# DMDD FINAL PROJECT REPORT

**Project Topic:** University Marketplace Database

**Submission by:** 1. Hrishikesh Sanjay Pawar (NUID: 002707307)
2. Abhishek Nair (NUID: 002724762)

# Table of Contents

# Project description:

When Abhishek and I arrived in Boston for our Master's program, it was an arduous task among 500+ students to find used essential items like study tables, chairs, bed frames, study lamps, etc. University Marketplace Database project is a database of used or pre-owned essential items for peer to peer buying/selling created for university students. The project addresses two issues:

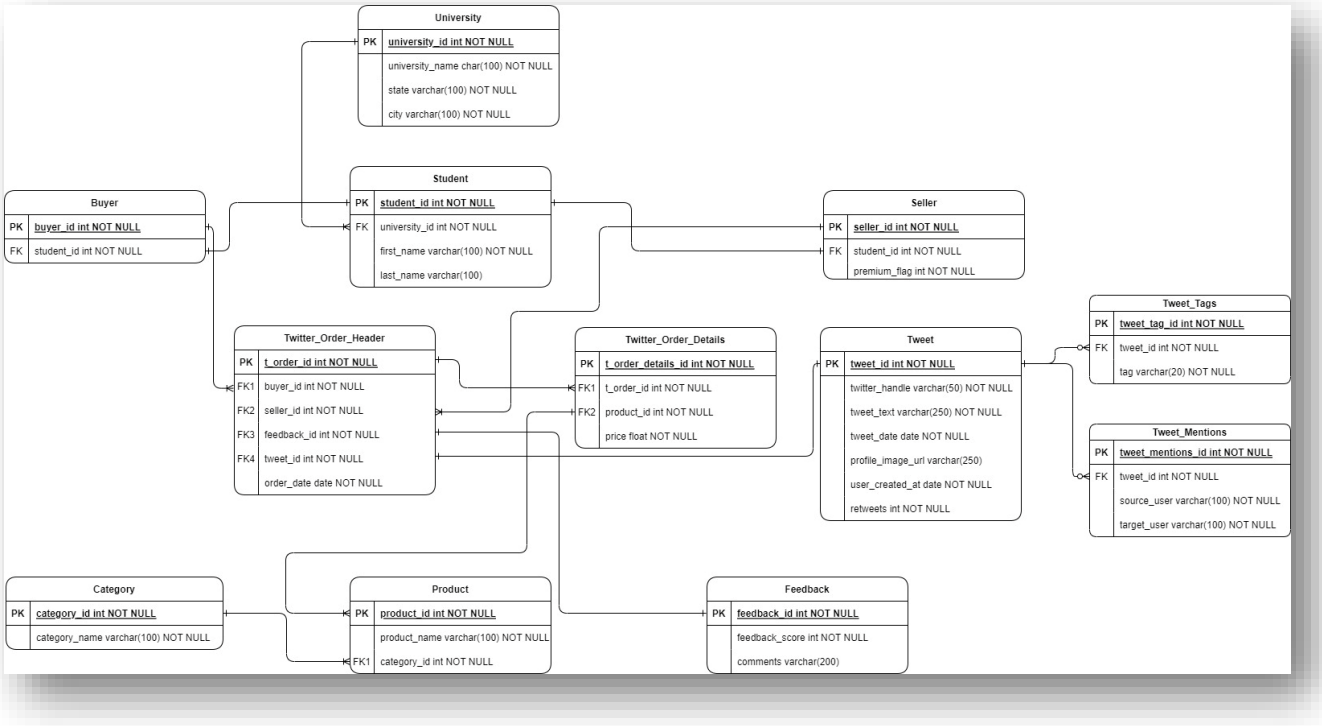
1. Mostly buying/selling of used items is done on instant messengers like Whatsapp via multiple groups. This creates a decentralized environment where it is difficult to keep track of all the buying/selling opportunities.
2. On general platforms like Facebook Marketplace or Ebay, the buyers/sellers might not be trustworthy and/or may be located at an inconvenient distance from the university.

We are trying to resolve these issues by creating a centralized database of pre-owned essential items that can be accessed at a single place. The marketplace will be segregated based on universities to make sure that buyers and sellers are located at a reasonable distance. We will be using users' university email address for the verification purpose. Along with the general details of the product, distance of each product from the buyers will be computed based on the lag-long coordinates. This will help buyers decide which seller to buy from. We are also planning to incorporate a premium subscription feature into the project for sellers. This subscription will enable the sellers to list their products at the top of the search results.

# Project Steps

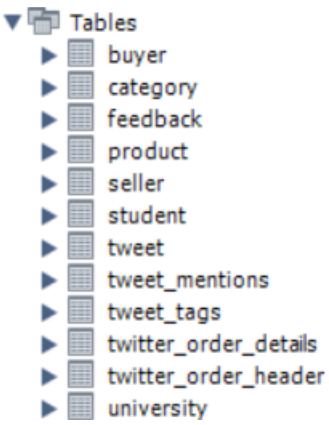
1. We searched for datasets online and found relevant datasets from sources like Kaggle, github, data world, etc.
2. We then extracted data from these sources into Python dataframes and applied cleaning and transformation techniques using pandas.
3. We audited the data for consistency, completeness and validity with the help of visualizations. Python libraries used for visualization are matplotlib and seaborn.
4. We then inserted the data into our database using MSQL library in python.
5. Next we normalized the database such that all the tables meet the requirements to be in the third normal form.
6. Last we created views for all the use cases that we had developed in the past assignments.

# Updated ER Diagram:




This is the ER Diagram that was updated after restructuring the database for normalization to the 3rd normal form.

# SNIPPETS FROM DATABASE SAMPLE TABLES



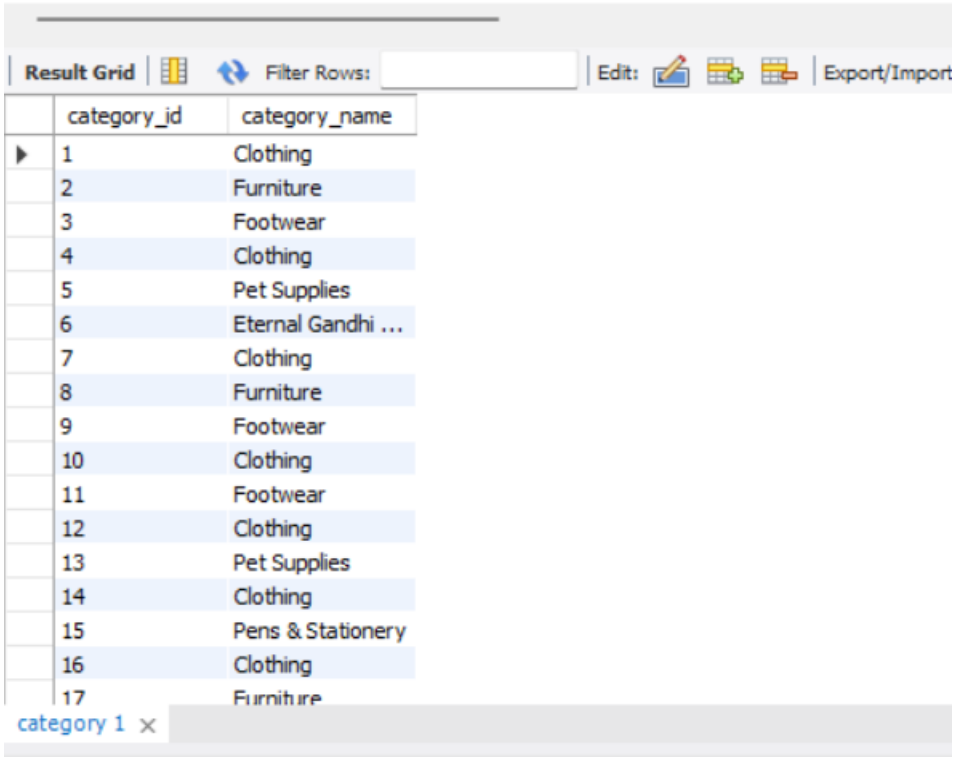

```
1 •   SELECT * FROM twitter_schema.buyer;
```

Result Grid | Filter Rows: | Edit: | Export/Impor

| buyer_id | student_id |
|---|---|
| 49 | 18 |
| 61 | 19 |
| 81 | 29 |
| 7 | 53 |
| 94 | 54 |
| 71 | 61 |
| 82 | 61 |
| 99 | 62 |
| 29 | 79 |
| 50 | 84 |
| 92 | 87 |
| 14 | 95 |
| 28 | 97 |
| 57 | 98 |
| 15 | 108 |
| 9 | 112 |
| 35 | 112 |

buyer 1 ×

```
1 •  SELECT * FROM twitter_schema.category;
```



Result Grid | Filter Rows: | Edit: | Export/Import

| category_id | category_name |
|---|---|
| 1 | Clothing |
| 2 | Furniture |
| 3 | Footwear |
| 4 | Clothing |
| 5 | Pet Supplies |
| 6 | Eternal Gandhi ... |
| 7 | Clothing |
| 8 | Furniture |
| 9 | Footwear |
| 10 | Clothing |
| 11 | Footwear |
| 12 | Clothing |
| 13 | Pet Supplies |
| 14 | Clothing |
| 15 | Pens & Stationery |
| 16 | Clothing |
| 17 | Furniture |

