IT UNIVERSITY OF COPENHAGEN

Big Data Management – Assignment 1

Adam Hadou Temsamani (ahad@itu.dk)

Big Data Management
Course code: KSBIDMT1KU
Course manager: Zoi Kaoudi (zoka@itu.dk)

Github Repository: `github.itu.dk/Big-Data-Management-2025/ahad_a1`
Hand-in date: 24/10/2025

Contents

1 Specific DataFrame Queries

Task 3.1.1

```
reviews.count()
```

Listing 1: Find the total number of reviews for all businesses.

Metric	Count
Total Reviews	6,990,280

Table 1: Output for Task 3.1.1

The query returns approximately 7 million reviews. Since every row in the `review` Dataframe has a business id i.e the business' being reviewed. Counting the rows provides the total number of reviews for all business'.

Task 3.1.2

```
five_stars_business = business.filter((business.stars == 5) & (
    business.review_count >= 750))\
                        .select( name ,  stars ,  review_count )
five_stars_business.show()
```

Listing 2: Find all businesses that have received 5 stars and that have been reviewed by 750 or more users.

Name	Stars	Review Count
Blues City Deli	5.0	991
Carlillos Cocina	5.0	799
Free Tours By Foot	5.0	769

Table 2: Businesses with 5 stars and $\geq$ 750 or more reviews.

The query identified only three business that meet the strict criteria of having a perfect rating of 5.0, while also having a high volume reviews. The results are displayed in Table 2 above. The scarcity of volumes suggest that maintaining a perfect score becomes more difficult as number of costumers increase.

The business dataframe has a `stars` and `review_count` column. As such we can `.filter` on business' who fit the criteria. The exercise asks for dataframe of `(name, stars, review count)`. so we `.select` those columns.

Task 3.1.3

```
influencers = users.filter((users.review_count > 750) & (users.
    average_stars > 4))\
                .select( user_id )
```

Listing 3: Find influencers who have written more than 750 reviews. And have an average rating higher than 4 stars.

Sample User IDs
MGPQVLsODMm9ZtYQW...
RgDVC3ZUBqpEe6Y1k...
VHdY6oG2JPVNjihWh...
UQFE3BT1rsIYrcDvu...
Total Influencers Found: 1,028

Table 3: Sample of Users meeting Influencer criteria

We define "influencers" as users who have written more than 750 reviews and maintain an average rating higher than 4 stars. The query returns a total of 1,028 influencers. This represents the group of highly active users who consistently leave positive feedback on the platform. However, without the total number of users with 750 reviews, we cannot how much the subset constitutes.

The users dataframe has a `review_count` and `average_stars` column. So we simply `.filter` on the users based on our predicate. The output should only be the user id of the influencers, so we `.select` on those.

Task 3.1.4

```
influencers = users.filter((users.review_count > 750) & (users.
    average_stars > 4))\
                .select( user_id )

review_influencers = reviews.join(influencers, on= user_id , how=
    inner ) \
                        .withColumnRenamed( stars ,  review_stars ) \
                        .select( business_id ,  user_id ,
                            review_stars )

business_review_influencers = business.join(review_influencers, on=
    business_id , how= inner )

business_influence_rating = business_review_influencers.groupBy(
    business_id ,  name )\
```

```
                    .agg(F.countDistinct( user_id ).alias(
                        influencer_count ), F.avg( review_stars ).alias
                        ( average_rating ))

business_review_by_five = business_influence_rating.filter(
    business_influence_rating.influencer_count > 5)

business_review_by_five.select( name ,  average_rating ).show()
```

Listing 4: Find the businesses names and their average ratings for ones that have been reviewed by more than 5 influencer users.

Business Name	**Influencer Avg Rating**
Ryman Auditorium	4.74
Homage	4.58
Tampa Premium Outlets	4.54
The Original Tony's	4.32
Adventure Aquarium	3.67

Table 4: Sample of businesses reviewed by > 5 influencers

To identify business popular among influential users (more than 5), I perform a multi-step join operation. First, we join the `influencers` view (defined in the previous task) with the `reviews` dataframe, to get only relevant reviews (written by influencers). This result is then joined with the `business` dataframe to attach business names.

I group the data by business' and aggregate the results to calculate the distinct count of influences and their average rating. Finally, filtering based on the previously defined predicate (business with more than 5 influencer reviews). This yields 2,298 results.

As we can see in the Table above, while many of the top results are highly rated, even popular locations among influencers such as "Adventure Aquarium" received mixes scores. Suggesting that the ratings vary significantly, and that influencers do not simply award high ratings blindly.

Task 3.1.5

```
user_avg_stars = reviews.groupBy( user_id ) \
                    .agg(F.avg(F.col( stars )).alias( avg_stars ), F.
                        count( * ).alias( review_count ))

user_avg_name = user_avg_stars.join(users.select( user_id ,  name ),
    on= user_id , how= inner ) # I don't join on all users, as I only
    want their names

```

```
user_avg_ordered = user_avg_name.orderBy(F.desc( avg_stars ),F.desc(
    review_count )) # I am assuming that I also need to all order it by
     their reivews (as secondary), as users with more reviews are more
    important

user_avg_ordered.show()
```

Listing 5: Find an ordered list of users based on the average star counts they have given in all their reviews.

User Name	Avg Stars	Review Count
Carolyn	5.0	231
Pinkie	5.0	214
Eve	5.0	138
Timothy	5.0	91
Nicole Caridad	5.0	89
Tom	5.0	70
Denise	5.0	64
Paul	5.0	63

Table 5: Top Users by Average Star Rating and Review Count

To identify the most consistently positive users, we group the reviews by `user_id` and calculate two aggregates: the average star rating and total count of reviews. We then join the intermediate result with the `users` dataframe to retrieve usernames.

Finally, we order the results by first average star (descending) and secondarily by review count (descending). The query prioritizes users who maintain high rating over a large volume of reviews, thereby filtering out users with only a single 5-star review.

