <tr> <th>Released:</th> <td>Apr 30th, 2021</td> </tr>
          <tr> <th>Price at Launch:</th> <td>105 USD</td> </tr>
          <tr> <th>Market:</th> <td>Consumer</td> </tr>
          ...
        </table>
      </section>

      <section class="details">
        <h1>Physical</h1>
        <table>
          <tr> <th>Power Draw:</th> <td>
              1.00 W (Idle)<br />
              2.0 W (Avg)<br />
```

```
                3.7 W (Max)<br />
              </td> </tr>
            ...
          </table>
        </section>

        <section class="details">
          <h1>NAND Flash</h1>
          <table>
            <tr> <th>Type:</th> <td>TLC</td> </tr>
            ...
          </table>
        </section>

        <section class="details">
          <h1>DRAM Cache</h1>
          <table>
            <tr> <th>Type:</th> <td>None</td> </tr>
            ...
          </table>
        </section>

        <section class="details">
          <h1>Performance</h1>
          <table>
            <tr> <th>Sequential Read:</th> <td>3,300 MB/s</td> </tr>
            <tr> <th>Sequential Write:</th> <td>2,700 MB/s</td> </tr>
            <tr> <th>Random Read:</th> <td>325,000 IOPS</td> </tr>
            <tr> <th>Random Write:</th> <td>293,000 IOPS</td> </tr>
            <tr> <th>Warranty:</th> <td>5 Years</td> </tr>
            ...
          </table>
        </section>

        ...

      </div>
    </body>
  </html>
```

A simplified excerpt from the HTML document of a data page

This simplied excerpt is taken from the HTML document from the URL *https://www.techpowerup.com/ssd-specs/acer-fa100-1-tb.d333*. This is the data page for a version of the drive Acer FA100. From the name `Acer FA100 1 TB (Micron B27B)`, this version has 1 TB of capacity and uses Micron B27B NAND Flash chips.

To distinguish between different SSD drives, I used the ending of a drive's URL as its ID. For example, this drive was taken from the URL *https://www.techpowerup.com/ssd-specs/acer-fa100-1-tb.d333* so I used `acer-fa100-1-tb.d333` as its ID.

I also wanted each drive to have a name. For this I take the text within the `<h1>` element with attribute `class="drivename"`. In this case the drive's name would be `Acer FA100 1 TB (Micron B27B)`.

The rest of the data are found within the `<section>` elements with attribute `class="details"`. Each of these elements contains a `<table>` element. The name of the table is found in the inner `<h1>` element. In this example, the tables are `Solid-State-Drive`, `Physical`, `NAND Flash`, `DRAM Cache` and `Performance`. Each table contains a group of attributes, each attribute is stored in a row represented by the `<tr>` tag. From each row, I extracted the name of the attribute's name from the `<th>` element and the attribute's value from the `<td>` element.

3.2 Storing as a JSON file

The attributes extracted from a HTML document were then temporarily stored in memory in a large dictionary object, where each element is an SSD drive with the key being its ID. I repeated this process for all the remaining HTML documents. Then I wrote the dictionary into a JSON file.

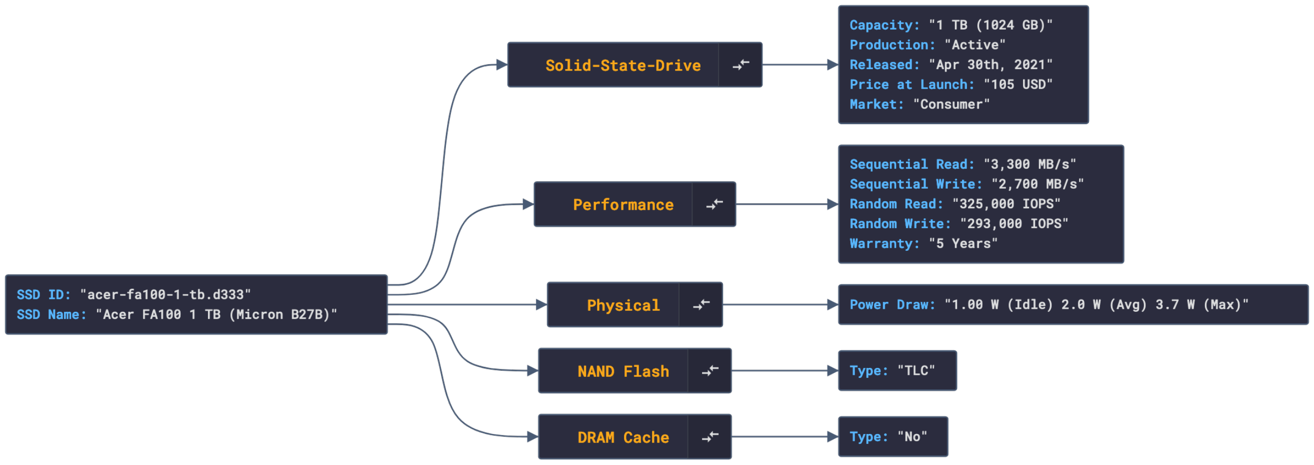
The resulting JSON file looks like this. Similar to all the examples above, this is only a simplified excerpt from the real data file. This section shows the resulting entry for the drive `Acer FA100 1 TB (Micron B27B)` which was discussed above.