category 1 ×

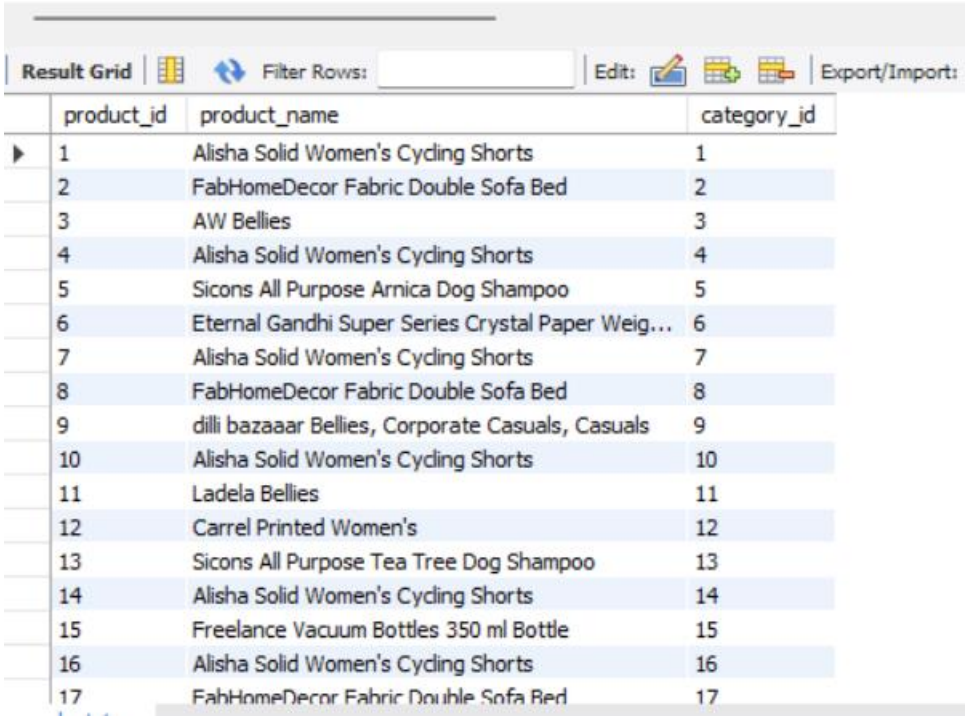
```
1 •  SELECT * FROM twitter_schema.feedback;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

| feedback_id | feedback_score | comments |
|---|---|---|
| 1 | -1 | Worst Product |
| 2 | -1 | Worst Product |
| 3 | -1 | Worst Product |
| 4 | -1 | Worst Product |
| 5 | -1 | Worst Product |
| 6 | -1 | Worst Product |
| 7 | -1 | Worst Product |
| 8 | -1 | Worst Product |
| 9 | -1 | Worst Product |
| 10 | -1 | Worst Product |
| 11 | 5 | Good Product |
| 12 | -1 | Worst Product |
| 13 | -1 | Worst Product |
| 14 | -1 | Worst Product |
| 15 | -1 | Worst Product |
| 16 | -1 | Worst Product |
| 17 | -1 | Worst Product |

feedback 1 ×

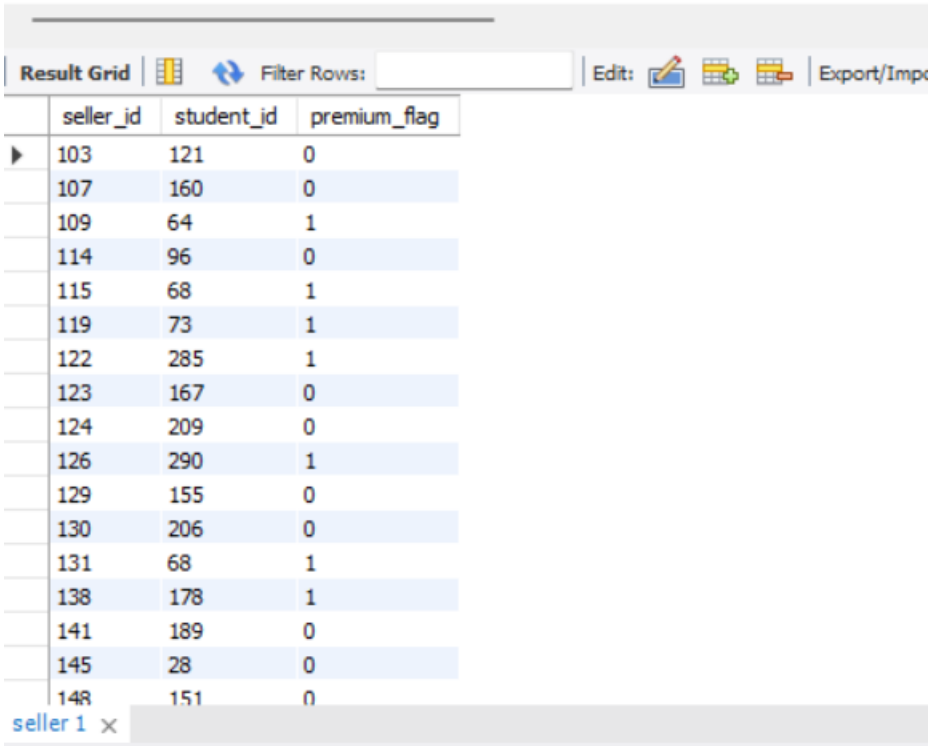
```
1 •    SELECT * FROM twitter_schema.product;
```



Result Grid | Filter Rows: | Edit: | Export/Import:

| product_id | product_name | category_id |
|---|---|---|
| 1 | Alisha Solid Women's Cycling Shorts | 1 |
| 2 | FabHomeDecor Fabric Double Sofa Bed | 2 |
| 3 | AW Bellies | 3 |
| 4 | Alisha Solid Women's Cycling Shorts | 4 |
| 5 | Sicons All Purpose Arnica Dog Shampoo | 5 |
| 6 | Eternal Gandhi Super Series Crystal Paper Weig... | 6 |
| 7 | Alisha Solid Women's Cycling Shorts | 7 |
| 8 | FabHomeDecor Fabric Double Sofa Bed | 8 |
| 9 | dilli bazaaar Bellies, Corporate Casuals, Casuals | 9 |
| 10 | Alisha Solid Women's Cycling Shorts | 10 |
| 11 | Ladela Bellies | 11 |
| 12 | Carrel Printed Women's | 12 |
| 13 | Sicons All Purpose Tea Tree Dog Shampoo | 13 |
| 14 | Alisha Solid Women's Cycling Shorts | 14 |
| 15 | Freelance Vacuum Bottles 350 ml Bottle | 15 |
| 16 | Alisha Solid Women's Cycling Shorts | 16 |
| 17 | FabHomeDecor Fabric Double Sofa Bed | 17 |

```
1 •    SELECT * FROM twitter_schema.seller;
```



Result Grid | Filter Rows: | Edit: | Export/Impo

| seller_id | student_id | premium_flag |
|---|---|---|
| 103 | 121 | 0 |
| 107 | 160 | 0 |
| 109 | 64 | 1 |
| 114 | 96 | 0 |
| 115 | 68 | 1 |
| 119 | 73 | 1 |
| 122 | 285 | 1 |
| 123 | 167 | 0 |
| 124 | 209 | 0 |
| 126 | 290 | 1 |
| 129 | 155 | 0 |
| 130 | 206 | 0 |
| 131 | 68 | 1 |
| 138 | 178 | 1 |
| 141 | 189 | 0 |
| 145 | 28 | 0 |
| 148 | 151 | 0 |

seller 1 ×

```
1 •  SELECT * FROM twitter_schema.student;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

| student_id | university_id | first_name | last_name |
|---|---|---|---|
| 0 | 961 | Kiana | Lor |
| 1 | 2125 | Joshua | Lonaker |
| 2 | 1575 | Dakota | Blanco |
| 3 | 2285 | Natasha | Yarusso |
| 4 | 1575 | Brooke | Cazares |
| 5 | 2530 | Rochelle | Johnson |
| 6 | 967 | Joey | Abreu |
| 7 | 1300 | Preston | Suarez |
| 8 | 2049 | Lee | Dong |
| 9 | 2125 | Maa'iz | al-Dia |
| 10 | 356 | Maja | Nicholson |

student 1 ×

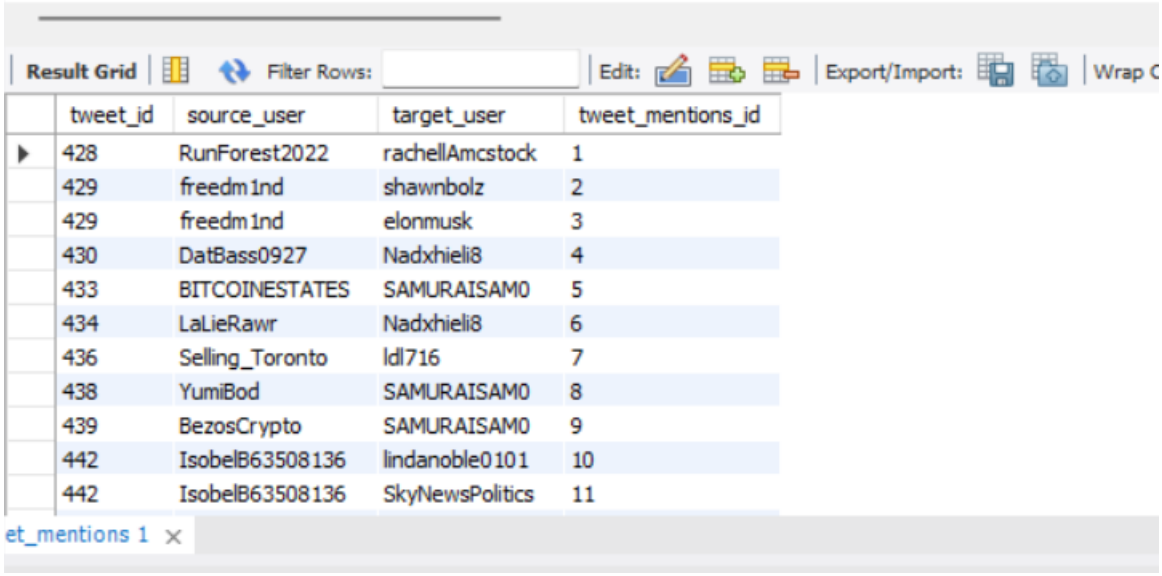
```
1 •  SELECT * FROM twitter_schema.tweet;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| tweet_id | twitter_handle | tweet_text | tweet_date | profile_image_url |
|---|---|---|---|---|
| 428 | RunForest2022 | @rachellAmcstock Have no... | 2022-12-15 | https://pbs.twimg.com/profile_images/1484679... |
| 429 | freedm1nd | @shawnbolz @elonmusk A... | 2022-12-15 | https://pbs.twimg.com/profile_images/1598153... |
| 430 | DatBass0927 | RT @Nadxhieli8: So I'm selli... | 2022-12-15 | https://pbs.twimg.com/profile_images/1245861... |
| 431 | BidadaraBesi | Even if education offers th... | 2022-12-15 | https://pbs.twimg.com/profile_images/1566433... |
| 432 | table_luke | Anyone selling a shit 1x ch... | 2022-12-14 | https://pbs.twimg.com/profile_images/1598297... |
| 433 | BITCOINESTATES | RT @SAMURAISAM0: $BTC... | 2022-12-14 | https://pbs.twimg.com/profile_images/1518697... |
| 434 | LaLieRawr | RT @Nadxhieli8: So I'm selli... | 2022-12-14 | https://pbs.twimg.com/profile_images/1595945... |
| 435 | MustafaSala7 | #wpDataTables is a best-s... | 2022-12-14 | https://pbs.twimg.com/profile_images/1568133... |
| 436 | Selling_Toronto | RT @ldl716: The sacrifice t... | 2022-12-14 | https://pbs.twimg.com/profile_images/1593747... |
| 437 | GraemeMonaghan1 | Any one selling a pool table?? | 2022-12-14 | https://pbs.twimg.com/profile_images/1564652... |

Result Grid | Form Editor | Field Types

```
1 •  SELECT * FROM twitter_schema.tweet_mentions;
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap C

| tweet_id | source_user | target_user | tweet_mentions_id |
|---|---|---|---|
| 428 | RunForest2022 | rachellAmcstock | 1 |
| 429 | freedm1nd | shawnbolz | 2 |
| 429 | freedm1nd | elonmusk | 3 |
| 430 | DatBass0927 | Nadxhieli8 | 4 |
| 433 | BITCOINESTATES | SAMURAISAM0 | 5 |
| 434 | LaLieRawr | Nadxhieli8 | 6 |
| 436 | Selling_Toronto | ldl716 | 7 |
| 438 | YumiBod | SAMURAISAM0 | 8 |
| 439 | BezosCrypto | SAMURAISAM0 | 9 |
| 442 | IsobelB63508136 | lindanoble0101 | 10 |
| 442 | IsobelB63508136 | SkyNewsPolitics | 11 |

et_mentions 1 ×

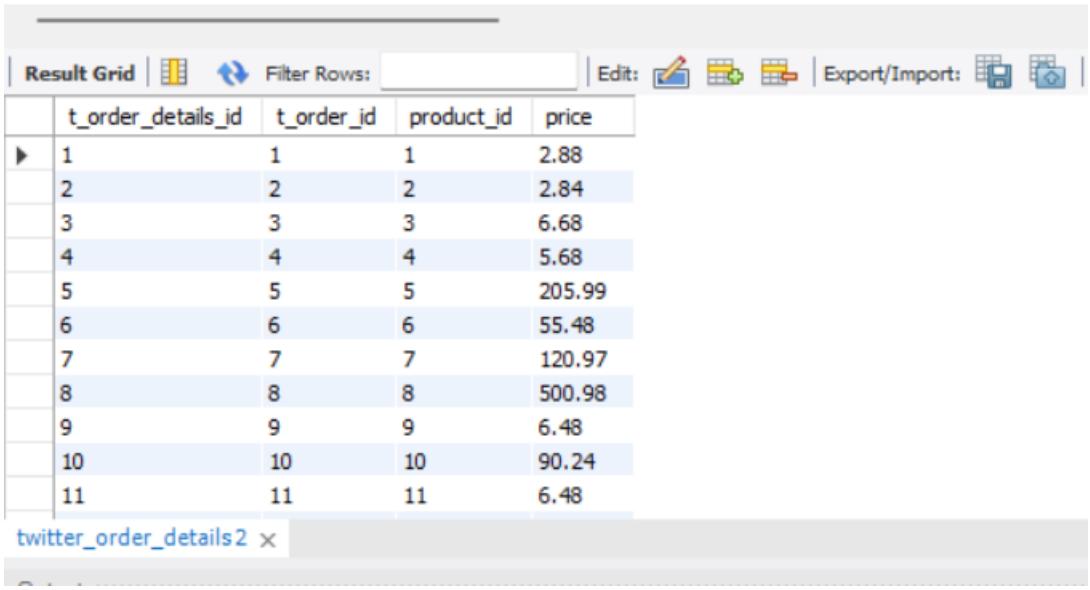
```
1 •   SELECT * FROM twitter_schema.tweet_tags;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

| tweet_id | tag | target_user | tweet_tag_id |
|---|---|---|---|
| 435 | wpDataTables | Nadxhieli8 | 1 |
| 435 | WordPress | Nadxhieli8 | 2 |
| 435 | table | Nadxhieli8 | 3 |
| 435 | plugin | Nadxhieli8 | 4 |
| 435 | charts | Nadxhieli8 | 5 |
| 435 | data | Nadxhieli8 | 6 |
| 454 | ecommerce | DirActionSkills | 7 |
| 456 | ecommerce | pamela25755266 | 8 |
| 513 | comicfiesta2022 | julyhunter07 | 9 |
| NULL | NULL | NULL | NULL |

tweet_tags 1 ×

