

# CS 839 Project Stage 4: Integrating and Performing Analysis

Aishwarya Ganesan

ag@cs.wisc.edu

David Liang

david.liang@wisc.edu

Viswesh Periyasamy

vperiyasamy@wisc.edu

## 1. Merging Schemas

We ran the entity matcher we developed in stage-3 on all the tuple pairs that survived the blocking step. Let  $M$  be the output of the entity matcher; it contains two columns namely,  $a\_id$  and  $w\_id$ , which identify products from Amazon and Walmart respectively. Let  $A$  and  $B$  be the tables that store Amazon and Walmart products respectively.  $A$  and  $B$  have identical schemas. Hence the schema  $E$  obtained by merging the two tables is identical except for the following difference:  $E$  has  $id$  and  $price$  columns for both Amazon and Walmart (as shown in the next section).

We merged the tuples as follows: we always select the value of the tuple as the one from Table  $A$ , unless this value is missing; in case the value from Table  $A$  is missing, we select the value from the tuple in Table  $B$ . We chose the above merging rule because we found the data in the Amazon products table to be cleaner and more complete. We append the code of the Python script (that merges the tables) to the end of this document as an Appendix.

However, once we performed the merging, we had to perform further steps to obtain clean data to perform analysis on.

**Clean numeric values.** For attributes with string values like *hard\_disk\_size*, *ram\_size*, *cpu\_speed*, etc., we had to clean the values so that the values are in the same unit. For example, the values of *hard\_disk\_size* were in GB, TB, KB, and MB in the data set. We cleaned the data so that all values are in GB uniformly. We also removed values that were extraneous or anomalies. For example, we removed those tuples that had processor\_speeds lesser than 100 Mhz or values greater than 5 Ghz.

**Clean string values.** For attributes with string values like *brand*, *operating\_system*, *cpu\_manufacturer*, etc., we performed clustering to obtain a representative name. For example, for the attribute *brand*, we clustered similar values like Dell Commercial, Dell Computers, Dell Consumer, Dell USA, etc., and gave a representative name for the cluster (Dell in case of the above example).

## 2. Merged schema statistics

The merged table E has a schema with the following attributes:

```
a_id
w_id
amazon_price
walmart_price
brand
hard_disk_size
OS
cpu_manufacturer
processor_count
processor_speed
ram_size
screen_size
```

In total there are **4,173 tuples** in the merged table. Below are a few examples of tuples (product title excluded):

a_id	w_id	amazon_price	walmart_price	brand	hard disk size	OS \
a0	w2300	1099.0	1136.09	Acer	256.0	Windows 10
a102	w1746	199.99	244.0	Asus	16.0	Chrome
a1113	w3534	699.0	579.99	Apple	128.0	Mac OS X
a0	w2596	1099.0	997.49	Acer	256.	Windows 10
a1002	w2733	1297.5	1358.27	HP	256.0	Windows 7

cpu	processor count	processor speed	ram size	screen size
Intel	4	3.8	16.0	15.6
Intel	2	1.6	4.0	13.3
Intel	1	1.8	4.0	13.3
Intel	4	3.8	16.0	15.6
Intel	2	2.3	8.0	14.0

### 3. Frequent Item Sets

The first analysis we perform on the above data is to mine the frequent item sets and mine the association rules. `association_rule_mining.py` has the corresponding code.

Following are the frequent item sets that have a minimum support of 0.3 and with set sizes of one and two. More frequent item sets can be found in `data/frequent_item_sets` in stage-4 of our repository.