As shown in the table above, the top users have maintained a perfect 5.0 average despite having written hundreds of reviews. This reveals the existence of "super-positive" power users. While these users are highly active, their ratings make it hard to distinguish between "good" and "exceptional" business' compared to users who utilize the full rating spectrum. However, it is worth noting that there exists 1987897 users in the total dataset. So these users would only represent an extreme outlier group.

2 Authenticity Study

Task 3.2.1: Data Exploration

This section of the assignment is about *authenticity* language used in the data.

Question 1: Percentage contain "authentic" and not "inauthentic"

```
# What is the percentage of reviews that contain a variant of the word
    authentic , and do not contain  inauthentic .

# Let's start by defining authentic and inauthentic words (i.e.
    synonyms):
authentic_words = [ authentic ,  genuine ,  legit ,  real ,
    legitimate ,  valid ,  original ,  credible ]
inauthentic_words = [ inauthentic ,  unauthentic ,  fake ,  sham ,
    ungenuine ,  unreal ,  invalid ,  unorginal ,  illigitimate ]

# Create a helper regex pattern (case-insensitive)
authentic_pattern = r (?i)(?<!not\s)(?<!no\s)\b(  +  | .join(
    authentic_words) + r )\w*\b
inauthentic_pattern = r (?i)\b(  +  | .join(inauthentic_words) + r )\w
    *\b

# Include authentic and exclude the inauthentic
authentic_reviews = reviews.filter(
    F.col( text ).rlike(authentic_pattern) &
    (~F.col( text ).rlike(inauthentic_pattern))
)

# Get percentage of reviews
total_reviews = reviews.count()
total_authentic_reviews = authentic_reviews.count()
percentage = total_authentic_reviews / total_reviews * 100

print( Total review count: , total_reviews)
print( Total authentic reviews count , total_authentic_reviews)
print( Percentage of reviews including authentic and excluding
    inauthentic: , percentage)
```

Listing 6: What is the percentage of reviews that contain a variant of the word "authentic", and do not contain "inauthentic".

To establish a baseline for the study, I define two lists of key words: `authentic_words` (positive synonyms) and `inauthentic_words` (negative synonyms).

I construct a case-insensitive Regex patterns to filter the dataset. A negation filter is used

Metric	Count / Percentage
Total Reviews	6,990,280
Authentic Reviews	1,648,844
Authenticity Percentage	**23.59%**

Table 6: Prevalence of Authenticity Language

to ensure that reviews contain "inauthentic" terms are excluded. This ensures that our query only captures reviews that are being positive (use of authentic words), rather than negative.

Crucially, a negative lookbehind was also implemented to strictly exclude negated contexts (e.g., "not authentic" or "no authentic"). This ensures that our query filters out false positives and captures only reviews that use authenticity terms affirmativel

Finally using the views `total_authentic_reviews` and `total_reviews`, we compute the percentage.

Findings. As shown in the Table over, approximately 22% of all reviews contain positive authenticity language. This figure is substantial, representing nearly one in four reviews across the entire dataset. Suggesting, that "authenticity" is not a niche descriptor, but used by everyday users. The high prevalence of these terms, makes subsequent analysis of *how* this language varies by cuisine (the *authenticity trap*) meaningful.

2.0.1 Question 2: Contain "genuine" grouped by business type

```
reviews_with_genuine = reviews.filter(col( text ).rlike( (?i)genuine )
    )

#Explode business.categories into business_type
business_exploded = business.select( business_id ,  categories )\
            .withColumn( business_type , F.explode(F.split(F.col(
                categories ),  ,  )))

# Join filtered reviews with  business (due to business type)
rg_with_business = reviews_with_genuine.join(business_exploded.select(
    business_id ,  business_type ), on= business_id , how= inner )

# Group by business_type and count reviews
rg_by_type = rg_with_business.groupBy( business_type ) \
    .agg(F.count( * ).alias( genuine_count )) \
    .orderBy(F.desc( genuine_count ))
```

Listing 7: How many reviews contain the string "genuine" grouped by businesses type?

Business Type	Count of "Genuine"
Restaurants	18,417
Food	7,529
Nightlife	5,891
Bars	5,561
American (New)	4,032
American (Traditional)	3,739
Shopping	3,626
Breakfast & Brunch	3,563

Table 7: Top Business Types associated with the word "Genuine"

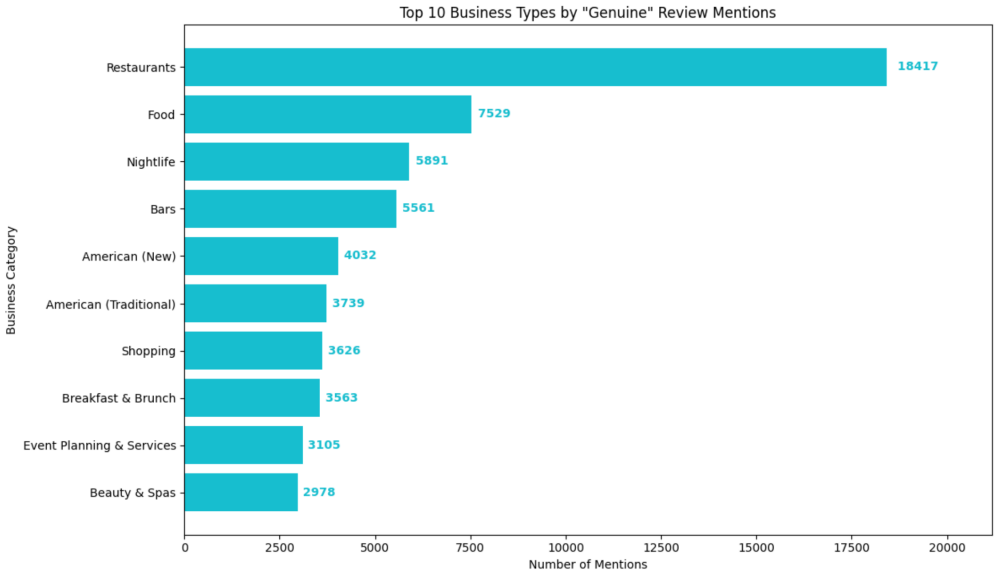

Figure 1: Top Business Types associated with the word "Genuine"

To analyze the usage of the word "genuine", we first filter reviews for containing the term. A critical challenge in the dataset is that, businesses often belong to multiple categories. To address this; i used the `explode` function to split the comma separated `categories` into a new column `business_type`. This makes it easier to accurately count a single review towards every category the business belongs to. The exploded business type is joined with reviews to aggregate the counts.