```json
{
  "acer-fa100-1-tb.d333": {
    "SSD ID": "acer-fa100-1-tb.d333",
    "SSD Name": "Acer FA100 1 TB (Micron B27B)",
    "Solid-State-Drive": {
      "Capacity": "1 TB (1024 GB)",
      "Production": "Active",
      "Released": "Apr 30th, 2021",
      "Price at Launch": "105 USD",
      "Market": "Consumer",
      ...
    },
    "Performance": {
      "Sequential Read": "3,300 MB/s",
      "Sequential Write": "2,700 MB/s",
      "Random Read": "325,000 IOPS",
      "Random Write": "293,000 IOPS",
      "Warranty": "5 Years",
      ...
    },
    "Physical": {
      "Power Draw": "1.00 W (Idle) 2.0 W (Avg) 3.7 W (Max)",
      ...
    },
    "NAND Flash": {
      "Type": "TLC",
      ...
    },
    "DRAM Cache": {
      "Type": "No",
      ...
    },
    ...
  },
  ...
}
```

An excerpt of the JSON file that stores attributes of all SSDs

For this step, I chose JSON as the data storage method for its simplicity as well as readability. It is easy to import the data from JSON into a dictionary which can then be flattened, a processing step which I will discuss in the next section, and loaded into a DataFrame. If I chose XML, I would end up with a longer file that is less readable. Alternatively, if I chose CSV, I would have to do the flattening first, the take the union of all the columns as there are attributes that are only present in some drives, and then write to the CSV file. Yet other alternatives include database solutions such as SQL and NoSQL which might be more efficient for large datasets but will increase the complexity of the code. For this project, since the amount of data is relatively small, I decided to keep the data storing simple to lower the chances of errors.

3.3 A brief desciption of the data


SSD ID: "acer-fa100-1-tb.d333"
SSD Name: "Acer FA100 1 TB (Micron B27B)"
Solid-State-Drive
Capacity: "1 TB (1024 GB)"
Production: "Active"
Released: "Apr 30th, 2021"
Price at Launch: "105 USD"
Market: "Consumer"
Performance
Sequential Read: "3,300 MB/s"
Sequential Write: "2,700 MB/s"
Random Read: "325,000 IOPS"
Random Write: "293,000 IOPS"
Warranty: "5 Years"
Physical
Power Draw: "1.00 W (Idle) 2.0 W (Avg) 3.7 W (Max)"
NAND Flash
Type: "TLC"
DRAM Cache
Type: "No"

Tree graph view produced by https://omute.net/editor

To get a better feel of the data, I visualised the data of an SSD drive as a tree graph using *https://omute.net/editor*. The data of the SSD drive shown here contains five groups of attributes corresponding the five tables mentioned above. Each group contains a set of attributes and their corresponding values. For example, here the group `Solid-State-Drive` contains five attributes: `Capacity`, `Production`, `Released`, `Price` and `Market`. The `Solid-State-Drive` group contains some general information about the drive.

Next we have the group `Performance`. This group contains attributes about different aspects of a drive's performance, including the read and write speeds. It also contains the length of warranty given by the manufacturer.

The group `Physical` contains more information about the drive. An example is the attribute `Power Draw` which shows how the drive's power consumption. It includes the idle or minimum value, the maximum value and the average value of Watts that the drive draws

The group `NAND Flash` shows information about the memory chips used on the drive, such as the type of chips (SLC, MLC, TLC, QLC and so on...). This represents the number of layers or bits that can be stored in a memory cell. More layers mean more capacity per cell, allowing for higher-capacity drives at the same size. In exchange, this lowers the performance and durability of the drive.

The group `DRAM Cache` tells us whether a drive has dedicated memory cache to help improving performance and how much cache capacity there is if any. Drives without DRAM Cache usually have lower prices in exchange for the performance drop.

Please note that these are only the attribute groups and the attributes that I selected for processing and analysis. There are more attribute groups (such as `Controller` and `Features`) and also more attributes in these selected groups (such as `Endurance` and `MTBF` in the group `Performance`) which I will not use in this project but can be of interest for some others.

4 Data Preparation

Before processing, I first had to import the data from the JSON file into a DataFrame. Even though `pandas` has a convenient function `read_json`, I could not use it to directly import all the data because of the multi-level structure of the data in the JSON file. The attributes are still divided into groups. To solve this, I first read the JSON file into a dictionary using `json.load`. Then I flattened each of the dictionary by combining the group name with the attribute name. This gave me a dictionary that looks like this

