**SIG743 End Term Assessment**

|  |  |
| --- | --- |
| Unit Code & Name | SIG742 - Modern Data Science |
| Assignment | End Term |
| Submission Date | October 0, 2025 |
| Team Members | Ankur Luthra (Student Id: **S225697234**)  Frederick Abbey (Student Id :)  Mmusi Dithotse (Student Id :) |

Table of Contents

[Background 4](#_Toc210516562)

[1. Load and explore the dataset 4](#_Toc210516563)

[Data Overview 5](#_Toc210516564)

[Structure of the data 5](#_Toc210516565)

[Missing Value Check 5](#_Toc210516566)

[Duplicate Values 5](#_Toc210516567)

[Invalid Values 5](#_Toc210516568)

[Statistical Summary 5](#_Toc210516569)

[Part 1 5](#_Toc210516570)

[Answer 1.1 5](#_Toc210516571)

[Answer 1.2 5](#_Toc210516572)

[Answer 1.3 5](#_Toc210516573)

[Answer 1.4 5](#_Toc210516574)

[Answer 1.5 6](#_Toc210516575)

[Answer 1.6 6](#_Toc210516576)

[Answer 1.6.1 6](#_Toc210516577)

[Answer 1.6.2 6](#_Toc210516578)

[Answer 1.7 6](#_Toc210516579)

[Answer 1.7.1 Explore the relationships of the rating and business categories 6](#_Toc210516580)

[Answer 1.7.2 Analysis of Low-Rating Reviews 17](#_Toc210516581)

[Answer 1.8 27](#_Toc210516582)

[Answer 1.8.1 Compile User Business History 27](#_Toc210516583)

[Answer 1.8.2 Remove Duplicate Businesses per User 30](#_Toc210516584)

[Answer 1.8.3 Analyse User Similarities Based on Reviewed Businesses 37](#_Toc210516585)

[Part 2 37](#_Toc210516586)

[Answer 2.1 37](#_Toc210516587)

[Answer 2.2 37](#_Toc210516588)

[Answer 2.3 37](#_Toc210516589)

**Table of Figures**

[Figure 1 Structure of Data 5](#_Toc210516590)

[Figure 2 Relationship of Ratings with Business 12](#_Toc210516591)

[Figure 3 Category Rating Visualization 14](#_Toc210516592)

[Figure 4 User Sentiment (by Category) 15](#_Toc210516593)

[Figure 5 Word clouds showing common words in low ratings 22](#_Toc210516594)

[Figure 6 Top Negative Themes in Reviews 23](#_Toc210516595)

[Figure 7 Principal Features of Negative Review Text 25](#_Toc210516596)

[Figure 8 Key Issues Across Main Business Segments 25](#_Toc210516597)

[Figure 9 Breakdown of Negative Themes per Business Category 26](#_Toc210516598)

[Figure 10 Impact of Duplicate Reviews on Reviewer History Metrics 34](#_Toc210516599)

# Background

The project focuses on analysing businesses and their customer reviews. The business dataset captures attributes such as location, category, ratings, and operational details, providing a structural overview of the business landscape. The review dataset contains user-generated ratings, text, and timestamps, enabling exploration of temporal trends in customer engagement and satisfaction. By linking businesses and reviews via unique identifiers, the datasets support forecasting future review activity, identifying patterns and seasonality, and applying advanced predictive models such as ARIMA, SARIMA, and LSTM for time series analysis.

# Load and explore the dataset

**Shape of the Data:**

* **Business dataset**: 12,774 rows × 15 columns
* **Review dataset**: 521,515 rows × 8 columns

**Business Dataset Columns:**

* name: Business name
* address: Business address
* gmap\_id: Unique Google Maps identifier
* description: Text description of the business
* latitude / longitude: Geolocation coordinates
* category: Business type/category
* avg\_rating: Average rating of the business
* num\_of\_reviews: Total number of reviews
* price: Price range of the business
* hours: Business operating hours
* MISC: Miscellaneous notes
* state: State or region of the business
* relative\_results: Related business results
* url: Google Maps URL for the business

**Review Dataset Columns:**

* user\_id: Unique identifier of the reviewer
* name: Name of the business reviewed
* time: Timestamp of the review
* rating: Rating given by the user
* text: Review text content
* pics: Photos uploaded with the review
* resp: Business responses to reviews
* gmap\_id: Unique Google Maps identifier linking to business