Findings: As shown in the Table above, the term "genuine" is dominantly associated with dining business (Restaurants and Food). The high ranking of Nightlife and Bars offers a potentially interesting insight: users may use the term "genuine" to value attributes other than the physical products being served (the food or drink itself).

The dominant dominance within dining, validates the study's specific focus on restaurant reviews, confirming this as the primary domain where the concept of authenticity is used.

2.0.2 Question 3: "legitimate" versus "illegitimate"

```
resturants = business.select( business_id ,  name ,  categories ).
    filter(business.categories.rlike( (?i)restaurant ))
resturants_exploded = resturants.withColumn( cuisine_type , F.explode(
    F.split(F.col( categories ),  ,  ))).select( business_id ,  name ,
     cuisine_type )

# Long list of cuisines by region:
# European Cuisine
european_cuisine = [ Italian ,  French ,  Spanish ,  Greek ,
    Portuguese ,  German ,  Dutch ,  Belgian ,  Swiss ,  Austrian ,
    Hungarian ,  Polish ,  Czech ,  Slovak ,  Danish ,  Swedish ,
    Norwegian ,  Finnish ,  Icelandic ]
european_pattern =  (?i)(  +  | .join(european_cuisine) +  )
european_cuisines = resturants_exploded.filter(col( cuisine_type ).
    rlike(european_pattern))

# Asian Cuisine
asian_cuisine = [ Japanese ,  Korean ,  Chinese ,  Thai ,  Vietnamese
    ,  Indian ,  Malaysian ,  Indonesian ,  Filipino ,  Singaporean ,
    Mongolian ,  Cambodian ,  Sri Lankan ,  Nepalese ,  Bangladeshi ]
asian_pattern =  (?i)(  +  | .join(asian_cuisine) +  )
asian_cuisines = resturants_exploded.filter(col( cuisine_type ).rlike(
    asian_pattern))

# Middle Eastern Cuisine
middle_eastern_cuisine = [ Palestenian ,  Lebanese ,  Turkish ,
    Persian ,  Israeli ,  Jordanian ,  Syrian ,  Iraqi ,  Egyptian ,
    Saudi ,  Emirati ,  Kuwaiti ,  Qatari ,  Bahraini ,  Omani ,
    Yemeni ]
middle_eastern_pattern =  (?i)(  +  | .join(middle_eastern_cuisine) +
     )
middle_eastern_cuisines = resturants_exploded.filter(col( cuisine_type
     ).rlike(middle_eastern_pattern))

# African Cuisine
african_cuisine = [ Moroccan ,  Ethiopian ,  Tunisian ,  Algerian ,
    Egyptian ,  Nigerian ,  Ghanaian ,  South African ,  Kenyan ,
    Tanzanian ,  Ugandan ,  Senegalese ,  Cameroonian ,  Ivorian ]
african_pattern =  (?i)(  +  | .join(african_cuisine) +  )
african_cuisines = resturants_exploded.filter(col( cuisine_type ).
    rlike(african_pattern))
```

```
24
25 # American Cuisine
26 american_cuisine = [ American , Mexican , Brazilian , Argentinian ,
       Peruvian , Colombian , Chilean , Cuban , Puerto Rican ,
   Venezuelan , Salvadoran , Guatemalan , Honduran , Nicaraguan ,
    Costa Rican , Panamanian ]
27 american_pattern = (?i)( + | .join(american_cuisine) + )
28 american_cuisines = resturants_exploded.filter(col( cuisine_type ).
   rlike(american_pattern))
29
30 # All Cuisine Types
31 all_cuisines = european_cuisines.union(asian_cuisines).union(
   middle_eastern_cuisines).union(african_cuisines).union(
   american_cuisines)
32
33 # Reviews that contain legitimate or illegitimate
34 legitimate_reviews = reviews.filter(F.col( text ).rlike( (?i)
   legitimate ))
35 illegitimate_reviews = reviews.filter(F.col( text ).rlike( (?i)
   illegitimate ))
36
37 # Join
38 legitimate_reviews_cuisines = all_cuisines.join(legitimate_reviews, on
   = business_id , how= inner )
39 illegitimate_reviews_cuisines = all_cuisines.join(illegitimate_reviews
   , on= business_id , how= inner )
40
41 # Group by cuisine type and count reviews
42 legitimate_count_by_cuisine = legitimate_reviews_cuisines.groupBy(
   cuisine_type ) \
43     .agg(F.count( * ).alias( legitimate_count )) \
44     .orderBy(F.desc( legitimate_count ))
45
46 illegitimate_count_by_cuisine = illegitimate_reviews_cuisines.groupBy(
     cuisine_type ) \
47     .agg(F.count( * ).alias( illegitimate_count )) \
48     .orderBy(F.desc( illegitimate_count ))
49
50 legitimate_count_by_cuisine.show(50, truncate=False)
51 illegitimate_count_by_cuisine.show(50, truncate=False)
```

Listing 8: How many reviews contain the string "legitimate" grouped by type of cuisine? How about "illegitimate"?