Table: Frequent Item Sets

Support	Size	Frequent Item Set
0.58	1	brand: Lenovo
0.35	1	hard_disk_size: 512.0 GB
0.88	1	operating_system: Windows 10
0.96	1	cpu_manufacturer: Intel
0.7	1	processor_count: 2
0.31	1	ram_size: 16.0 GB
0.56	1	screen_size: 14.0 in
0.31	1	screen_size: 15.6 in
0.32	2	brand: Lenovo, hard_disk_size: 512.0 GB
0.52	2	brand: Lenovo, operating_system: Windows 10
0.58	2	brand: Lenovo, cpu_manufacturer: Intel
0.53	2	brand: Lenovo, processor_count: 2
0.52	2	brand: Lenovo, screen_size: 14.0 in
0.31	2	hard_disk_size: 512.0 GB, operating_system: Windows 10
0.34	2	hard_disk_size: 512.0 GB, cpu_manufacturer: Intel
0.32	2	hard_disk_size: 512.0 GB, processor_count: 2
0.32	2	hard_disk_size: 512.0 GB, screen_size: 14.0 in
0.84	2	operating_system: Windows 10, cpu_manufacturer: Intel
0.62	2	operating_system: Windows 10, processor_count: 2
0.31	2	operating_system: Windows 10, ram_size: 16.0 GB
0.51	2	operating_system: Windows 10, screen_size: 14.0 in
0.69	2	cpu_manufacturer: Intel, processor_count: 2
0.56	2	cpu_manufacturer: Intel, screen_size: 14.0 in
0.54	2	processor_count: 2, screen_size: 14.0 in

Using the frequent item sets with a minimum support of 0.4, we performed some correlation analysis. We can make some of the following conclusions from

the above analysis. Assuming a minimum support of 0.4, the brand Lenovo is associated with the Windows 10 operating\_system with a confidence of 0.9. Assuming a minimum support of 0.4, laptops of Lenovo brand with the Windows 10 operating\_system with screen\_size of 14.0 inches are most likely to be associated with a processor\_count of 2 with a confidence of 0.99. More such association rules we mined can be found in data/association\_rules in our repository.

## 4. OLAP style analysis

We now perform some olap-style analysis on the merged table described previously. We have a fact table of laptops that gives the product prices in Amazon and Walmart. Some dimensions in this table are brand, operating system, processor (cpu) manufacturer etc. We perform different aggregate queries on the merged table where we group by based on different dimensions like brand, ram\_size etc. The notebook olap.ipynb in our repository has the scripts used in this analysis.

### 4.1. Walmart vs. Amazon Pricing:

**Observation 1: In general, prices are more expensive in Walmart than in Amazon.**

	amazon_price	walmart_price
Mean	1385.5805966930266	1571.5141073568175

**Observation 2: But if we drill down on the prices, using a particular brand, then we find that for a few brands like Apple, Alienware, Asus, Dell, etc., Amazon is pricier.**

### Drilling down by brand:

	amazon_price	walmart_price
brand		
AORUS	2299.000000	2199.000000
Acer	487.122361	509.250694
Alienware	1785.146316	1714.692632
Aorus	2199.000000	2099.000000
Apple	1118.584123	729.805614
Asus	994.911304	864.117681
Bit	214.990000	179.000000
CTL	292.970000	264.810000
Dell	740.719229	692.582207

HP	928.033926	1066.269453
Huawei	689.280000	662.200000
LG	1247.030500	894.477000
Lenovo	1672.351025	2018.848058
MSI	1363.211951	1352.693659
Microsoft	2542.027500	2071.745000
OMEN	3599.000000	3563.950000
Panasonic	1224.791250	728.715000
Prostar	1318.549763	1217.438270
Razer	1604.247500	1790.990000
SLIDE	224.990000	224.990000
Samsung	972.627778	780.921111
Toshiba	1322.960000	1319.400000
VIZIO	979.990000	364.495000

#### 4.2. Analysis by Brand:

We obtain average price by taking a mean of price of the product in Amazon and Walmart. **Observation 3: If we look at the average price range, Lenevo has a lot of products whose price > \$1500**

brand	average_price_range	count
Acer	0-500	87
	1000-1500	8
	1500-2000	2
Apple	500-1000	47
	0-500	1
	1000-1500	27
	1500-2000	6
	2000-2500	2
Dell	500-1000	78
	0-500	71
	1000-1500	52
	1500-2000	8
	2000-2500	2
HP	500-1000	243
	0-500	89
	1000-1500	248
	1500-2000	46
	2000-2500	4