```
{
  "acer-fa100-1-tb.d333": {
    "SSD ID": "acer-fa100-1-tb.d333",
    "SSD Name": "Acer FA100 1 TB (Micron B27B)",
    "Solid-State-Drive - Capacity": "1 TB (1024 GB)",
    "Solid-State-Drive - Market": "Consumer",
    "Solid-State-Drive - Price at Launch": "105 USD",
    "Solid-State-Drive - Production": "Active",
    "Solid-State-Drive - Released": "Apr 30th, 2021",
    "Performance - Random Read": "325,000 IOPS",
    "Performance - Random Write": "293,000 IOPS",
    "Performance - Sequential Read": "3,300 MB/s",
    "Performance - Sequential Write": "2,700 MB/s",
    "Performance - Warranty": "5 Years",
    "NAND Flash - Type": "TLC",
    "DRAM Cache - Type": "No",
    "Physical - Power Draw": "1.00 W (Idle) 2.0 W (Avg) 3.7 W (Max)",
    ...
  },
  ...
}
```

The dictionary after flattening

The flattened dictionary was then loaded into a DataFrame which then contained over 100 columns.

If we were to use all the attributes, there would potentially be a huge amount of processing that needed to be done. I decided to keep only the columns as shown in the dictionary above.

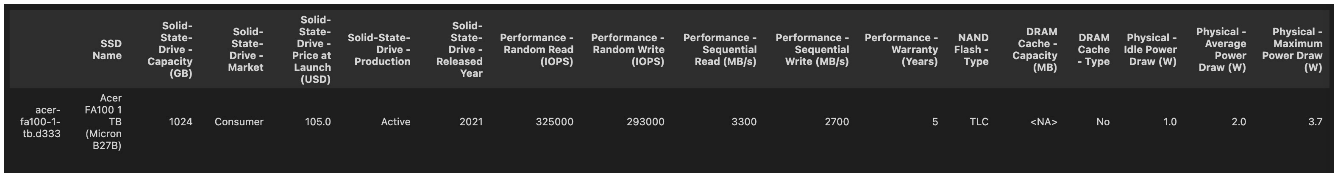
Some of these columns can be left as text while others need to be converted to other data types. For instance, columns about price at launch, read/write speeds and warranty need to be converted to numeric data types. To extract the number, I simply split the string using whitespace as the separator and took converted the first token to a number. I added the unit to the column name, for example from `Performance - Warranty` to `Performance - Warranty (Years)`. Some values such as the read/write speeds also contain commas as thousands separators. To parse this correctly, I set a suitable `locale` configuration and used `locale.atof` for number conversion.

The column `Solid-State-Drive - Released` contains information about when the drive is released by the manufacturer. Usually this would be converted to a datetime data type. However, the value in this column can be a full date like `Apr 30th, 2021`, or just the month `Apr 2021`, or just the year `2021`. Because of the inconsistency, I decided to keep only the year number and store it as an integer.

Another column with some consistency is the capacity where the capacity could be shown in GB unit only, like `512 GB`, or in both TB and GB units, like `1 TB (1024 GB)`. I kept only the values in GB.

The trickiest column was the `Physical - Power Draw` attribute. Each value has three numbers for Idle, Avg and Max power consumption. I separated these to 3 separate columns. More specifically, I replaced `"Power Draw":"1.00 W (Idle) 2.0 W (Avg) 3.7 W (Max)"` with `"Idle Power Draw (W)":"1.00"`, `"Avg Power Draw (W)":"2.0"` and `"Max Power Draw (W)":"3.7"`