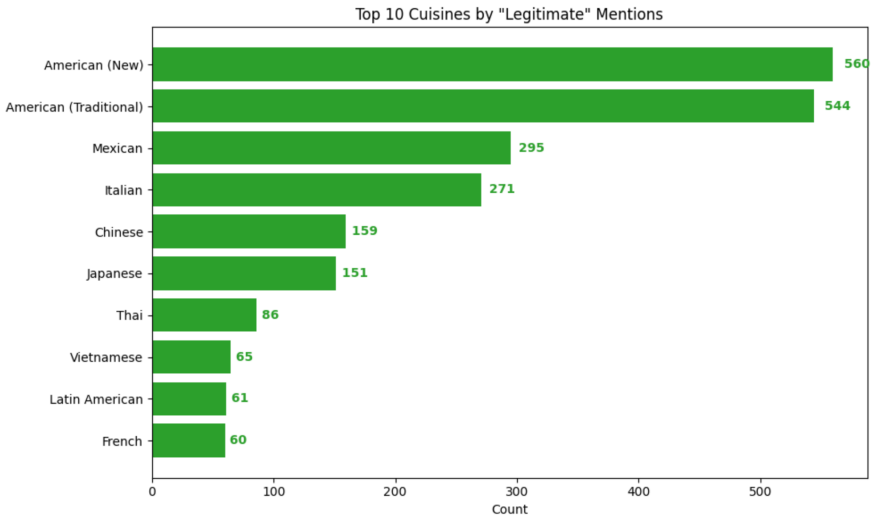

Figure 2: Legitimate Plot

Cuisine Type	**Legitimate Count**
American (New)	560
American (Traditional)	544
Mexican	295
Italian	271
Chinese	159
Japanese	151
Thai	86

Table 8: Top Cuisines for "Legitimate" Reviews

Cuisine Type	**Illegitimate Count**
American (Traditional)	6
Mexican	5
American (New)	4
Italian	3
Vietnamese	2
Latin American	1

Table 9: Top Cuisines for "Illegitimate" Reviews

We start by addressing the lack of a `Cuisine` column. This is by creating predefined keyword lists of cuisines in five major regions (European, Asian, Middle Eastern, African,

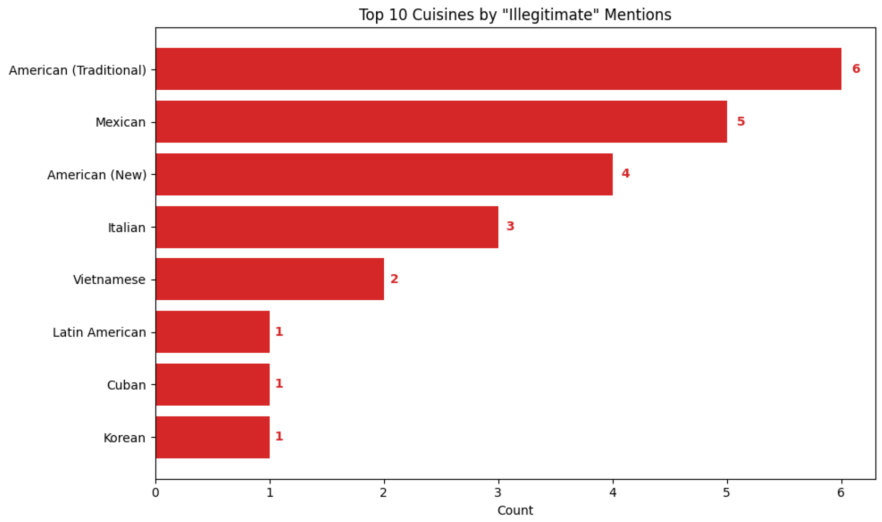

Figure 3: Illegitimate plot

American). Exploded categories (explained in previous tasks) is used to assign business to their specific cuisine. Which we then union, to maintain one large dataframe with all restaurants and their cuisines.

We filter reviews into two separate views, by matching the exact words case-insensitively using regex. These are then joined with the dataframe consisting of cuisines, to link the usage of these words to specific cuisine types.

Finally, we group by `cuisine_type` and count the occurrences across both views.

Findings: The tables above, reveal a massive discrepancy in the usage the terms. The term "legitimate" is used frequently across all major cuisines, with American and Mexican food leading the counts. By contrast, the word "illegitimate" is statistically negligible, appearing in single digits.

This suggests that "legitimate" is part of the vocabulary, while users likely default to other language to describe in-authenticity.

The high frequency in "legitimate" in American (New and Traditional) and Mexican reviews may highlight the underlying composition of the Yelp dataset. As the dataset, has data specific to North America, and the majority of cuisines there are most likely those. So the dominance of American and Mexican Cuisines in the "legitimate" counts likely just reflects the sheer volume and popularity within these North American regions, rather than a unique linguistic tendency.

2.0.3 Question 4: Difference in authenticity language across areas

```
# Flag reviews with authenticity language
```

```
reviews_flagged = reviews.join(business.select( business_id ,  state ,
     city ),  business_id ) \
    .withColumn( has_authenticity , F.col( text ).rlike( (?i)authentic
       |genuine|legit|real|legitimate|valid|original|credible ).cast(
       int ))

# Full cube over state, city
cube_results = reviews_flagged.cube( state ,  city ) \
    .agg(
        F.sum( has_authenticity ).alias( authenticity_count ),
        F.count( * ).alias( total_count )
    ) \
    .withColumn( percentage , (F.col( authenticity_count ) / F.col(
       total_count ) * 100)) \
    .orderBy(F.desc( percentage ))

# However this gives a lot of null values, so we can filter them out
    if needed
cube_filtered = cube_results.filter(F.col( state ).isNotNull() & F.col
    ( city ).isNotNull())
cube_filtered.show(50, truncate=False)
```

Listing 9: Is there a difference in the amount of authenticity language used in the different areas? (e.g., by state, north/south, urban/rural)

State	City	Auth Count	Total Reviews	%
PA	East Greenville	4	5	80.0%
PA	New Britain	4	6	66.7%
NV	Sparks	4	6	66.7%
FL	Bradenton Beach	5	8	62.5%
NJ	Pittsgrove Township	3	5	60.0%

Table 10: Top Cities by Percentage of Authenticity Language

The task recommends using `cube` or `rollup`. To investigate, I used the `cube` function, which allows for multidimensional. First, we join reviews with location data (State and City), including a binary flag `has_authenticity`, using regex.