```
1 •    SELECT * FROM twitter_schema.twitter_order_details;
```



Result Grid | Filter Rows: | Edit: | Export/Import:

| t_order_details_id | t_order_id | product_id | price |
|---|---|---|---|
| 1 | 1 | 1 | 2.88 |
| 2 | 2 | 2 | 2.84 |
| 3 | 3 | 3 | 6.68 |
| 4 | 4 | 4 | 5.68 |
| 5 | 5 | 5 | 205.99 |
| 6 | 6 | 6 | 55.48 |
| 7 | 7 | 7 | 120.97 |
| 8 | 8 | 8 | 500.98 |
| 9 | 9 | 9 | 6.48 |
| 10 | 10 | 10 | 90.24 |
| 11 | 11 | 11 | 6.48 |

twitter_order_details2 ×

```
1 •   SELECT * FROM twitter_schema.twitter_order_header;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell C

| t_order_id | buyer_id | seller_id | tweet_id | feedback_id | order_date |
|---|---|---|---|---|---|
| 1 | 73 | 183 | 1 | 1 | 2012-05-28 |
| 2 | 86 | 114 | 2 | 2 | 2010-07-07 |
| 3 | 73 | 165 | 3 | 3 | 2011-07-27 |
| 4 | 81 | 123 | 4 | 4 | 2011-07-27 |
| 5 | 9 | 158 | 5 | 5 | 2011-07-27 |
| 6 | 99 | 141 | 6 | 6 | 2011-11-09 |
| 7 | 37 | 167 | 7 | 7 | 2013-07-01 |
| 8 | 34 | 168 | 8 | 8 | 2010-12-13 |
| 9 | 12 | 165 | 9 | 9 | 2012-05-12 |
| 10 | 40 | 126 | 10 | 10 | 2011-05-26 |
| 11 | 71 | 183 | 11 | 11 | 2012-12-29 |

rder_header 1 ×

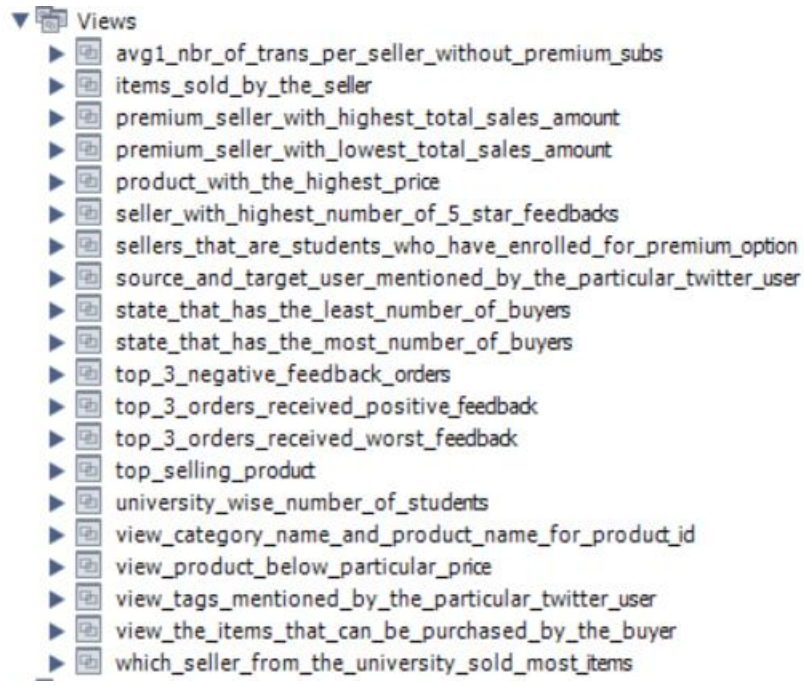
```
1 •   SELECT * FROM twitter_schema.university;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| university_id | university_name | state | city |
|---|---|---|---|
| 138 | HARMONY HEALTH CARE INSTITUTE | NH | MERRIMACK |
| 152 | SKIN SCIENCE INSTITUTE | UT | SANDY |
| 177 | NATIONAL AMERICAN UNIVERSITY-GARDEN CITY | KS | GARDEN CITY |
| 200 | OTTAWA UNIVERSITY-PHOENIX | AZ | PHOENIX |
| 204 | CALIFORNIA WESTERN SCHOOL OF LAW | CA | SAN DIEGO |
| 292 | VOGUE INTERNATIONAL ACADEMY | TX | HOUSTON |
| 308 | HUDSON COUNTY COMMUNITY COLLEGE | NJ | JERSEY CITY |
| 317 | METROPOLITAN COLLEGE OF NEW YORK | NY | NEW YORK |
| 342 | DSDT | MI | DETROIT |
| 352 | WYOTECH | WY | LARAMIE |
| 356 | COLUMBIA CENTRAL UNIVERSITY-CAGUAS | PR | CAGUAS |

university 1 ×  Apply

# Output snippets from Views for Use Cases:




1) Use case: What is the average number of transactions per seller for sellers without premium subscription?

```
345 •   SELECT *
346     FROM twitter_schema.avg1_nbr_of_trans_per_seller_without_premium_subs;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Avg_Transactions_Per_Seller |
|---|
| 9.9310 |

2) Use Case: View the items sold by a seller

```
348 •   SELECT *
349     FROM twitter_schema.items_sold_by_the_seller;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| product_name |
|---|
| Angelfish Silk Potali Potli |
| Nuride Canvas Shoes |
| HRS ULTIMATE BOY Chest Pads |
| Wrangler Skanders Fit Men's Jeans |
| Calibro SW-125 Analog-Digital Watch - For Men... |
| Swag 670038 Analog Watch - For Boys |
| Disney 98189 Analog Watch - For Boys, Girls |
| Ausy Casual Short Sleeve Floral Print Women's ... |
| Urban Fashion Bank Casual, Party, Festive, Lou... |
| Colbrii Casual Full Sleeve Printed Women's Top |

3) Use Case: View premium seller with highest total sales amount

```
351 •   SELECT *
352     FROM twitter_schema.premium_seller_with_highest_total_sales_amount;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| seller_id | Total_Sales |
|---|---|
| 165 | 3926.6098964214325 |

4) Use Case: View the seller that sold the most number of items

```
402 •   SELECT *
403     FROM twitter_schema.which_seller_from_the_university_sold_most_items;
404
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| first_name | last_name | Number_of_items_sold |
|---|---|---|
| Alexander | Sherman | 18 |

5) Use Case: View the product with highest price.

```
357 •   SELECT *
358     FROM twitter_schema.product_with_the_highest_price;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| product_name | Price |
|---|---|
| Quechua Arpenaz Novadry Boots | 6783.02 |

6) Use Case: View the seller with highest number of 5-star feedbacks

```
360 •   SELECT *
361     FROM twitter_schema.seller_with_highest_number_of_5_star_feedbacks;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| seller_id | 5_STAR_TXN_COUNT |
|---|---|
| 125 | 2 |

7) Use Case: View who are the sellers that are students and have they enrolled for a premium option.

```
363 •   SELECT *
364     FROM twitter_schema.sellers_that_are_students_who_have_enrolled_for_premium_option;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| first_name | last_name | premium_flag |
|---|---|---|
| Danielle | Nguyen | 1 |
| Miriam | Aguilar | 1 |
| Sheyenne | Delgado-Manzanares | 1 |
| Jesse | Carballo | 1 |
| Duncan | Kruse | 1 |
| Harper | Wheeler-Marques | 1 |
| Dimitri | Castillo | 1 |
| Abdul | Jabbaar | 1 |
| Joseph | Snider | 1 |
| Bryce | Vaillancourt | 1 |
| Shuraih | el-Karim | 1 |

sellers_that_are_students_who ...

8) Use Case: View the source and the target user mentioned by the particular twitter user in a tweet

```
366 •   SELECT *
367     FROM twitter_schema.source_and_target_user_mentioned_by_the_particular_twitter_user;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| twitter_handle | tweet_text | tweet_date | source_user | target_user |
|---|---|---|---|---|
| YumiBod | RT @SAMURAISAM0: $BTC.X The play right no... | 2022-12-14 | YumiBod | SAMURAISAM0 |
| BezosCrypto | RT @SAMURAISAM0: $BTC.X The play right no... | 2022-12-14 | BezosCrypto | SAMURAISAM0 |
| IsobelB63508136 | RT @lindanoble0101: @SkyNewsPolitics But NO... | 2022-12-14 | IsobelB63508136 | lindanoble0101 |
| IsobelB63508136 | RT @lindanoble0101: @SkyNewsPolitics But NO... | 2022-12-14 | IsobelB63508136 | SkyNewsPolitics |
| TheWifeofNordic | @AlpacaAurelius Eh, not hard for me I'm just la... | 2022-12-14 | TheWifeofNordic | AlpacaAurelius |
| MgsSmu | @keith_s_miller @Patrick_Reusse You have 0%... | 2022-12-14 | MgsSmu | keith_s_miller |
| MgsSmu | @keith_s_miller @Patrick_Reusse You have 0%... | 2022-12-14 | MgsSmu | Patrick_Reusse |
| CiboloCattle | @MCerean Thanks, I appreciate it. We are a fa... | 2022-12-14 | CiboloCattle | MCerean |
| BrennanSzostak2 | RT @JANELABABY: Good news everyone I'm go... | 2022-12-14 | BrennanSzostak2 | JANELABABY |
| DomainsVvs | @DrPaulGosar If the Mexican groups that used ... | 2022-12-14 | DomainsVvs | DrPaulGosar |
| icantmosh | RT @JANELABABY: Good news everyone I'm go... | 2022-12-14 | icantmosh | JANELABABY |

source_and_target_user_menti...

9) Use case: View the state that has least number of buyers

```
369 •   SELECT *
370     FROM twitter_schema.state_that_has_the_least_number_of_buyers;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| State | Buyer_Count |
|---|---|
| UT | 0 |