The resulting DataFrame looks like this



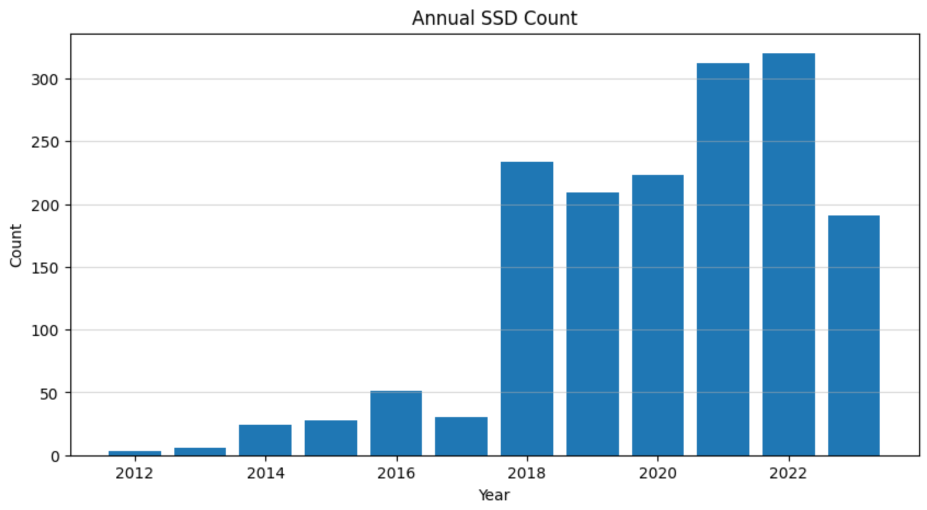
	SSD Name	Solid-State-Drive - Capacity (GB)	Solid-State-Drive - Market	Solid-State-Drive - Price at Launch (USD)	Solid-State-Drive - Production	Solid-State-Drive - Released Year	Performance - Random Read (IOPS)	Performance - Random Write (IOPS)	Performance - Sequential Read (MB/s)	Performance - Sequential Write (MB/s)	Performance - Warranty (Years)	NAND Flash - Type	DRAM Cache - Capacity (MB)	DRAM Cache - Type	Physical - Idle Power Draw (W)	Physical - Average Power Draw (W)	Physical - Maximum Power Draw (W)
acer-fa100-1-tb.d333	Acer FA100 1 TB (Micron B27B)	1024	Consumer	105.0	Active	2021	325000	293000	3300	2700	5	TLC	<NA>	No	1.0	2.0	3.7

The DataFrame after processing

In the next section I will make some simple visualisations to highlight a few characteristics of this dataset.

5 Data Analysis

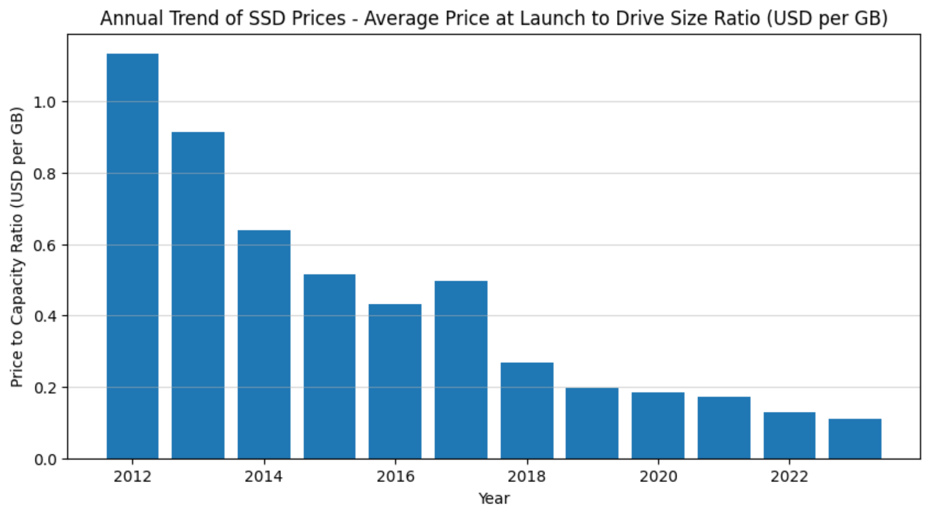
5.1 SSD Production



Annual SSD Production

By plotting the number of SSDs for each year, we can see that there was a sudden jump in SSD production in 2018. It seems that there were not many SSDs before 2018. This would make sense as SSDs were much more expensive back then and Hard Disk Drives (HDDs) were very common. This is assuming that the data is more or less complete, i.e. there were not many missing SSD data for the years before 2018.

5.2 Pricing

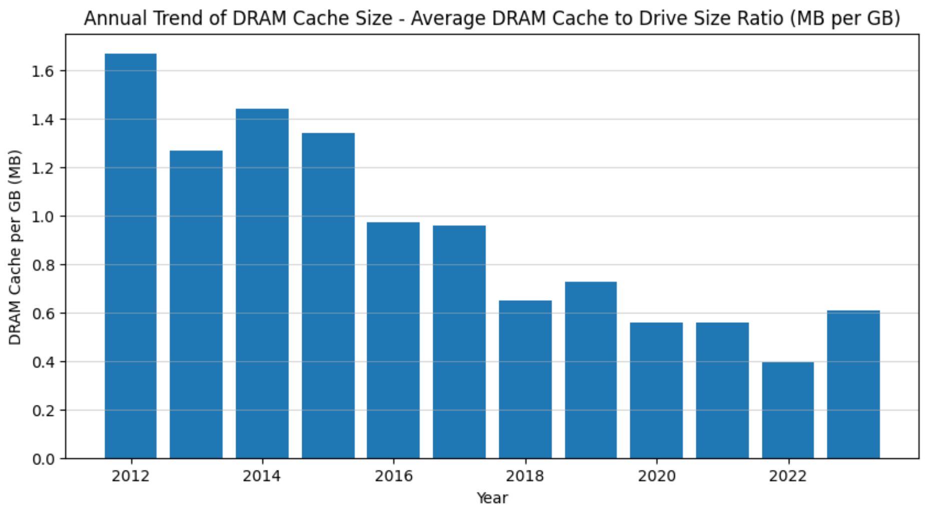


Annual Average "Price to Capacity" Ratio

A simple way to get a sense of value is to divide the price by the capacity of the drive. This consistently decreasing pattern indicates that drives are either becoming cheaper for the same capacity, or larger drives are being produced at a similar or lower cost due to advancements in production and technology.

While the decrease in price is beneficial, it raises the question: does this reduction come with with a cost in other aspects?

5.3 DRAM Cache



Annual Average "DRAM Cache Size to Capacity" Ratio

One noticeable trend is the decreasing amount of DRAM Cache, which suggests a potential regression. DRAM Cache is beneficial for performance, so its reduction could represent a trade-off to decrease prices. Consequently, consumers might be inclined to seek older generation SSDs with DRAM Cache, rather than newer models without it. However, this trend might also indicate that increasing DRAM Cache size no longer significantly enhances speed performance.

6 Libraries

Some important libraries used for this project are:

```python
import requests # send HTTP requests to get the HTML code of a web page
from bs4 import BeautifulSoup # parse HTML files
import time    # time.sleep to pause between requests to the server
import random # randomize the time between requests
import json # read and write json files
import glob # get the list of file names in a directory
import pandas as pd
import matplotlib.pyplot as plt
import re       # regular expression for finding numbers in strings
import locale    # convert string to numbers with thousand separators
```

7 Usage

I submitted three Jupyter notebooks as well as the accompanying data for demonstration purposes. These were designed to be run in order.

The first notebook, titled `step_1_collect_html_files.ipynb`, is designed for downloading the HTML files that store the data of the SSD drives. Please note that parts of the code might not work correctly, as it is primarily for demonstration purposes. Additionally, it makes several assumptions about the website's structure, which could change in the future.

The second notebook, `step_2_convert_html_to_json.ipynb`, is for extracting data from the HTML files and writing it to a JSON file. Since the JSON file is also provided, it is not necessary to run this notebook again to extract the data.

The last notebook, presumably titled `step_3_process_and_visualize_data.ipynb`, is for importing data from the JSON file into DataFrames for processing and visualisation.

8 Conclusion

The biggest challenge in this project was perhaps the process of collecting the data by scraping it directly from the website. It was difficult and time-consuming to ensure that the data was correctly scraped.

The second challenge involved storing the data. I opted to convert the individual HTML files into individual JSON files, which, while not ideal for large datasets, provided an easy starting point.

In the end, I obtained a collection of SSD data on which I did a preliminary processing and created a few visualisations showing some annual trends.

Anyone interested in reproducing this work can follow these steps to obtain more current data and conduct their own analysis.