`cube("state", "city")` is used to generate aggregates for every combination of state and city. Which tells us authenticity trends at city-level, state-level and global (where city and state are null).

Findings: The locations with the highest percentages are small towns or cities with single-digit review counts In these cases, a mere handful of reviews containing the word "authentic" can skew the percentage dramatically. This suggest that to find meaningful geographic trends, we would need to filter for cities with a significant minimum threshold of reviews.

State	City	Auth Count	Total Reviews	%
PA	Philadelphia	270,910	967,517	28.0%
TN	Nashville	112,762	451,506	25.0%
IN	Indianapolis	88,885	361,489	24.6%
LA	New Orleans	154,657	635,329	24.3%
NV	Reno	84,092	351,518	23.9%
FL	Tampa	107,293	454,833	23.6%
AZ	Tucson	94,897	404,865	23.4%

Table 11: Top Cities by Authenticity Percentage (Min. 300k Total Reviews)

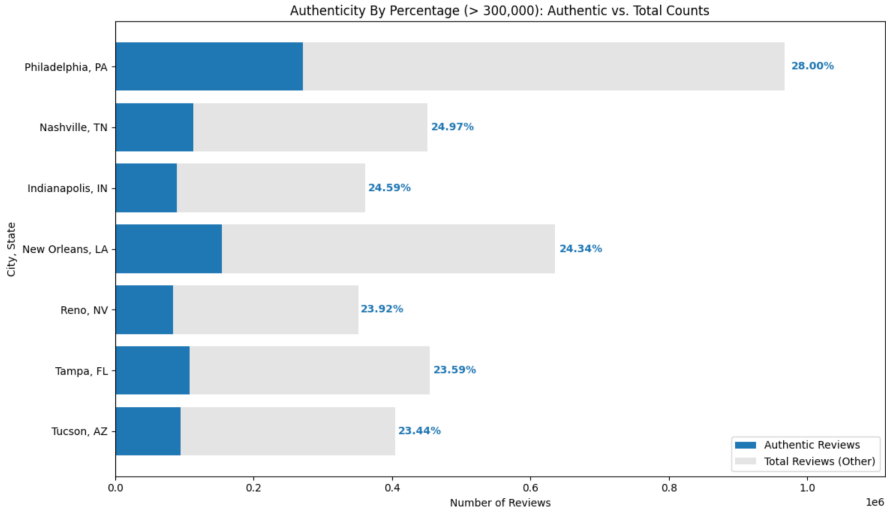

Figure 4: Enter Caption

The above table presents a far more reliable view. The data shows that Edmonton has the highest relative usage of authenticity at 30.7%. Additionally, there seems to be a notable geographic cluster (Greater Philadelphia region). Which may indicate the prevalence of authenticity language is not randomly distributed but appear significantly higher in this region, compared to other metropolitan areas. However, this should be interpreted with caution, as these results are directly influenced by the arbitrary threshold set.

Task 3.2.1: Hypothesis Testing

In the following section, we investigate the following research question: *Can you identify a difference in the relationship between authenticity language and typically negative words, in restaurants serving South American or Asian cuisine compared to restaurants serving European cuisine? And to what degree?.*

The hypothesis suggests that "authenticity" might be a trap for non-European cuisines i.e. used simultaneously with negative feedback, where-as European cuisines, "authenticity" is used as a purely positive descriptor.

```
# Filter to only restaurants and explode cuisine types
resturants = business.select( business_id ,  name ,  categories ) \
                                .filter(business.categories.rlike( (?i)restaurant )) \
                                .filter(F.col( categories ).isNotNull())
resturants_exploded = resturants.withColumn( cuisine_type , F.explode( F.split(F.col( categories ),  ,  )))

# Patterns for authenticity and negative words
authenticity_words = [ authentic ,  genuine ,  legit ,  real ,  legitimate ,  valid ,  original ,  credible ]
negative_words = [ bad ,  terrible ,  awful ,  worst ,  poor ,  disappointing ,  horrible ,  dreadful ,  lousy ,  unpleasant ]

authentic_pattern =  (?i)(  +  | .join(authenticity_words) +  )
negative_pattern =  (?i)(  +  | .join(negative_words) +  )

# Cuisines (South American, Asian, European)
south_american_cuisine = [ Brazilian ,  Argentinian ,  Peruvian ,  Colombian ,  Chilean ,  Cuban ,  Puerto Rican ,  Venezuelan ,  Guatemalan ,  Costa Rican ]
asian_cuisine = [ Japanese ,  Korean ,  Chinese ,  Thai ,  Vietnamese ,  Indian ,  Malaysian ,  Indonesian ,  Filipino ,  Mongolian ,  Sri Lankan ,  Nepalese ]
european_cuisine = [ Italian ,  French ,  Spanish ,  Greek ,  Portuguese ,  German ,  Dutch ,  Belgian ,  Swiss ,  Austrian ,  Hungarian ,  Polish ,  Czech ,  Danish ,  Swedish ,  Norwegian ,  Finnish ]

south_american_asian_pattern =  (?i)(  +  | .join( south_american_cuisine + asian_cuisine) +  )
european_pattern =  (?i)(  +  | .join(european_cuisine) +  )

# Join reciews to get only those with relevant cuisine types
reviews_contain_authenticity_negativity = reviews.join( resturants_exploded.select( business_id , cuisine_type ), on= business_id , how= inner ) \
     .withColumn( cuisine_group ,
         F.when(F.col( cuisine_type ).rlike( south_american_asian_pattern),  South American/Asian )
         .when(F.col( cuisine_type ).rlike(european_pattern),  European )
```

```
            .otherwise(None)) \
        .filter(F.col( cuisine_group ).isNotNull()) \
        .withColumn( has_authenticity , F.col( text ).rlike(
            authentic_pattern).cast( int )) \
        .withColumn( has_negative , F.col( text ).rlike(
            negative_pattern).cast( int ))

authenticity_negativity_reduced =
    reviews_contain_authenticity_negativity.select( cuisine_group ,
    has_authenticity ,  has_negative ,  stars )
```