10) Use case: View the items that are available for purchase by buyers.

```
399 •   SELECT *
400     FROM twitter_schema.view_the_items_that_can_be_purchased_by_the_buyer;
401
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| product_name | category_name |
|---|---|
| Alisha Solid Women's Cycling Shorts | Clothing |
| FabHomeDecor Fabric Double Sofa Bed | Furniture |
| AW Bellies | Footwear |
| Alisha Solid Women's Cycling Shorts | Clothing |
| Sicons All Purpose Arnica Dog Shampoo | Pet Supplies |
| Eternal Gandhi Super Series Crystal Paper Weig... | Eternal Gandhi Super Series Crystal Paper Weig... |
| Alisha Solid Women's Cycling Shorts | Clothing |
| FabHomeDecor Fabric Double Sofa Bed | Furniture |
| dilli bazaaar Bellies, Corporate Casuals, Casuals | Footwear |
| Alisha Solid Women's Cycling Shorts | Clothing |
| Ladela Bellies | Footwear |
| Carrel Printed Women's | Clothing |
| Sicons All Purpose Tea Tree Dog Shampoo | Pet Supplies |
| Alisha Solid Women's Cycling Shorts | Clothing |
| Freelance Vacuum Bottles 350 ml Bottle | Pens & Stationery |
| Alisha Solid Women's Cycling Shorts | Clothing |

view_the_items_that_can_be_p...

11) Use case: View the top 3 orders with negative feedback

```
375 •   SELECT *
376     FROM twitter_schema.top_3_negative_feedback_orders;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| t_order_id | feedback_score |
|---|---|
| 1 | -1 |
| 2 | -1 |
| 3 | -1 |

12) Use case: View tags mentioned by a particular user

```
396 •   SELECT *
397     FROM twitter_schema.view_tags_mentioned_by_the_particular_twitter_user;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| twitter_handle | tweet_text | tweet_date | tag |
|---|---|---|---|
| nabeelaq | RT @DirActionSkills: Meet Mr. Amjad Hussain, a... | 2022-12-14 | ecommerce |
| DirActionSkills | Meet Mr. Amjad Hussain, an Amazon PL Expert ... | 2022-12-14 | ecommerce |
| husxijima | RT @julyhunter07: Hi, I am very fortunate to h... | 2022-12-13 | comicfiesta2022 |

13) Use case: View the top selling product in the system

```
384 •   SELECT *
385     FROM twitter_schema.top_selling_product;
```

Result Grid | Filter Rows: | Export: | Wrap Ce

| product_name | ORDER_CNT |
|---|---|
| Alisha Solid Women's Cycling Shorts | 1 |

14) Use case: View the count of students in each university

```
387 •   SELECT *
388     FROM twitter_schema.university_wise_number_of_students;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| university_name | student_count |
|---|---|
| HARMONY HEALTH CARE INSTITUTE | 2 |
| SKIN SCIENCE INSTITUTE | 3 |
| NATIONAL AMERICAN UNIVERSITY-GARDEN CITY | 5 |
| OTTAWA UNIVERSITY-PHOENIX | 3 |
| VOGUE INTERNATIONAL ACADEMY | 4 |
| HUDSON COUNTY COMMUNITY COLLEGE | 2 |
| METROPOLITAN COLLEGE OF NEW YORK | 3 |
| DSDT | 5 |
| WYOTECH | 4 |
| COLUMBIA CENTRAL UNIVERSITY-CAGUAS | 3 |
| BRYANT & STRATTON COLLEGE-RACINE | 3 |

university_wise_number_of_stu...

15) Use case: View category and product name for a particular product_id

```
390 •   SELECT *
391     FROM twitter_schema.view_category_name_and_product_name_for_product_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| product_name | category_name |
|---|---|
| Alisha Solid Women's Cycling Shorts | Clothing |
| FabHomeDecor Fabric Double Sofa Bed | Furniture |
| AW Bellies | Footwear |
| Alisha Solid Women's Cycling Shorts | Clothing |
| Sicons All Purpose Arnica Dog Shampoo | Pet Supplies |
| Eternal Gandhi Super Series Crystal Paper Weig... | Eternal Gandhi Super Series Crystal Paper Weig... |
| Alisha Solid Women's Cycling Shorts | Clothing |
| FabHomeDecor Fabric Double Sofa Bed | Furniture |
| dilli bazaaar Bellies, Corporate Casuals, Casuals | Footwear |
| Alisha Solid Women's Cycling Shorts | Clothing |
| Ladela Bellies | Footwear |

view_category_name_and_prod...

16) Use case: View products below a particular price (filter)

```
393 •   SELECT *
394     FROM twitter_schema.view_product_below_particular_price;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| product_name |
|---|
| Alisha Solid Women's Cycling Shorts |
| FabHomeDecor Fabric Double Sofa Bed |
| AW Bellies |
| Alisha Solid Women's Cycling Shorts |
| Sicons All Purpose Arnica Dog Shampoo |
| Eternal Gandhi Super Series Crystal Paper Weig... |
| Alisha Solid Women's Cycling Shorts |
| FabHomeDecor Fabric Double Sofa Bed |
| dilli bazaaar Bellies, Corporate Casuals, Casuals |
| Alisha Solid Women's Cycling Shorts |
| Ladela Bellies |
| Carrel Printed Women's |
| Sicons All Purpose Tea Tree Dog Shampoo |
| Alisha Solid Women's Cycling Shorts |

view_product_below_particular...

17) Use case: View the top 3 orders that have a positive feedback

```
SELECT *
FROM twitter_schema.top_3_orders_received_positive_feedback;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| t_order_id |
|---|
| 95 |
| 98 |
| 11 |

18) Use case: View the state having highest number of buyers

```
SELECT *
FROM twitter_schema.state_that_has_the_most_number_of_buyers;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| State | Buyer_Count |
|---|---|
| CA | 5 |

19) Use case: View premium seller with lowest total sales amount

```
SELECT *
FROM twitter_schema.premium_seller_with_lowest_total_sales_amount;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| seller_id | Total_Sales |
|---|---|
| 131 | 50.16999959945679 |

## <u>Note</u> – Some use cases have INSERT statements for which creating views is not possible.

# USE CASE QUERIES AND OUTPUT SNIPPETS

```
102     -- 1) Use Case 1: Top 3 orders which have received the negative feedback
103 •   DROP VIEW twitter_schema.TOP_3_Negative_feedback_orders;
104 •   CREATE VIEW  twitter_schema.TOP_3_Negative_feedback_orders AS
105     SELECT t.t_order_id, f.feedback_score
106     FROM twitter_schema.twitter_order_header t
107     INNER JOIN twitter_schema.Feedback f
108     ON f.feedback_id = t.feedback_id
109     ORDER BY feedback_score ASC
110     LIMIT 3;
111
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

| t_order_id | feedback_score |
|---|---|
| 1 | -1 |
| 2 | -1 |
| 3 | -1 |

```
112     -- 2) Use Case 2: View Category Name and Product Name for a specific product ID
113 •   DROP VIEW twitter_schema.View_Category_Name_and_Product_name_for_product_id;
114 •   CREATE VIEW  twitter_schema.View_Category_Name_and_Product_name_for_product_id AS
115     SELECT p.product_name, c.category_name
116     FROM twitter_schema.Product p
117     INNER JOIN twitter_schema.Category c
118     ON p.category_id = c.category_id;
119
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| product_name | category_name |
|---|---|
| Alisha Solid Women's Cycling Shorts | Clothing |
| FabHomeDecor Fabric Double Sofa Bed | Furniture |
| AW Bellies | Footwear |
| Alisha Solid Women's Cycling Shorts | Clothing |
| Sicons All Purpose Arnica Dog Shampoo | Pet Supplies |
| Eternal Gandhi Super Series Crystal Paper Weig... | Eternal Gandhi ... |
| Alisha Solid Women's Cycling Shorts | Clothing |
| FabHomeDecor Fabric Double Sofa Bed | Furniture |

Result 53 ×

```
120     -- 3) Use Case 3: View tags mentioned by the particular twitter_user in a tweet_text
121 •  DROP VIEW twitter_schema.View_tags_mentioned_by_the_particular_twitter_user;
122 •  CREATE VIEW  twitter_schema.View_tags_mentioned_by_the_particular_twitter_user AS
123     SELECT t.twitter_handle, t.tweet_text, t.tweet_date, tt.tag
124     FROM twitter_schema.Tweet t
125     INNER JOIN twitter_schema.Tweet_tags tt
126     ON t.tweet_id = tt.tweet_id;
127
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| twitter_handle | tweet_text | tweet_date | tag |
|---|---|---|---|
| nabeelaq | RT @DirActionSkills: Meet ... | 2022-12-14 | ecommerce |
| DirActionSkills | Meet Mr. Amjad Hussain, a... | 2022-12-14 | ecommerce |
| husxijima | RT @julyhunter07: Hi, I am... | 2022-12-13 | comicfiesta2022 |

```
128     -- 4) Use Case 4: Who are the sellers that are students and have they enrolled for a premium option?
129 •  DROP VIEW twitter_schema.sellers_that_are_students_who_have_enrolled_for_premium_option;
130 •  CREATE VIEW  twitter_schema.sellers_that_are_students_who_have_enrolled_for_premium_option AS
131     SELECT student.first_name, student.last_name, seller.premium_flag
132     FROM twitter_schema.Student student
133     INNER JOIN twitter_schema.Seller seller
134     ON student.student_id = seller.student_id
135     where seller.premium_flag = 1;
136
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| first_name | last_name | premium_flag |
|---|---|---|
| Danielle | Nguyen | 1 |
| Miriam | Aguilar | 1 |
| Sheyenne | Delgado-Manzanares | 1 |
| Jesse | Carballo | 1 |
| Duncan | Kruse | 1 |
| Harper | Wheeler-Marques | 1 |
| Dimitri | Castillo | 1 |

esult 55 ×

```
137     -- 5) Use Case 5: Who is the source and the target user mentioned by the particular twitter user in a tweet
138 •   DROP VIEW twitter_schema.source_and_target_user_mentioned_by_the_particular_twitter_user;
139 •   CREATE VIEW  twitter_schema.source_and_target_user_mentioned_by_the_particular_twitter_user AS
140     SELECT t.twitter_handle, t.tweet_text, t.tweet_date, tm.source_user, tm.target_user
141     FROM twitter_schema.Tweet t
142     INNER JOIN twitter_schema.Tweet_Mentions tm
143     ON t.tweet_id = tm.tweet_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| twitter_handle | tweet_text | tweet_date | source_user | target_user |
|---|---|---|---|---|
| YumiBod | RT @SAMURAISAM0: $BTC... | 2022-12-14 | YumiBod | SAMURAISAM0 |
| BezosCrypto | RT @SAMURAISAM0: $BTC... | 2022-12-14 | BezosCrypto | SAMURAISAM0 |
| IsobelB63508136 | RT @lindanoble0101: @Sky... | 2022-12-14 | IsobelB63508136 | lindanoble0101 |
| IsobelB63508136 | RT @lindanoble0101: @Sky... | 2022-12-14 | IsobelB63508136 | SkyNewsPolitics |
| TheWifeofNordic | @AlpacaAurelius Eh, not h... | 2022-12-14 | TheWifeofNordic | AlpacaAurelius |
| MgsSmu | @keith_s_miller @Patrick_R... | 2022-12-14 | MgsSmu | keith_s_miller |
| MgsSmu | @keith_s_miller @Patrick_R... | 2022-12-14 | MgsSmu | Patrick_Reusse |
| CiboloCattle | @MCerean Thanks, I appre... | 2022-12-14 | CiboloCattle | MCerean |
| BrennanSzostak2 | RT @JANELABABY: Good n... | 2022-12-14 | BrennanSzostak2 | JANELABABY |
| DomainsVvs | @DrPaulGosar If the Mexic... | 2022-12-14 | DomainsVvs | DrPaulGosar |

Result 56

```
145     -- 6)  Use Case 6: View the items that can be purchased by the buyer
146 •   DROP VIEW twitter_schema.view_the_items_that_can_be_purchased_by_the_buyer;
147 •   CREATE VIEW  twitter_schema.view_the_items_that_can_be_purchased_by_the_buyer AS
148     SELECT p.product_name, c.category_name
149     FROM twitter_schema.Product p
150     INNER JOIN twitter_schema.Category c
151     ON c.category_id = p.category_id;
152
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| product_name | category_name |
|---|---|
| Alisha Solid Women's Cycling Shorts | Clothing |
| FabHomeDecor Fabric Double Sofa Bed | Furniture |
| AW Bellies | Footwear |
| Alisha Solid Women's Cycling Shorts | Clothing |
| Sicons All Purpose Arnica Dog Shampoo | Pet Supplies |
| Eternal Gandhi Super Series Crystal Paper Weig... | Eternal Gandhi ... |
| Alisha Solid Women's Cycling Shorts | Clothing |
| FabHomeDecor Fabric Double Sofa Bed | Furniture |
| dilli bazaaar Bellies, Corporate Casuals, Casuals | Footwear |

Result 57