	500-1000	125
Lenovo	0-500	39
	1000-1500	66
	1500-2000	1592
	2000-2500	478
	2500-3000	133
	3000-3500	2
	500-1000	100
Prostar	1000-1500	259
	1500-2000	91
	500-1000	72

**There a lot of Lenevo products in our database.**

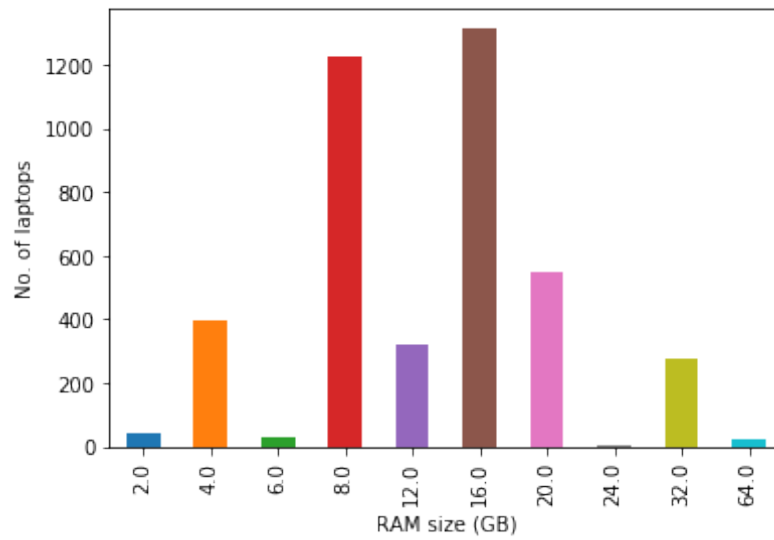
brand	count
Acer	144
Alienware	19
Apple	114
Asus	69
Dell	376
HP	512
LG	20
Lenovo	2410
MSI	41
Prostar	422
Samsung	18

**Observation 4: Most of the Lenevo products in our database have Windows 10 as their operating systems**

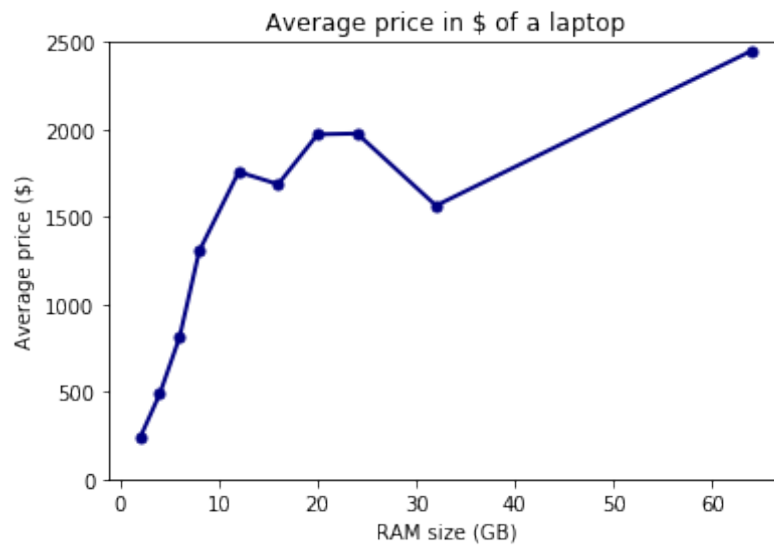
OS	count
Chrome	26
WINDOWS 10	1
Windows 10	2163
Windows 7	215
Windows 8	5