Listing 10: Hypothesis Testing 1

```
authenticity_negativity_stars_by_region = (
    authenticity_negativity_reduced.withColumn( star_group ,
            F.when(F.col( stars ) < 3,  Low Stars ).when(F.col( stars
                ) > 3,  High Stars ).otherwise( Neutral Stars ))
        .groupBy( region_group ,  star_group )
            .agg(
                F.count( * ).alias( total_count ),
                F.sum( has_authenticity ).alias( authenticity_count ),
                F.sum( has_negative ).alias( negativity_count ),
                F.sum(F.when((F.col( has_authenticity ) == 1) & (F.col
                    ( has_negative ) == 1), 1).otherwise(0)
            ).alias( both_count ))
        .withColumn( authenticity_percentage , F.round(100.0 * F.col(
            authenticity_count ) / F.col( total_count ), 3))
        .withColumn( negativity_percentage , F.round(100.0 * F.col(
            negativity_count ) / F.col( total_count ), 3))
        .withColumn( both_percentage , F.round(100.0 * F.col(
            both_count ) / F.col( total_count ), 3))
        .orderBy( region_group ,  star_group )
        .select( region_group ,  star_group ,  authenticity_percentage
            ,  negativity_percentage ,  both_percentage ))

authenticity_negativity_stars_by_region.show()
```

Listing 11: Hypothesis Testing 2

To test this, we categorized restaurants into two group: **European** and **South American/Asian**. This is done by filtering business' to restaurants and exploding their categories into `cuisine_types`; we define two keyword lists and use them to build a case-insensitive regex pattern.

Next, we join reviews to these cuisine groups and create binary flags for whether they contain authenticity and negativity (using similar regex matching).

To figure out user sentiment, we separate review rating into three categories: **Low** (< 3 stars), **Neutral** ($= 3$ stars), and **High** (> 3 stars). Finally, we aggregate the percentage

of reviews in each category, that contain either authenticity, negativity, or crucially both (co-occurrence).

To ensure the validity, of the observations, Chi-squared is also performed, to check for statistically significant differences between the two cuisine groups.

Findings: The Table below, presents the percentage of reviews containing authenticity terms, negative terms, and their co-occurrence (both). Unexpectedly, the data reveals almost identical patterns across both cuisine groups. Authenticity is mentioned in approximately 26-27% of reviews for both groups for high star reviews. While negativity is around 5%, with a negligible co-occurrence.

For low star groups, negativity sky rockets to 42-43% for both groups, as expected. Although curiously, authenticity is still used a good amount, rocking around 23-24%.

This lack of variation suggests there is no meaningful difference between the two cuisines groups. However, it may suggest authenticity is not inherently a positive quality marker, but rather a descriptor used even when the overall experience is poor.

To validate my findings, `chi-squared` was performed. The formal hypothesis test confirmed the absence of any statistical difference. However, as this was not introduced in the course, I have omitted it from the report.

Table 12: Authenticity and negativity percentages by region and star group

Region	Star group	Authenticity (%)	Negativity (%)	Both (%)
European	High Stars	26.057	5.103	1.930
European	Low Stars	23.639	42.468	11.423
European	Neutral Stars	30.766	21.813	8.289
South American/Asian	High Stars	27.386	5.284	2.044
South American/Asian	Low Stars	24.643	43.085	11.695
South American/Asian	Neutral Stars	31.730	21.772	8.504

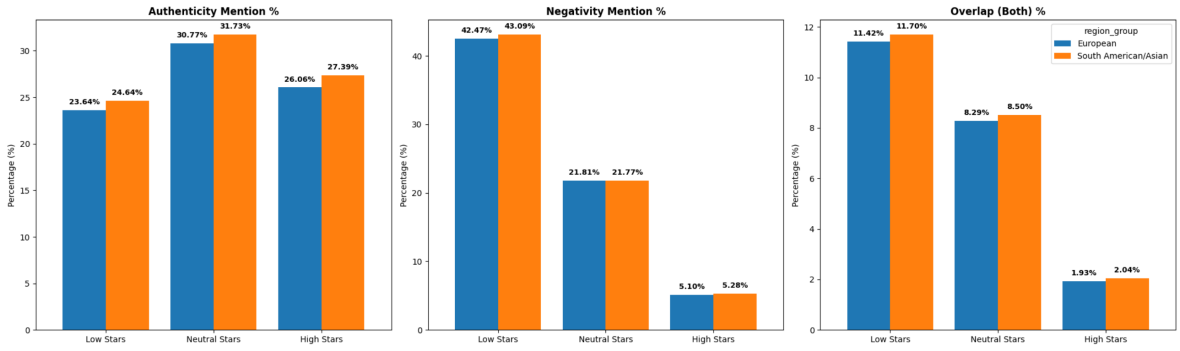

Figure 5: Enter Caption

3 Rating Prediction

Building a Rating Prediction Model

I built a three-class classifier to predict review rating: `low`, `neutral`, `high`. The pipeline consists of filtering and labeling restaurant reviews, cleaning and pre-processing text, an 80/20 train/test split with class balancing on the training set, and a TF-IDF feature pipeline. The main steps are shown in the listings below.

```
# Load reviews from resturants only
resturants = business.select( business_id ,  name ,  categories ,  stars ).filter(business.categories.rlike( (?i)restaurant ))

# However reviews also has stars column, so we have to the business stars column
resturants = resturants.withColumnRenamed( stars ,  business_stars )
reviews_resturants = reviews.join(resturants, on= business_id , how= inner )

# Remove all columns that are not important
reduced_reviews_resturants = reviews_resturants.select( business_id ,  text ,  stars ,  business_stars )

# Label the reviews based on their stars
reviews_stars_labeled = reduced_reviews_resturants.withColumn( rating_label , \
    F.when(F.col( stars ) <= 2, F.lit(0.0)).when(F.col( stars ) == 3, F.lit(1.0)).when(F.col( stars ) >= 4, F.lit(2.0))) \
    .withColumn( rating_label , F.col( rating_label ).cast(DoubleType()))
```

Listing 12: Filter and label restaurant reviews.