```sql
-- 7) Use Case 7: Which seller from the University sold the most items
DROP VIEW twitter_schema.which_seller_from_the_university_sold_most_items;
CREATE VIEW  twitter_schema.which_seller_from_the_university_sold_most_items AS
SELECT  a.first_name, a.last_name, t.Number_of_items_sold
FROM
(
		SELECT seller_id, count(product_id) AS `Number_of_items_sold`
		FROM twitter_schema.twitter_order_header oh
		INNER JOIN twitter_schema.twitter_order_details od
		ON oh.t_order_id = od.t_order_id
		GROUP BY seller_id
		ORDER BY 2 desc
		LIMIT 1
) AS t INNER JOIN twitter_schema.Seller b
 ON t.seller_id = b.seller_id
 INNER JOIN twitter_schema.Student a
 ON a.student_id = b.student_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| first_name | last_name | Number_of_items_sold |
|---|---|---|
| Alexander | Sherman | 18 |

```sql
-- 8) Use Case 8: View a product below a particular price
DROP VIEW twitter_schema.view_product_below_particular_price;
CREATE VIEW  twitter_schema.view_product_below_particular_price AS
SELECT p.product_name
FROM twitter_schema.Product p
INNER JOIN twitter_schema.twitter_order_details t
ON p.product_id = t.product_id
WHERE t.price < 5000;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| product_name |
|---|
| Alisha Solid Women's Cycling Shorts |
| FabHomeDecor Fabric Double Sofa Bed |
| AW Bellies |
| Alisha Solid Women's Cycling Shorts |
| Sicons All Purpose Arnica Dog Shampoo |
| Eternal Gandhi Super Series Crystal Paper Weig... |
| Alisha Solid Women's Cycling Shorts |
| FabHomeDecor Fabric Double Sofa Bed |
| dilli bazaaar Bellies, Corporate Casuals, Casuals |
| Alisha Solid Women's Cycling Shorts |

```sql
-- 9) Use Case 9: Top 3 orders which received the worst feedback
DROP VIEW twitter_schema.top_3_orders_received_worst_feedback;
CREATE VIEW  twitter_schema.top_3_orders_received_worst_feedback AS
SELECT t.t_order_id
FROM twitter_schema.twitter_order_header t
INNER JOIN twitter_schema.Feedback f
ON f.feedback_id = t.feedback_id
ORDER BY feedback_score ASC
LIMIT 3;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

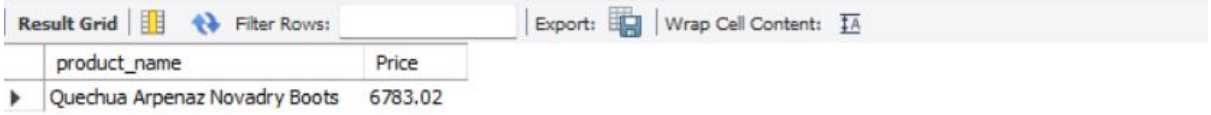
| t_order_id |
|---|
| 1 |
| 2 |
| 3 |

```sql
-- 10) Use Case 10: Top 3 orders which have received the positive feedback
DROP VIEW twitter_schema.top_3_orders_received_positive_feedback;
CREATE VIEW  twitter_schema.top_3_orders_received_positive_feedback AS
SELECT t.t_order_id
FROM twitter_schema.twitter_order_header t
INNER JOIN twitter_schema.Feedback f
ON f.feedback_id = t.feedback_id
ORDER BY feedback_score DESC
LIMIT 3;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

| t_order_id |
|---|
| 95 |
| 98 |
| 11 |

```sql
-- 11) Use Case 11: View the product with highest price.
DROP VIEW twitter_schema.product_with_the_highest_price;
CREATE VIEW twitter_schema.product_with_the_highest_price AS
SELECT product_name, Price
FROM
(SELECT product_id, MAX(price) Price
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
GROUP BY 1
ORDER BY Price Desc
LIMIT 1) O
INNER JOIN
twitter_schema.Product P
ON P.product_id = O.product_id;
```




| product_name | Price |
|---|---|
| Quechua Arpenaz Novadry Boots | 6783.02 |

```
216     -- 12) Use Case 12: View the items sold by a seller
217 •   DROP VIEW twitter_schema.items_sold_by_the_seller;
218 •   CREATE VIEW twitter_schema.items_sold_by_the_seller AS
219     SELECT P.product_name as product_name
220     FROM
221     (SELECT product_id
222     FROM twitter_schema.twitter_order_header oh
223     INNER JOIN twitter_schema.twitter_order_details od
224     ON oh.t_order_id = od.t_order_id
225     WHERE seller_id = 111) O
226     INNER JOIN
227     twitter_schema.Product P
228     ON P.product_id = O.product_id;
229
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| product_name |
|---|
| Angelfish Silk Potali Potli |
| Nuride Canvas Shoes |
| HRS ULTIMATE BOY Chest Pads |
| Wrangler Skanders Fit Men's Jeans |
| Calibro SW-125 Analog-Digital Watch - For Men... |
| Swag 670038 Analog Watch - For Boys |
| Disney 98189 Analog Watch - For Boys, Girls |
| Ausy Casual Short Sleeve Floral Print Women's ... |
| Urban Fashion Bank Casual, Party, Festive, Lou... |
| Colbrii Casual Full Sleeve Printed Women's Top |

```sql
-- 13) Use Case 13: View the top selling product
DROP VIEW twitter_schema.top_selling_product;
CREATE VIEW twitter_schema.top_selling_product AS
SELECT product_name, ORDER_CNT
FROM
(SELECT product_id, COUNT(oh.t_order_id) ORDER_CNT
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
GROUP BY 1
ORDER BY ORDER_CNT DESC
LIMIT 1) O
INNER JOIN
twitter_schema.Product P
ON P.product_id = O.product_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| product_name | ORDER_CNT |
|---|---|
| Alisha Solid Women's Cycling Shorts | 1 |

```sql
-- 14) Use Case 14: View the seller with highest number of 5-star feedbacks
DROP VIEW twitter_schema.seller_with_highest_number_of_5_star_feedbacks;
CREATE VIEW twitter_schema.seller_with_highest_number_of_5_star_feedbacks AS
SELECT seller_id, COUNT(T_order_id) 5_STAR_TXN_COUNT
FROM twitter_schema.twitter_order_header T
LEFT JOIN
twitter_schema.Feedback F
ON F.feedback_id = T.feedback_id
WHERE feedback_score = 5
GROUP BY 1
ORDER BY 5_STAR_TXN_COUNT DESC
LIMIT 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

| seller_id | 5_STAR_TXN_COUNT |
|---|---|
| 125 | 2 |

```sql
-- 15) Use Case 15: What is the average number of transactions per seller for sellers without premium subscription?
DROP VIEW twitter_schema.avg1_nbr_of_trans_per_seller_without_premium_subs;
CREATE VIEW twitter_schema.avg1_nbr_of_trans_per_seller_without_premium_subs AS
SELECT AVG(Order_Count) Avg_Transactions_Per_Seller
FROM
(SELECT T.seller_id as seller_id, COUNT(T_order_id) Order_Count
FROM twitter_schema.twitter_order_header T
INNER JOIN
twitter_schema.Seller S
ON S.seller_id = T.seller_id
WHERE S.premium_flag = 0
GROUP BY 1) D;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Avg_Transactions_Per_Seller |
|---|
| 9.9310 |

```sql
-- 16) Use Case 16: Which State has the most number of buyers?
DROP VIEW twitter_schema.state_that_has_the_most_number_of_buyers;
CREATE VIEW twitter_schema.state_that_has_the_most_number_of_buyers AS
SELECT U.state as State, COUNT(B.buyer_id) Buyer_Count
FROM twitter_schema.University U
LEFT JOIN
twitter_schema.Student S
ON S.university_id = U.university_id
LEFT JOIN
twitter_schema.Buyer B
ON B.student_id = S.student_id
GROUP BY State
ORDER BY Buyer_Count DESC
LIMIT 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

| State | Buyer_Count |
|---|---|
| CA | 5 |

```sql
-- 17) Use Case 17: View premium seller with highest total sales amount
DROP VIEW twitter_schema.premium_seller_with_highest_total_sales_amount;
CREATE VIEW twitter_schema.premium_seller_with_highest_total_sales_amount AS
SELECT oh.seller_id as seller_id, SUM(od.price) as Total_Sales
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
INNER JOIN
twitter_schema.Seller S
ON S.seller_id = oh.seller_id
WHERE premium_flag = 1
GROUP BY seller_id
ORDER BY Total_Sales DESC
LIMIT 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

| seller_id | Total_Sales |
|---|---|
| 165 | 3926.6098964214325 |

```sql
-- 18) Use Case 18: University wise number of students
DROP VIEW twitter_schema.university_wise_number_of_students;
CREATE VIEW twitter_schema.university_wise_number_of_students AS
SELECT university_name, COUNT(student_id) student_count
FROM twitter_schema.University U
INNER JOIN
twitter_schema.Student S
ON S.university_id = U.university_id
GROUP BY 1;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| university_name | student_count |
|---|---|
| HARMONY HEALTH CARE INSTITUTE | 2 |
| SKIN SCIENCE INSTITUTE | 3 |
| NATIONAL AMERICAN UNIVERSITY-GARDEN CITY | 5 |
| OTTAWA UNIVERSITY-PHOENIX | 3 |
| VOGUE INTERNATIONAL ACADEMY | 4 |
| HUDSON COUNTY COMMUNITY COLLEGE | 2 |
| METROPOLITAN COLLEGE OF NEW YORK | 3 |
| DSDT | 5 |
| WYOTECH | 4 |
| COLUMBIA CENTRAL UNIVERSITY-CAGUAS | 3 |
| BRYANT & STRATTON COLLEGE-RACINE | 3 |
| PLATT COLLEGE-MILLER-MOTTE TECHNICAL-C... | 5 |
| TULSA TECHNOLOGY CENTER | 4 |
| VENTURA COUNTY COMMUNITY COLLEGE SYST... | 4 |
| REGIONAL CENTER FOR BORDER HEALTH | 1 |

```sql
-- 19) Use Case 19: View premium seller with lowest total sales amount
DROP VIEW twitter_schema.premium_seller_with_lowest_total_sales_amount;
CREATE VIEW twitter_schema.premium_seller_with_lowest_total_sales_amount AS
SELECT oh.seller_id as seller_id, SUM(od.price) as Total_Sales
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
INNER JOIN
twitter_schema.Seller S
ON S.seller_id = oh.seller_id
WHERE premium_flag = 1
GROUP BY seller_id
ORDER BY Total_Sales ASC
LIMIT 1;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

| seller_id | Total_Sales |
|---|---|
| 131 | 50.16999959945679 |

```sql
-- 20) Use Case 20: Which State has the least number of buyers?
DROP VIEW twitter_schema.state_that_has_the_least_number_of_buyers;
CREATE VIEW twitter_schema.state_that_has_the_least_number_of_buyers AS
SELECT U.state as State, COUNT(B.buyer_id) Buyer_Count
FROM twitter_schema.University U
LEFT JOIN
twitter_schema.Student S
ON S.university_id = U.university_id
LEFT JOIN
twitter_schema.Buyer B
ON B.student_id = S.student_id
GROUP BY State
ORDER BY Buyer_Count ASC
LIMIT 1;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

| State | Buyer_Count |
|---|---|
| UT | 0 |

# UPDATED PYTHON SCRIPT (AFTER NORMALIZATION)

## Scraping Twitter: University Marketplace

Introduction

Here we are using twitter API to extract tweets related to buy and sell of used items in twitter. After extracting all the tweets using the API related to our search value, we will save all the records in our MySQL tables (tweet, tweet_mentions, tweet_tags).