#### **4.3. Analysis by RAM size:**

**Observation 5: RAM sizes of 8GB and 16GB are most popular.**

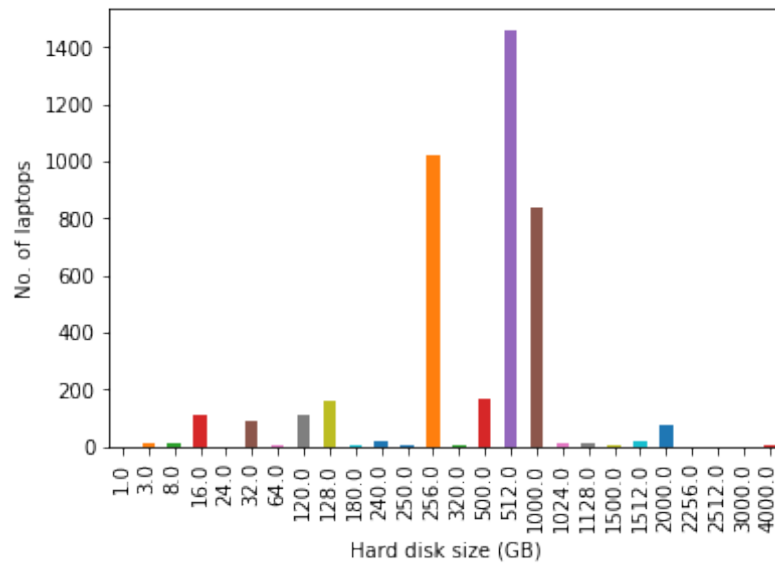


**Observation 6: As the RAM size increases, the average price of a laptop increases.**

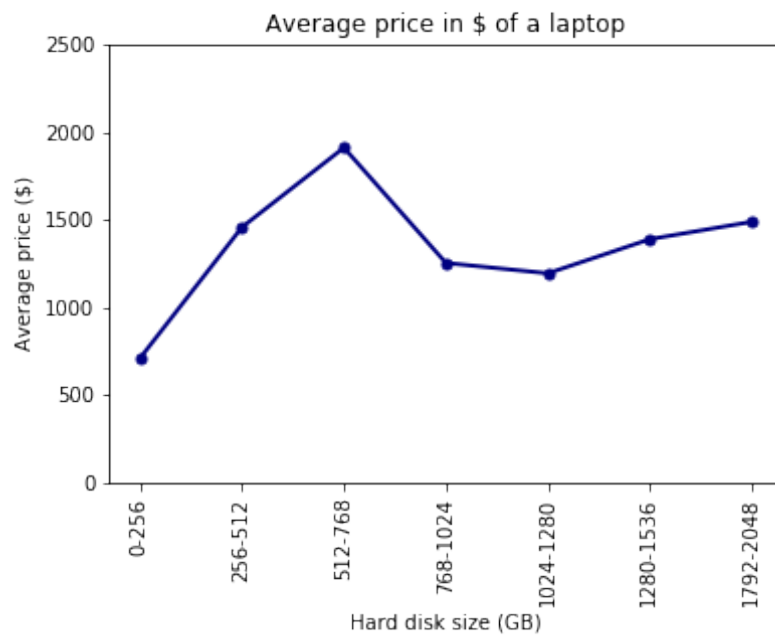


#### 4.4. Analysis by Hard disk size:

**Observation 7: Hard disk sizes of 256GB, 512GB, and 1TB are most popular.**



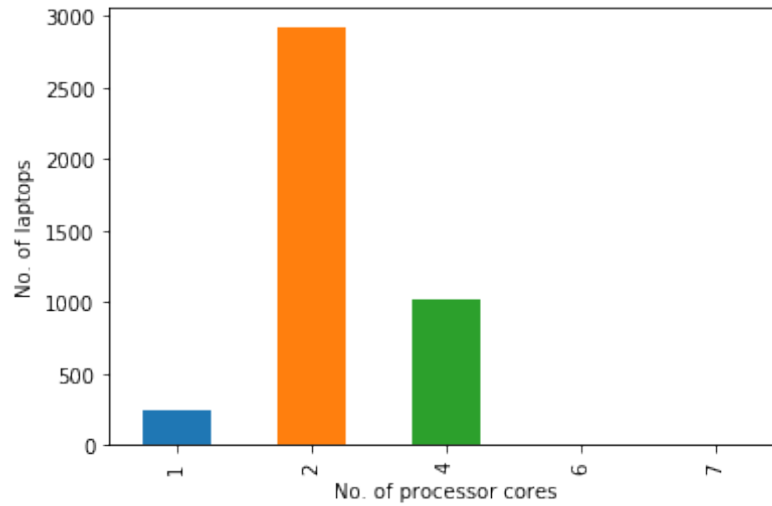
**Observation 8:** There is no obvious trend in the price of a laptop with the increase in hard disk size. A peak at 512GB could be due to the 512GB hard disks being SSDs instead of HDDs.