I start by filtering the `business` into restaurants and renaming the `stars` column to avoid conflicts during the join. The reviews are labeled into three discrete labels: `0.0` (Low, ≤ 2 stars), `1.0` (Neutral, 3 stars), and `2.0` (High, ≥ 4 stars). These three labels are chosen due to the difference between 4 and 5-star review often is subjective, whereas the difference between positive, neutral and negative is more distinct.

```
# Clean the data and preprocess
processed_reviews = reviews_stars_labeled.withColumn( text , F.regexp_replace(F.col( text ),  [^a-zA-Z0-9\\s] ,  )) \
    .withColumn( text , F.lower(F.col( text ))) \
    .withColumn( review_stars , F.col( stars ).cast(DoubleType())) \
    .withColumn( business_stars , F.col( business_stars ).cast(DoubleType()))
```

Listing 13: Clean and preprocess review text.

Text was cleaned by removing non-alphanumeric characters and converting to lowercase. The columns are cast to double to avoid issues.

```
# Split the data into training and test sets (80% training, 20% test)
train_data, test_data = processed_reviews.randomSplit([0.8, 0.2], seed=42)

# Balancing
low_test = train_data.filter(train_data.rating_label == 0.0).count()
neutral_test = train_data.filter(train_data.rating_label == 1.0).count()
high_test = train_data.filter(train_data.rating_label == 2.0).count()

target = int(min(low_test, neutral_test, high_test))
seed = 42

parts = []
for label, count in [(0.0, low_test), (1.0, neutral_test), (2.0, high_test)]:
    df_label = train_data.filter(F.col( rating_label ) == label)
    if count > target:
        frac = float(target) / float(count)
        sampled = df_label.sample(withReplacement=False, fraction=frac, seed=seed)
    else:
        sampled = df_label
    parts.append(sampled)

trained_balanced = reduce(lambda a, b: a.union(b), parts)

# Unbalanced
train_unbalanced = train_data

# Class Distirbutions before and after:
train_unbalanced.groupBy( rating_label ).count().orderBy( rating_label ).show()
trained_balanced.groupBy( rating_label ).count().orderBy( rating_label ).show()
```

Listing 14: Train/test split and class balancin

Table 13: Class Distribution Before and After Under-sampling

Rating Label	Sentiment	Unbalanced Count	Balanced Count
0.0	Low	776,688	434,644
1.0	Neutral	434,367	434,367
2.0	High	2,566,885	433,980

The cleaned data is split into training and test sets. This is important to ensure generalization; model can predict ratings for new, unseen reviews rather than simply memorizing the training data. Additionally, the data reveals are severe class imbalance: High rating (2566885), Neutral (434367), and Low (776688). To prevent the model from biasing toward the majority class (High), I applied random under-sampled the training data to match the count of the smallest label. This tries to ensure the classifier learns to distinguish all categories equally rather than defaulting to the most frequent one. However, it is worth noting that we discard over 2.1 million "High" star reviews, and may suffer from sampling bias (the "high" reviews not being perfectly representative).

The resulting count fo the data can be seen above.

```
# Feature extraction, using TF-IDF
tokenizer = Tokenizer(inputCol= text , outputCol= words )
remover = StopWordsRemover(inputCol= words , outputCol= filtered_words )
hashingTF = HashingTF(inputCol= filtered_words , outputCol= raw_features , numFeatures=10000)
idf = IDF(inputCol= raw_features , outputCol= features )
lr = LogisticRegression(featuresCol= features , labelCol= rating_label , maxIter=20)

# Pipeline
pipeline_tfidf = Pipeline(stages=[tokenizer, remover, hashingTF, idf, lr])

# Evaluate the models
evaluator_acc = MulticlassClassificationEvaluator(labelCol= rating_label , predictionCol= prediction , metricName= accuracy )
evaluator_f1 = MulticlassClassificationEvaluator(labelCol= rating_label , predictionCol= prediction , metricName= f1 )

training_sets = [( Unbalanced , train_unbalanced), ( Balanced , trained_balanced)]

for name, train_set in training_sets:
    # Fit the model
    model = pipeline_tfidf.fit(train_set)

```

```
    # Make predictions on the shared test set
    predictions = model.transform(test_data)

    # Evaluate
    acc = evaluator_acc.evaluate(predictions)
    f1 = evaluator_f1.evaluate(predictions)

    print(f {name} TF-IDF Model Accuracy: {acc:.4f} )
    print(f {name} TF-IDF Model F1 score: {f1:.4f} )
```

Listing 15: TF–IDF pipeline and model training

I specifically chose Multiclass Classification, rather than Regression for this task, this is due to star rating being discrete values, than than continuous values. Additionally by using classification, we can bin the data into three broad categories. This is valuable as distinguishing for example a 4-star rating from a 5-star rating is notoriously difficult and highly subjective between users. Grouping these into a "High" class creates a more robust and interpretable target rather that predicting the exact numerical score. However, it can be argued that this approach leads to a loss of variance.

My feature extraction uses a standard `TF-IDF` pipeline. A logistic regression classifier was trained on both an unbalanced and balanced data via a `Spark ML` pipeline. The model performance was measured with `accuracy` and an `F1` score.