## Importing Essential Libraries

```python
import tweepy
import configparser
import pandas as pd
import pymysql
import matplotlib
pymysql.install_as_MySQLdb()
import mysql.connector
from mysql.connector import Error
```

✓ 5.1s Python

## Authentication keys

Here we are defining keys to authenticate with twitter API and start calling API functions to extract tweets for our analysis.

You need to register for a Twitter dev account https://developer.twitter.com

Look at the Twitter data model https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/object-model/tweet

Apply for a Twitter Developer Account

Go to the Twitter developer site to apply for a developer account.

Step 2: Create an Application

Twitter grants authentication credentials to apps, not accounts. An app can be any tool or bot that uses the Twitter API. So you need to register your an app to be able to make API calls.

To register your app, go to your Twitter apps page and select the Create an app option.

You need to provide the following information about your app and its purpose:

App name: a name to identify your application (such as examplebot) Application description: the purpose of your application (such as An example bot for a Real Python article) Your or your application's website URL: required, but can be your personal site's URL since bots don't need a URL to work Use of the app: how users will use your app (such as This app is a bot that will automatically respond to users) Step 3: Create the Authentication Credentials

To create the authentication credentials, go to your Twitter apps page. Here's what the Apps page looks like:

Edit app details Here you'll find the Details button of your app. Clicking this button takes you to the next page, where you can generate the credentials.

By selecting the Keys and tokens tab, you can generate and copy the key, token, and secrets to use them in your code:

Generate keys and tokens After generating the credentials, save them to later use them in your code.

```python
# read config from config.ini file
config = configparser.ConfigParser()
config.read('config.ini')

api_key = config['twitter']['api_key']
api_key_secret = config['twitter']['api_key_secret']
access_token = config['twitter']['access_token']
access_token_secret = config['twitter']['access_token_secret']
```

[2] ✓ 0.1s Python

## Authentication

As we have previously seen, the Twitter API requires that all requests use OAuth to authenticate. So you need to create the required authentication credentials to be able to use the API. These credentials are four text strings:

Consumer key Consumer secret Access token Access secret

```python
# Using Tweepy to authenticate user using api key and access token
auth = tweepy.OAuthHandler(api_key, api_key_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth,wait_on_rate_limit=True)

# Checking if the api credentials is verified
try:
    api.verify_credentials()
    print("Authentication OK")
except:
    print("Error during authentication")
```

3] ✓ 0.9s Python

```
Authentication OK
```

## Extracting Tweets

We are using search_tweets function to get all the tweets with keyword 'selling a table'

```python
# Searching tweets based on the text message
tweets = api.search_tweets('selling a table',count=10000)

# Checking the database connection
try:
    connection = mysql.connector.connect(host='localhost',
                                         database='twitter_schema',
                                         user='root',
                                         password='root')
    if connection.is_connected():
        db_Info = connection.get_server_info()
        print("Connected to MySQL Server version ", db_Info)
        cursor = connection.cursor()
        cursor.execute("select database();")
        record = cursor.fetchone()
        print("You're connected to database: ", record)
except Error as e:
    print("Error while connecting to MySQL", e)


```

20] ✓ 0.6s Python

```
Connected to MySQL Server version  8.0.31
You're connected to database:  ('twitter_schema',)
```

## Loading the tweets into the MySQL database

```python
# Looping over the entire tweets to fetch the required information and inserting the values in three twitter tables: tweet, tweet_mentions, tweet_

tweet_id_list = []
tweet_retweet_count_list = []
tweet_mentions_id_index = 1
tweet_tags_id_index = 1
for tweet in tweets:
    tweet_id = tweet.id
    created_at = tweet.created_at
    twitter_text = tweet.text
    username = tweet.user.screen_name
    name = tweet.user.name
    userId = tweet.user.id
    follower_count = tweet.user.followers_count
    following_count = tweet.user.friends_count
    twitter_handle = tweet.user.screen_name
    profile_image_url = tweet.user.profile_image_url_https
    description = tweet.user.description
    userCreated_at = tweet.user.created_at
    status = api.get_status(tweet_id)
    retweet_count = status.retweet_count

    tweet_id_list.append(tweet_id)
    tweet_retweet_count_list.append(retweet_count)

    cursor.execute('''insert into tweet ( twitter_handle, tweet_text, profile_image_url, tweet_date, user_created_at, retweets) values ( %s, %s, %
    connection.commit()
    tweet_id_index = cursor._last_insert_id
```

```python
    tweet_id_list.append(tweet_id)
    tweet_retweet_count_list.append(retweet_count)

    cursor.execute('''insert into tweet ( twitter_handle, tweet_text, profile_image_url, tweet_date, user_created_at, retweets) values ( %s, %s, %
    connection.commit()
    tweet_id_index = cursor._last_insert_id
    if(len(tweet.entities['user_mentions']) > 0):
        for mention in tweet.entities['user_mentions']:
            target_user = mention['screen_name']
            cursor.execute('''insert into tweet_mentions (tweet_id,source_user, target_user,tweet_mentions_id) values ( %s,%s, %s, %s)''', (tweet_
            connection.commit()
            tweet_mentions_id_index = tweet_mentions_id_index + 1
    if(len(tweet.entities['hashtags']) > 0):
        for tag in tweet.entities['hashtags']:
            tag = tag['text']
            cursor.execute('''insert into tweet_tags (tweet_id,tag, target_user,tweet_tag_id) values (%s,%s, %s, %s)''', (tweet_id_index,tag,targ
            connection.commit()
            tweet_tags_id_index = tweet_tags_id_index + 1
```

22.3s Python

## Audit Completeness and Audit Consistency

```python
import pandas as pd

# Source for flipkart csv dataset from kaggle: https://www.kaggle.com/datasets/PromptCloudHQ/flipkart-products
# reading csv files
flipkart_csv_df = pd.read_csv('C:\\Personal_Documents\\Assignments\\DMDD\\data_sources\\flipkart_com-ecommerce_sample.csv',delimiter = ',')

# Source for orders csv from github: https://github.com/pawarbi/datasets/blob/master/Orders.csv
# reading csv files
order_df = pd.read_csv('C:\\Personal_Documents\\Assignments\\DMDD\\data_sources\\orders.csv',delimiter = ',')



#Code to clean the data by removing all the na values, drop all the unwanted columns and removing all the duplicates
flipkart_csv_df.drop(columns = [flipkart_csv_df.columns[0],flipkart_csv_df.columns[1],flipkart_csv_df.columns[2],flipkart_csv_df.columns[5],flipk
flipkart_csv_df.dropna()
flipkart_csv_df.drop_duplicates(subset=['product_name'])

#Code to clean the data by removing all the na values, drop all the unwanted columns and removing all the duplicates. We have also used rename fu
order_df.drop(columns = [order_df.columns[0],order_df.columns[1],order_df.columns[2],order_df.columns[4],order_df.columns[5],order_df.columns[6],
order_df.rename(columns={"Unit Price": "price", "Order Date": "order_date"}, inplace=True)
order_df.dropna()
order_df.drop_duplicates()
```

[9] 0.9s Python

## Code block to insert records in the Category table

We have written the code to insert the records in the category table.

```python
category_id = 1
product_category_list = []
for index,data in flipkart_csv_df.iterrows():
    if index == 500:
        break
    product_category_tree_df = data['product_category_tree']
    #To fetch only the root value of the product_category_tree using the below logic
    startString = '["'
    endString = ' >>'
    data['product_category_tree'] = product_category_tree_df[product_category_tree_df.find(startString)+len(startString):product_category_tree_df
    product_category_list.append(data['product_category_tree'])
    cursor.execute('''insert into category (category_id,category_name) values (%s,%s)''', (category_id,data["product_category_tree"]))
    category_id = category_id + 1
    connection.commit()
```

[10] ✓ 1.4s Python

## Code block to insert records in the Product table

We have written the code to insert the records in the product table.

```python
product_id = 1
category_id = 1
for index,data in flipkart_csv_df.iterrows():
    if index == 500:
        break
    product_name_df = data['product_name']
    data['product_name'] = product_name_df[:100]
    cursor.execute('''insert into product (product_id,product_name,category_id) values (%s,%s,%s)''', (product_id, data["product_name"],category_
    connection.commit()
    product_id = product_id + 1
    category_id = category_id + 1
```

[11] ✓ 1.3s Python

## Code block to insert records in the Feedback table

We have written the code to insert the records in the feedback table.

```python
feedback_id = 1
for index,data in flipkart_csv_df.iterrows():
    if index == 500:
        break
# We will use proucting rating to generate value for the comments column. If the rating is not available we will assign -1 score.
    product_rating_df = data['product_rating']
    if product_rating_df == 'No rating available':
        comments = 'No comments for the product'
        feedback_score = -1
    else:
        feedback_score = round(float(product_rating_df.strip()))


    rounded_feedback_score = feedback_score

# If the feedback score is less than 3, we will assign comment as 'Worst Product' else we will assing comment as 'Good Product'
    if rounded_feedback_score < 3:
        comments = 'Worst Product'
    else:
        comments = 'Good Product'

    cursor.execute('''insert into feedback (feedback_id,feedback_score,comments) values (%s,%s,%s)''', (feedback_id, feedback_score,comments))
    connection.commit()
    feedback_id = feedback_id + 1
```

[12] ✓ 1.4s Python

## Code to read us-colleges-and-universities csv and student.csv in dataframe. We have also cleaned and standardized the data by removing all the na and duplicate values.

```python
import random

# Source for university dataset: https://public.opendatasoft.com/explore/dataset/us-colleges-and-universities/export/
# reading csv files
university = pd.read_csv('C:\\Personal_Documents\\Assignments\\DMDD\\data_sources\\us-colleges-and-universities.csv', sep=",")

# formatting university data
university = university[['OBJECTID', 'NAME', 'STATE', 'CITY']]
university = university.rename(columns={"OBJECTID": "university_id", "NAME": "university_name", "CITY": "city", "STATE": "state"})

#print(university.head(10))

# Source for student dataset: https://github.com/ShapeLab/ZooidsCompositePhysicalizations/blob/master/Zooid_Vis/bin/data/student-dataset.csv
# reading csv files
student = pd.read_csv('C:\\Personal_Documents\\Assignments\\DMDD\\data_sources\\student.csv', sep=",")

# formatting student data
student = student[['id', 'name']]
student[['first_name', 'last_name', 'temp']] = student.name.str.split(" ", expand=True)
student = student.drop('temp', axis=1)
student = student.rename(columns={"id": "student_id"})
student.dropna()

# generating random university ids for student
univ_id = []
for i in range(len(student)):
    univ_id.append(random.choice(university['university_id']))
```

```python
# adding university_id to student
student['university_id'] = ''
student['university_id'] = univ_id

#Populating buyer
buyer_id = random.sample(range(3, 100), 50)
stud_id = []

for i in range(50):
    stud_id.append(random.choice(student['student_id']))

temp_data = {'buyer_id': buyer_id,
        'student_id': stud_id}

buyer = pd.DataFrame(temp_data)

#Populating seller
seller_id = random.sample(range(103, 200), 50)
premium_flag = []
stud_id = []

for i in range(50):
    premium_flag.append(random.randint(0,1))

for i in range(50):
    stud_id.append(random.choice(student['student_id']))
```

```python
temp_data = {'buyer_id': buyer_id,
        'student_id': stud_id}

buyer = pd.DataFrame(temp_data)

#Populating seller
seller_id = random.sample(range(103, 200), 50)
premium_flag = []
stud_id = []

for i in range(50):
    premium_flag.append(random.randint(0,1))

for i in range(50):
    stud_id.append(random.choice(student['student_id']))

temp_data = {'seller_id': seller_id,
            'student_id': stud_id,
            'premium_flag':premium_flag}

seller = pd.DataFrame(temp_data)
seller.dropna()
```