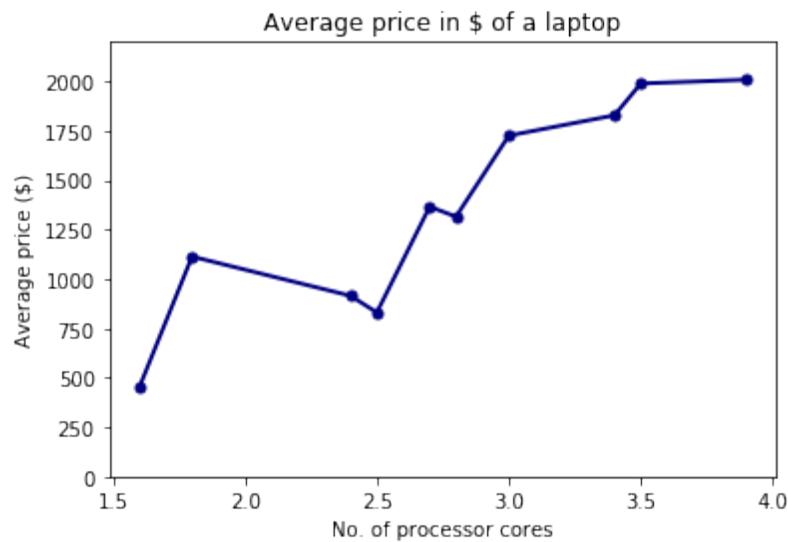


#### 4.5. Analysis by Processor counts and speed:

**Observation 9: Dual core processors are most popular.**



**Observation 10: There is an increase in the price of a laptop with the increase in the processor speed.**



## 5. Price prediction analysis

Having matched entities between Walmart and Amazon, one task we wanted to investigate is whether we could predict the Amazon price and the Walmart

price (multi-label classification) for the same entity given all of the other attributes. Originally we framed this as a regression task, however the data was too sparse and noisy to get price predictions that were remotely close to the actual prices. Specifically, the mean squared error was skyrocketing and upon inspection the predicted prices were several hundreds of dollars off, so we decided to reframe it for classification using price buckets. For example, whether or not the price is between \$0 and \$500, or between \$2500 and \$3000. We then used two classification algorithms well suited for multi-labeled, multi-class data sets: Decision Trees and Random Forests. Overall the process involved three steps: Data Cleaning, Vectorizing Features, and Classification. All code for this section can be found in `predict_prices.py`.

### 5.1. Data Cleaning

The data cleaning process has been described in 1. This data cleaning was especially important for String/categorical attributes which needed to be converted to one-hot encoding for feature vectors as every additional possible value added further and unnecessary dimensionality to our feature vectors.

### 5.2. Vectorizing Features

After cleaning the data thoroughly and manually inspecting it using table viewers such as Excel, we then converted each tuple into a feature vector. We ignored attributes like `w_id` (Walmart ID), `a_id` (Amazon ID), and product title (a large string representing the entire product title) as these were non-informative for the classification task. We then converted all String/categorical attributes to a one-hot binary encoding so that we could input them to the sci-kit learn models. This increased the dimensionality of each feature vector greatly but it was necessary in order to include these features for analysis.

Additionally, we converted the Amazon and Walmart price labels into buckets. We did this using an equal-width binning strategy where each price was digitized into a possible bucket or range of values that it fell into. This turned our regression task into a classification task as we now only had to predict which bucket a product fell into. This completed our conversion of each tuple into a feature vector with only numeric attributes.