Rating Prediction Model Results

Table 14: Performance Comparison: Unbalanced vs. Balanced TF-IDF Models

Model Strategy	**Accuracy**	**F1 Score**
Unbalanced (Full Data)	0.8576	0.8421
Balanced (Under-sampled)	0.8161	0.8332

Findings The table above compares the performance of the Unbalanced and Balanced models. The Unbalanced model interestingly achieves the highest metrics (Accuracy: 0.8576, F1: 0.8421), slightly outperforming the Balanced approach. The high accuracy is likely misleading. The default F1 metric in Spark ML is **Weighed F1 Score**. Because the dataset is heavily skewed towards positive ratings (68 procent positive rating), this metric metric is mathematically dominated by the majority class. The Unbalanced model likely achieves its high score by maximizing recall on "High", which accounts for the majority of the "weight" in the final calculation, which can obscure poor performance on "Low" and "Neutral" reviews.

In contrast, the **Balanced Model** (F1: 0.8332) cannot "hide" behind a majority class. In a balanced dataset, every class contributes equally to the final score. Therefore, the fact that

the Balanced model maintains a high F1 score proves it is genuinely distinguishing between `Low`, `Neutral`, and `High`.

Limitations: While an accuracy of 0.81 is good enough, the results could be significantly higher (up towards 95%). I argue that the performance is primary due to the choice of model architecture and feature extraction.

The model relies on TF-IDF, which works by counting how often a word appears in a specific review (TF), and weighing it down if it appears in all reviews (IDF), effectively highlighting unqiue keywords. However, TF-IDF ignores word order and context; "not good" and "good" mathematically similar, as they shared the same word "good"

I believe that this is single biggest factor limiting higher accuracy. Additionally, the decision to bin rating, was a trade-off, While this simplified the problem, it may have been suboptimal, as it relies on the assumption that 3-star reviews are always "Neutral", users often give 3 stars for "Great food, terrible service" (Mixed), while simple review saying "Okay good" is actually Neutral. Additionally, grouping 1-2 stars reviews (Low) and 4-5 stars (High) sacrifices granularity. The model is forced to treat them as identical potentially confusing the decision boundary.

General Limitation: The pipeline is intentionally simple, and as such there obviously limitations. These include information loss from text cleaning, sensitivity to the under-sampling strategy, and also the coarse three class label mapping. Misclassifications are expected to happen for reviews containing mixed sentiment (praise of food but complaints about service). Improvements could be class-weighting (assigning higher penalty weights to minority classes to handle imbalance without the data loss cuased by under-sampling or noise by oversampling) and using a more expressive classifier.

Table 15: Sample predictions from the TF-IDF model

Text	**Prediction**
The food was really good and the staff were very friendly to me	2.0
The food was okay, nothing special but not terrible	0.0
The food was very great, but the service was terrible	0.0
Horrible service, my order took way too long and was even wrong	0.0

4 Potential Exam Questions

Regarding legitimate and illegitimate categories, how did you implement the word boundaries in regex?

For rating prediction task, you could implement between a Classifier and Regression. Justify this choice and explain the trade-offs

The yelp dataset is heavily skewed towards positive reviews. Explain your strategy for handling this class imbalance, and discuss potential risks by your approach

Chi-Squared

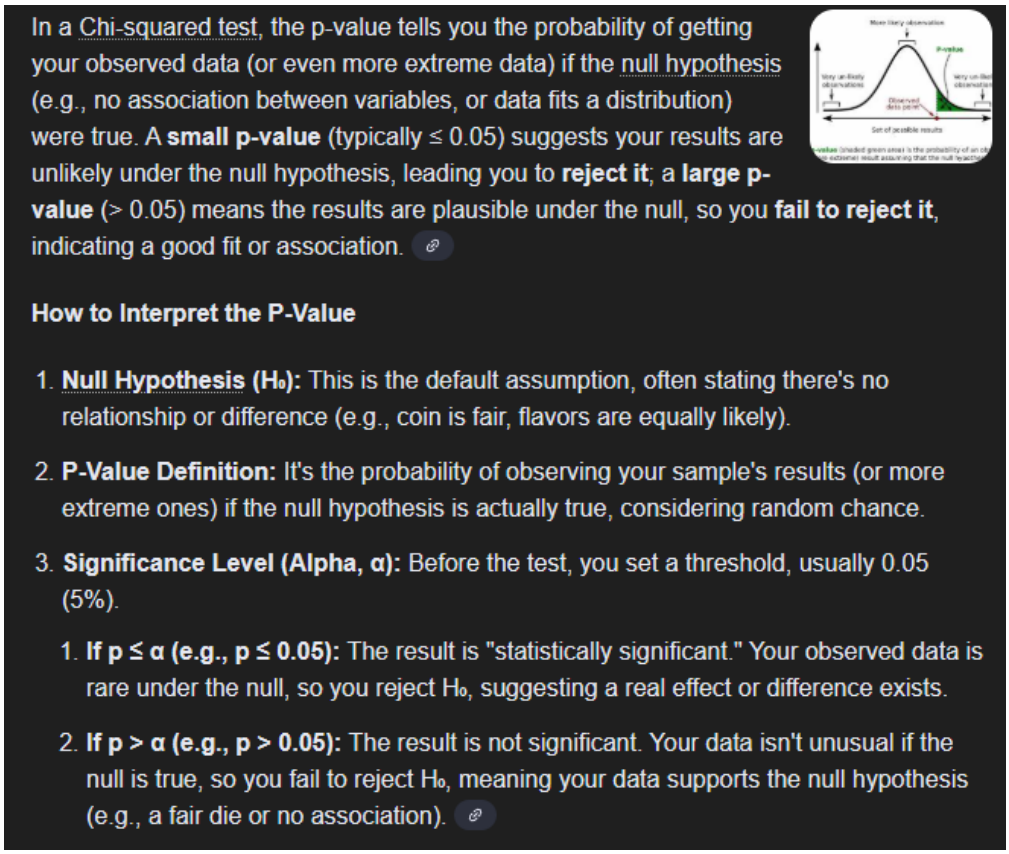

Figure 6: Notes for Chi-squared Test