[13] ✓ 0.2s Python

## Code block to insert records in the University table

We have written the code to insert the records in the university table.

```
#Inserting data into the database
# university_id = 1
for uni in university.iterrows():
    university_id = uni[1][0]
    university_name = uni[1][1]
    state = uni[1][2]
    city = uni[1][3]

    cursor.execute(
        '''insert into university (university_id, university_name, state, city) values ( %s, %s, %s, %s);''',
        (university_id, university_name, state, city))
    # university_id = university_id + 1
    connection.commit()
```

[14] ✓ 0.3s Python

## Code block to insert records in the Student table

We have written the code to insert the records in the student table.

```
for stud in student.iterrows():
    student_id = stud[1][0]
    university_id = stud[1][4]
    first_name = stud[1][2]
    last_name = stud[1][3]
    cursor.execute(
        '''insert into student (student_id, university_id, first_name, last_name) values ( %s, %s, %s, %s);''',
        (student_id, university_id, first_name, last_name))
    connection.commit()
```

[15] ✓ 1.5s Python

## Code block to insert records in the Buyer table

We have written the code to insert the records in the buyer table.

```
for b in buyer.iterrows():
    student_id = int(b[1][1])
    buyer_id = int(b[1][0])
    cursor.execute(
        '''insert into buyer (buyer_id, student_id) values ( %s, %s);''',
        (buyer_id, student_id))
    connection.commit()
```

[16] ✓ 0.1s Python

## Code block to insert records in the Seller table

We have written the code to insert the records in the seller table.

```
for s in seller.iterrows():
    student_id = int(s[1][1])
    seller_id = int(s[1][0])
    premium_flag = int(s[1][2])

    cursor.execute(
        '''insert into seller (seller_id, student_id,premium_flag) values ( %s, %s, %s);''',
        (seller_id, student_id, premium_flag))
    connection.commit()
```

[17] ✓ 0.1s Python

## Code block to insert records in the Tweet Order table

We have written the code to insert the records in the tweet_order table.

```python
t_order_id = 1
tweet_id = 1
product_id = 1
feedback_id = 1
t_order_details_id = 1

seller_id = []
buyer_id = []
price = []
seller_id_1 = []
product_id_1 = []
for i in range(len(order_df)):
    seller_id.append(random.choice(seller['seller_id']))

for i in range(len(order_df)):
    buyer_id.append(random.choice(buyer['buyer_id']))

temp_data = {
        'seller_id' : seller_id,
        'buyer_id' : buyer_id}

main_df = pd.DataFrame(temp_data)

result = pd.concat([main_df, order_df], axis=1)

```

```python
for index,data in result.iterrows():
    if index == 500:
        break
    seller_id_1.append(data['seller_id'])
    product_id_1.append(product_id)
    price.append(data['price'])
    d = data['order_date'].replace('/','-')
    data['order_date'] = d.split("-")[2]+"-"+d.split("-")[0]+"-"+d.split("-")[1]
    cursor.execute('''insert into twitter_order_header(t_order_id,buyer_id,seller_id,tweet_id,feedback_id,order_date) values (%s,%s,%s,%s,%s,%s)'
    connection.commit()
    cursor.execute('''insert into twitter_order_details(t_order_details_id, t_order_id,product_id, price) values (%s,%s,%s,%s)''', (t_order_detai
    connection.commit()
    tweet_id = tweet_id + 1
    t_order_id = t_order_id + 1
    t_order_details_id = t_order_details_id + 1
    product_id = product_id + 1
    feedback_id = feedback_id + 1
```

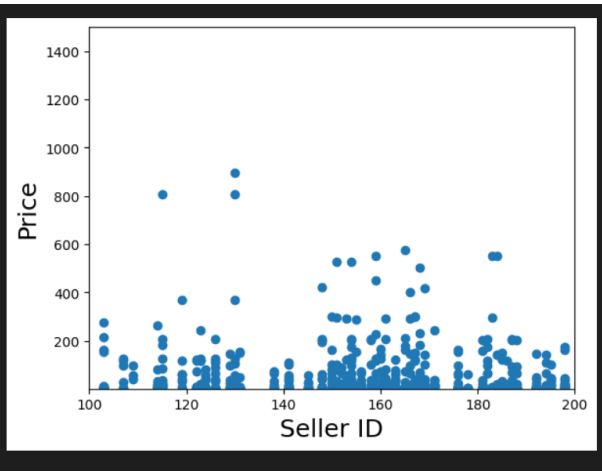
[21] ✓ 2.6s Python

This scatterplot represents a distribution of prices at which items were sold by Seller IDs. This indicates that the data is valid.

```python
import matplotlib.pyplot as plt


plt.xlim([100, 200])
plt.ylim([1, 1500])
plt.xlabel('Seller ID', fontsize=18)
plt.ylabel('Price', fontsize=18)

plt.scatter(seller_id_1,price)


```

[22] ✓ 1.3s Python

```
<matplotlib.collections.PathCollection at 0x21b252a6f80>
```

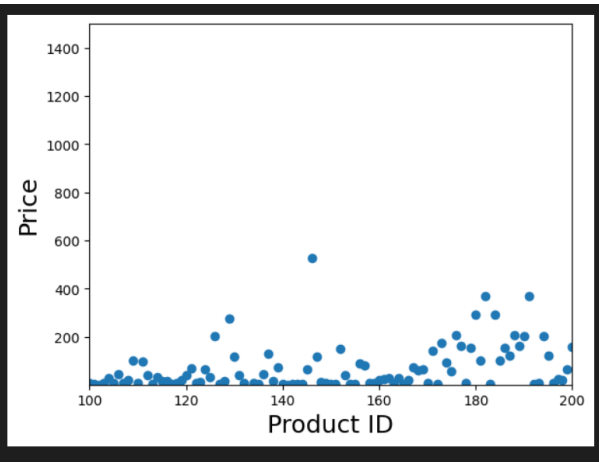



This scatterplot represents a distribution of prices at which products were sold in the system. This indicates that the data is valid.

```python
import matplotlib.pyplot as plt


plt.xlim([100, 200])
plt.ylim([1, 1500])
plt.xlabel('Product ID', fontsize=18)
plt.ylabel('Price', fontsize=18)


plt.scatter(product_id_1,price)

```

[23] ✓ 0.2s Python

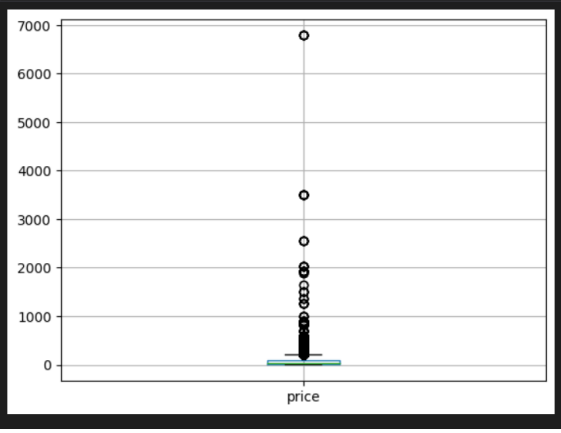
```
<matplotlib.collections.PathCollection at 0x21b24eebd30>
```




This box represents a univariate analysis of prica variable to identify whether the data is uniformly distributed or not. The boxplot indicates that the all the price data points are concentrated close to the lower limit. There are certain data points that are outliers but such outliers are present due to the presence of high value products.

```python
boxplot = order_df.boxplot(column=['price'])

# This box represents a univariate analysis of prica variable to identify whether the data is uniformly
# distributed or not.
# The boxplot indicates that the all the price data points are concentrated close to the lower limit.
# There are certain data points that are outliers but such outliers are present due to the presence of high value products.
```

[24] ✓ 0.2s Python

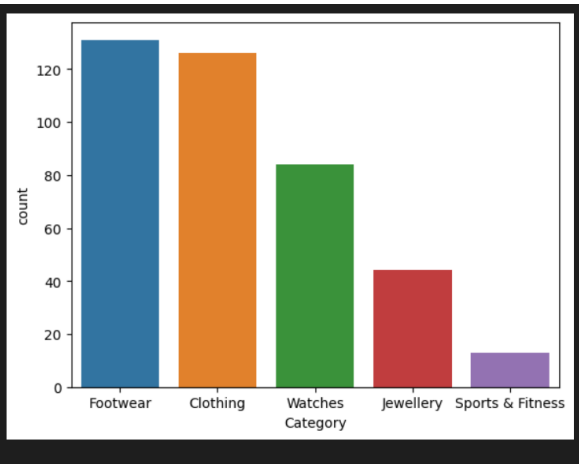



```python
import seaborn as sns
df2 = pd.DataFrame(product_category_list)
df2.columns =['Category']
sns.countplot(x = 'Category',
              data = df2,
              order = df2['Category'].value_counts().head(5).index)
```

[25] ✓ 0.3s Python

```
<AxesSubplot: xlabel='Category', ylabel='count'>
```

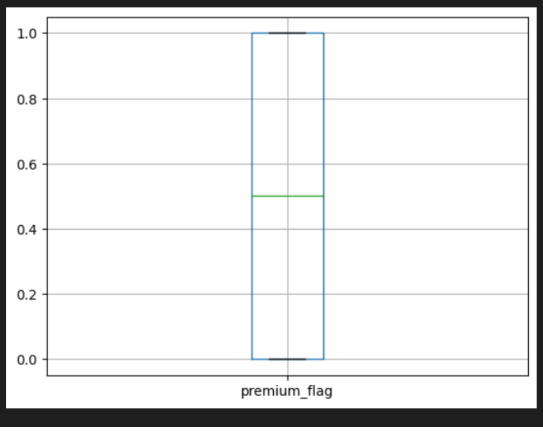
This bar graph represents category wise count of products in the system. Data accuracy can be validated using this graph by comparing the count with the source data.



This boxplot shows the uniform and consistent distribution of premium flag variable. That is because all the data points lie between the inter quartile range.

```
[26]  ✓ 0.2s                                                    Python
```

```
boxplot = seller.boxplot(column=['premium_flag'])

```



# SQL FOR DATABASE CREATION (AFTER NORMALIZATION)

1. SQL creating a new table for twitter_order_header:

```
CREATE TABLE `twitter_schema`.`twitter_order_header` (
  `t_order_id` INT NOT NULL,
  `buyer_id` INT NOT NULL,
  `seller_id` INT NOT NULL,
  `tweet_id` INT NOT NULL,
  `feedback_id` INT NOT NULL,
  `order_date` DATE NOT NULL,
  PRIMARY KEY (`t_order_id`));
```

2. SQL creating a new table for twitter_order_details:

```
CREATE TABLE `twitter_schema`.`twitter_order_details` (
  `t_order_details_id` INT NOT NULL,
  `t_order_id` INT NOT NULL,
  `product_id` INT NOT NULL,
  `price` FLOAT NOT NULL,
  PRIMARY KEY (`t_order_details_id`),
  INDEX `t_order_id_fk_idx` (`t_order_id` ASC) VISIBLE,
  CONSTRAINT `t_order_id_fk`
    FOREIGN KEY (`t_order_id`)
    REFERENCES `twitter_schema`.`twitter_order_header` (`t_order_id`));
```