### 5.3. Classification

To perform the multi-label multi-classification task, we attempted two binning strategies: even increments of \$200 and even increments of \$500. We created bins from \$0 all the way to the maximum price rounded off evenly (\$6500), resulting in 33 bins for the first strategy and only 13 bins for the second strategy. We then performed 5-fold cross validation using our set of tuples (shuffled) and reported accuracy metrics for each classifier (Decision Tree and Random Forest). The results are displayed in the tables below.

Table 1: Using a binning strategy with bin-width of \$200

Classifier	Retailer	Accuracy	Precision	Recall	F-1
Decision Tree	Amazon	0.872029	0.881569	0.872029	0.876770
Decision Tree	Walmart	0.329978	0.288899	0.329978	0.307264
Random Forest	Amazon	0.873708	0.884511	0.873708	0.879072
Random Forest	Walmart	0.336927	0.264181	0.336927	0.304575

Table 2: Using a binning strategy with bin-width of \$500

Classifier	Retailer	Accuracy	Precision	Recall	F-1
Decision Tree	Amazon	0.942730	0.946339	0.942730	0.944529
Decision Tree	Walmart	0.537027	0.528575	0.537027	0.526495
Random Forest	Amazon	0.943449	0.946596	0.943449	0.945019
Random Forest	Walmart	0.542057	0.531078	0.542057	0.531475

Looking at Table 1, we can see that our classifiers' predictions of Amazon's prices are vastly closer to the actual than compared to Walmart's prices. One reason for this is that when we were consolidating attributes between merged tuples, we tended to prefer Amazon's attribute values as they were much cleaner than Walmart's values - it's possible that these are more indicative of Amazon's pricing as well. If we look at Table 2 where our binning strategy was much more broad, we see a sharp spike of improvement in all metrics for both Walmart and Amazon. However, we still see the lag in predicting Walmart's prices as compared to Amazon's prices.

## 6. Conclusions

With our analysis of Walmart and Amazon prices, we were able to see that Amazon prices were much easier to predict than Walmart prices. This could be a factor of us preferring Amazon attribute values when consolidating matched tuples, or it could be that the laptop prices were less varied on Amazon and therefore much easier to place into bins. As Walmart was much more difficult to predict, it is also possible that their laptop pricing is somewhat non-representative of the product (i.e. they do not price the laptop proportionately based on its attributes but rather to keep in line with their business goals).

If we performed further analysis, one thing we would have to refine is the data cleaning stage. Much of the data was extremely dirty and required several passes of manual rules and replacements, and due to our labor constraints we often had to consolidate a variety of values into one central value. This caused us to lose a lot of valuable details and information that some attributes had to offer. Furthermore, we encountered a lot of missing values which forced us to lose information about those tuples. With more time, we might have used tactics such as Information Extraction to fill in missing values (e.g. if the operating system is missing, go to the product title and extract it from there).

Coupled with these issues is the size of the dataset - it is much too small, and to draw more meaningful conclusions we would need a much larger set to train and test on. Given more time, we would crawl several more products (say tens of thousands) from each website so that we would have an ample supply to sample from.

## Appendix: Code for Merging

The following is code of the Python script that merges the tables.

```
for match in matches:
    if match[0] in amazon_products and match[1] in walmart_products:
        merged_tuple = match[0] + ',' + match[1]
        for i in range(1, len(metadata)):
            meta = metadata[i]
            if meta == 'price':
                merged_tuple += ',' + amazon_products[match[0]][meta]
                merged_tuple += ',' + walmart_products[match[1]][meta]
            else:
                if amazon_products[match[0]][meta] != '':
                    merged_tuple += ',' + amazon_products[match[0]][meta]
                else:
                    merged_tuple += ',' + walmart_products[match[1]][meta]
```

However, we had to perform more steps after merging. The steps and the corresponding script files are as follows : i) clean numeric values (clean\_numeric.py),ii) Clean string values (clean\_strings.py)