SQL creating rest of the normalized database:

```
CREATE TABLE `buyer` (
  `buyer_id` int NOT NULL,
  `student_id` int NOT NULL,
  PRIMARY KEY (`buyer_id`),
  KEY `student_id_fk_idx` (`student_id`),
  CONSTRAINT `student_id_fk` FOREIGN KEY (`student_id`) REFERENCES `student` (`student_id`));
```

```
CREATE TABLE `category` (
  `category_id` int NOT NULL,
  `category_name` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`category_id`));
```

```
CREATE TABLE `feedback` (
  `feedback_id` int NOT NULL,
```

```
  `feedback_score` int NOT NULL,
  `comments` varchar(200) DEFAULT NULL,
  PRIMARY KEY (`feedback_id`));


CREATE TABLE `product` (
  `product_id` int NOT NULL,
  `product_name` varchar(100) NOT NULL,
  `category_id` int DEFAULT NULL,
  PRIMARY KEY (`product_id`),
  KEY `category_fk_idx` (`category_id`),
  CONSTRAINT `category_id_fk` FOREIGN KEY (`category_id`) REFERENCES
`category` (`category_id`));


CREATE TABLE `seller` (
  `seller_id` int NOT NULL,
  `student_id` int NOT NULL,
  `premium_flag` int NOT NULL,
  PRIMARY KEY (`seller_id`),
  KEY `student_id_fk_idx` (`student_id`),
  CONSTRAINT `student_id_fk_seller` FOREIGN KEY (`student_id`) REFERENCES
`student` (`student_id`));


CREATE TABLE `student` (
```

```
  `student_id` int NOT NULL,
  `university_id` int NOT NULL,
  `first_name` varchar(100) NOT NULL,
  `last_name` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`student_id`),
  KEY `university_id_fk_idx` (`university_id`),
  CONSTRAINT `university_id_fk` FOREIGN KEY (`university_id`) REFERENCES
`university` (`university_id`));



CREATE TABLE `tweet` (
  `tweet_id` int NOT NULL AUTO_INCREMENT,
  `twitter_handle` varchar(50) NOT NULL,
  `tweet_text` varchar(250) NOT NULL,
  `tweet_date` date NOT NULL,
  `profile_image_url` varchar(300) DEFAULT NULL,
  `user_created_at` date NOT NULL,
  `retweets` int NOT NULL,
  PRIMARY KEY (`tweet_id`));



CREATE TABLE `tweet_mentions` (
  `tweet_id` int NOT NULL AUTO_INCREMENT,
  `source_user` varchar(100) NOT NULL,
```

```
  `target_user` varchar(100) NOT NULL,

  `tweet_mentions_id` int NOT NULL,

  PRIMARY KEY (`tweet_mentions_id`),

  KEY `tweet_id_fk_mentions_idx` (`tweet_id`),

  CONSTRAINT `tweet_mentions_fk` FOREIGN KEY (`tweet_id`) REFERENCES `tweet`
(`tweet_id`));
```

```
CREATE TABLE `tweet_tags` (

  `tweet_id` int NOT NULL AUTO_INCREMENT,

  `tag` varchar(20) NOT NULL,

  `target_user` varchar(100) NOT NULL,

  `tweet_tag_id` int NOT NULL,

  PRIMARY KEY (`tweet_tag_id`),

  KEY `twitter_id_fk_tags_idx` (`tweet_id`),

  CONSTRAINT `tweet_id_fk_tags` FOREIGN KEY (`tweet_id`) REFERENCES `tweet`
(`tweet_id`));
```

```
CREATE TABLE `university` (

  `university_id` int NOT NULL,

  `university_name` varchar(100) NOT NULL,

  `state` varchar(100) NOT NULL,

  `city` varchar(100) NOT NULL,

  PRIMARY KEY (`university_id`));
```

# SQL FOR CREATION OF VIEWS FOR ALL THE USE CASE QUERIES

1) Use Case 1: Top 3 orders which have received the negative feedback

```
CREATE VIEW  twitter_schema.TOP_3_Negative_feedback_orders AS

SELECT t.t_order_id, f.feedback_score

FROM twitter_schema.twitter_order_header t

INNER JOIN twitter_schema.Feedback f

ON f.feedback_id = t.feedback_id

ORDER BY feedback_score ASC

LIMIT 3;
```

2) Use Case 2: View Category Name and Product Name for a specific product ID

```
CREATE VIEW  twitter_schema.View_Category_Name_and_Product_name_for_product_id AS

SELECT p.product_name, c.category_name

FROM twitter_schema.Product p

INNER JOIN twitter_schema.Category c

ON p.category_id = c.category_id;
```

3) Use Case 3: View tags mentioned by the particular twitter_user in a tweet_text

```
CREATE VIEW  twitter_schema.View_tags_mentioned_by_the_particular_twitter_user AS

SELECT t.twitter_handle, t.tweet_text, t.tweet_date, tt.tag

FROM twitter_schema.Tweet t

INNER JOIN twitter_schema.Tweet_tags tt

ON t.tweet_id = tt.tweet_id;
```

4) Use Case 4: Who are the sellers that are students and have they enrolled for a premium option?

```
CREATE VIEW
twitter_schema.sellers_that_are_students_who_have_enrolled_for_premium_option AS

SELECT student.first_name, student.last_name, seller.premium_flag

FROM twitter_schema.Student student

INNER JOIN twitter_schema.Seller seller

ON student.student_id = seller.student_id

where seller.premium_flag = 1;
```

5) Use Case 5: Who is the source and the target user mentioned by the particular twitter user in a tweet

```
CREATE VIEW
twitter_schema.source_and_target_user_mentioned_by_the_particular_twitter_user AS

SELECT t.twitter_handle, t.tweet_text, t.tweet_date, tm.source_user, tm.target_user

FROM twitter_schema.Tweet t

INNER JOIN twitter_schema.Tweet_Mentions tm

ON t.tweet_id = tm.tweet_id;
```

6) Use Case 6: View the items that can be purchased by the buyer

```
CREATE VIEW  twitter_schema.view_the_items_that_can_be_purchased_by_the_buyer AS

SELECT p.product_name, c.category_name

FROM twitter_schema.Product p

INNER JOIN twitter_schema.Category c

ON c.category_id = p.category_id;
```

7) Use Case 7: Which seller from the University sold the most items

```
CREATE VIEW  twitter_schema.which_seller_from_the_university_sold_most_items AS
SELECT  a.first_name, a.last_name, t.Number_of_items_sold
FROM
(
        SELECT seller_id, count(product_id) AS `Number_of_items_sold`
        FROM twitter_schema.twitter_order_header oh
        INNER JOIN twitter_schema.twitter_order_details od
        ON oh.t_order_id = od.t_order_id
        GROUP BY seller_id
        ORDER BY 2 desc
        LIMIT 1
) AS t INNER JOIN twitter_schema.Seller b
 ON t.seller_id = b.seller_id
 INNER JOIN twitter_schema.Student a
 ON a.student_id = b.student_id;
```

8) Use Case 8: View a product below a particular price

```
CREATE VIEW  twitter_schema.view_product_below_particular_price AS
SELECT p.product_name
FROM twitter_schema.Product p
```

```
INNER JOIN twitter_schema.twitter_order_details t
ON p.product_id = t.product_id
WHERE t.price < 5000;
```

9) Use Case 9: Top 3 orders which received the worst feedback

```
CREATE VIEW  twitter_schema.top_3_orders_received_worst_feedback AS
SELECT t.t_order_id
FROM twitter_schema.twitter_order_header t
INNER JOIN twitter_schema.Feedback f
ON f.feedback_id = t.feedback_id
ORDER BY feedback_score ASC
LIMIT 3;
```

10) Use Case 10: Top 3 orders which have received the positive feedback

```
CREATE VIEW  twitter_schema.top_3_orders_received_positive_feedback AS
SELECT t.t_order_id
FROM twitter_schema.twitter_order_header t
INNER JOIN twitter_schema.Feedback f
ON f.feedback_id = t.feedback_id
ORDER BY feedback_score DESC
LIMIT 3;
```

11) Use Case 11: View the product with highest price.

```
CREATE VIEW twitter_schema.product_with_the_highest_price AS
SELECT product_name, Price
FROM
(SELECT product_id, MAX(price) Price
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
GROUP BY 1
ORDER BY Price Desc
LIMIT 1) O
INNER JOIN
twitter_schema.Product P
ON P.product_id = O.product_id;
```

12) Use Case 12: View the items sold by a seller

```
CREATE VIEW twitter_schema.items_sold_by_the_seller AS
```

```
SELECT P.product_name as product_name
FROM
(SELECT product_id
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
WHERE seller_id = 1) O
INNER JOIN
twitter_schema.Product P
ON P.product_id = O.product_id;
```

13) Use Case 13: View the top selling product

```
CREATE VIEW twitter_schema.top_selling_product AS
SELECT product_name, ORDER_CNT
FROM
(SELECT product_id, COUNT(oh.t_order_id) ORDER_CNT
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
GROUP BY 1
ORDER BY ORDER_CNT DESC
LIMIT 1) O
INNER JOIN
twitter_schema.Product P
ON P.product_id = O.product_id;
```

14) Use Case 14: View the seller with highest number of 5-star feedbacks

```
CREATE VIEW twitter_schema.seller_with_highest_number_of_5_star_feedbacks AS

SELECT seller_id, COUNT(T_order_id) 5_STAR_TXN_COUNT

FROM twitter_schema.twitter_order_header T

LEFT JOIN

twitter_schema.Feedback F

ON F.feedback_id = T.feedback_id

WHERE feedback_score = 5

GROUP BY 1

ORDER BY 5_STAR_TXN_COUNT DESC

LIMIT 1;
```

15) Use Case 15: What is the average number of transactions per seller for sellers without premium subscription?

```
CREATE VIEW twitter_schema.avg1_nbr_of_trans_per_seller_without_premium_subs AS

SELECT AVG(Order_Count) Avg_Transactions_Per_Seller

FROM

(SELECT T.seller_id as seller_id, COUNT(T_order_id) Order_Count

FROM twitter_schema.twitter_order_header T

INNER JOIN

twitter_schema.Seller S

ON S.seller_id = T.seller_id

WHERE S.premium_flag = 0

GROUP BY 1) D;
```

16) Use Case 16: Which State has the most number of buyers?

```
CREATE VIEW twitter_schema.state_that_has_the_most_number_of_buyers AS
SELECT U.state as State, COUNT(B.buyer_id) Buyer_Count
FROM twitter_schema.University U
LEFT JOIN
twitter_schema.Student S
ON S.university_id = U.university_id
LEFT JOIN
twitter_schema.Buyer B
ON B.student_id = S.student_id
GROUP BY State
ORDER BY Buyer_Count DESC
LIMIT 1;
```

17) Use Case 17: View premium seller with highest total sales amount

```
CREATE VIEW twitter_schema.premium_seller_with_highest_total_sales_amount AS
SELECT oh.seller_id as seller_id, SUM(od.price) as Total_Sales
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
INNER JOIN
twitter_schema.Seller S
ON S.seller_id = oh.seller_id
WHERE premium_flag = 1
```

```
GROUP BY seller_id
ORDER BY Total_Sales DESC
LIMIT 1;
```

18) Use Case 18: University wise number of students

```
CREATE VIEW twitter_schema.university_wise_number_of_students AS
SELECT university_name, COUNT(student_id) student_count
FROM twitter_schema.University U
INNER JOIN
twitter_schema.Student S
ON S.university_id = U.university_id
GROUP BY 1;
```

19) Use Case 19: View premium seller with lowest total sales amount

```
CREATE VIEW twitter_schema.premium_seller_with_lowest_total_sales_amount AS
SELECT oh.seller_id as seller_id, SUM(od.price) as Total_Sales
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
INNER JOIN
twitter_schema.Seller S
```

```
ON S.seller_id = oh.seller_id
WHERE premium_flag = 1
GROUP BY seller_id
ORDER BY Total_Sales ASC
LIMIT 1;
```

20) Use Case 20: Which State has the least number of buyers?

```
CREATE VIEW twitter_schema.state_that_has_the_least_number_of_buyers AS
SELECT U.state as State, COUNT(B.buyer_id) Buyer_Count
FROM twitter_schema.University U
LEFT JOIN
twitter_schema.Student S
ON S.university_id = U.university_id
LEFT JOIN
twitter_schema.Buyer B
ON B.student_id = S.student_id
GROUP BY State
ORDER BY Buyer_Count ASC
LIMIT 1;
```