## Data Overview

### Structure of the data

|  |  |
| --- | --- |
| Figure 1 Structure of Data |  |

### Missing Value Check

### Duplicate Values

### Invalid Values

### Statistical Summary

# Part 1

## Answer 1.1

## Answer 1.2

## Answer 1.3

## Answer 1.4

## Answer 1.5

## Answer 1.6

### Answer 1.6.1

### Answer 1.6.2

## Answer 1.7

### Answer 1.7.1 Explore the relationships of the rating and business categories

#### Code

|  |  |
| --- | --- |
| |  | | --- | | def parse\_categories(category\_str):      """      Parse category string that may contain multiple categories in list format.      Returns a list of individual categories.      """      if pd.isna(category\_str):          return ['Unknown']        try:          if isinstance(category\_str, str) and category\_str.startswith('['):              categories = ast.literal\_eval(category\_str)              if isinstance(categories, list):                  return categories              else:                  return [str(categories)]          else:              return [str(category\_str)]      except:          return [str(category\_str)]  def get\_primary\_category(category\_str):      """      Get the primary (first) category from multi-category data.      """      categories = parse\_categories(category\_str)      return categories[0] if categories else 'Unknown'  def clean\_category\_name(category):      """      Clean category names for better visualization.      """      if pd.isna(category) or category == 'Unknown':          return 'Unknown'        # Remove common prefixes/suffixes and clean up      category = str(category).strip()        # Handle common variations      replacements = {          'Restaurant': 'Restaurant',          'Food': 'Food & Dining',          'Shopping': 'Shopping',          'Health': 'Health & Medical',          'Beauty': 'Beauty & Spa',          'Automotive': 'Automotive',          'Entertainment': 'Entertainment',          'Professional': 'Professional Services'      }        for key, value in replacements.items():          if key.lower() in category.lower():              return value        return category | |
| # Data preprocessing and merging  if 'reviews\_df' in locals() and 'business\_df' in locals():        # Process categories      print("\n Processing business categories...")      # Parse categories into lists for each business      merged\_df['category\_list'] = merged\_df['category'].apply(parse\_categories)        # Create a copy for exploding - this will create multiple rows per review if multiple categories      print("Exploding categories to capture all category associations...")      exploded\_df = merged\_df.explode('category\_list').copy()        # Clean the exploded categories      exploded\_df['clean\_category'] = exploded\_df['category\_list'].apply(clean\_category\_name)        # Remove any rows where category parsing failed      exploded\_df = exploded\_df[exploded\_df['clean\_category'] != 'Unknown'].copy()        print(f"Original reviews: {len(merged\_df):,}")      print(f"After category explosion: {len(exploded\_df):,} review-category combinations")        # Get category statistics      category\_counts = exploded\_df['clean\_category'].value\_counts()      print(f"Found {len(category\_counts)} unique categories")      top\_n = 10      print(f"\nTop {top\_n} categories (after explosion):")        for cat, count in category\_counts.head(top\_n).items():          percentage = count/len(exploded\_df)\*100          print(f"  {cat}: {count:,} review-category pairs ({percentage:.1f}%)")        # Filter for analysis (keep categories with sufficient data)      min\_reviews\_per\_category = 100  # Minimum reviews for meaningful analysis      popular\_categories = category\_counts[category\_counts >= min\_reviews\_per\_category].index        analysis\_df = exploded\_df[exploded\_df['clean\_category'].isin(popular\_categories)].copy()      print(f"\n Analysis dataset:")      print(f"Categories with ≥{min\_reviews\_per\_category} reviews: {len(popular\_categories)}")      print(f"Review-category pairs in analysis dataset: {len(analysis\_df):,} ({len(analysis\_df)/len(exploded\_df)\*100:.1f}% of exploded total)")      print(f"Unique reviews in analysis: {analysis\_df['user\_id'].nunique():,}")      print(f"Unique businesses in analysis: {analysis\_df['gmap\_id'].nunique():,}")        # Create rating categories for analysis      analysis\_df['rating\_category'] = analysis\_df['rating'].apply(          lambda x: 'Low (1-2)' if x <= 2 else 'Medium (3)' if x == 3 else 'High (4-5)'      )        print(f"\n Data preprocessing completed")      print(f"Ready for analysis with {len(analysis\_df):,} review-category pairs across {len(popular\_categories)} categories")      print(f"\n BENEFIT: By exploding categories, we capture ALL category associations.")      print(f"   Example: A review for 'Italian Restaurant' now contributes to both 'Restaurant' AND 'Italian' analysis.")      print(f"   This provides more comprehensive insights into category-specific patterns.")  else:      print(" Cannot proceed - data not loaded") |
| if 'analysis\_df' in locals():      print("\n CREATING VISUALIZATIONS: RATING-CATEGORY RELATIONSHIPS")      print("=" \* 60)        # Set up the plotting environment      plt.rcParams['figure.figsize'] = (15, 10)      # plt.rcParams['font.size'] = 12      top\_categories = analysis\_df['clean\_category'].value\_counts().head(50).index      analysis\_df = analysis\_df[analysis\_df['clean\_category'].isin(top\_categories)]        # Create 4 subplots, each in its own row      fig, axes = plt.subplots(4, 1, figsize=(20, 28))      fig.suptitle('Rating Analysis by Business Category', fontsize=18, fontweight='bold', y=1.02)        # Subplot 1: Average Rating by Category      avg\_ratings = analysis\_df.groupby('clean\_category')['rating'].agg(['mean', 'count']).reset\_index()      avg\_ratings = avg\_ratings.sort\_values('mean', ascending=True)      axes[0].barh(avg\_ratings['clean\_category'], avg\_ratings['mean'], color='skyblue')      axes[0].set\_xlabel('Average Rating')      axes[0].set\_title('Average Rating by Category')      axes[0].set\_xlim(0, 5)      for i, v in enumerate(avg\_ratings['mean']):          axes[0].text(v + 0.05, i, f'{v:.2f}', va='center')        # Subplot 2: Rating Distribution Heatmap      rating\_category\_crosstab = pd.crosstab(analysis\_df['clean\_category'], analysis\_df['rating'], normalize='index') \* 100      sns.heatmap(rating\_category\_crosstab, annot=True, fmt='.1f', cmap='RdYlBu\_r',                  ax=axes[1], cbar\_kws={'label': 'Percentage of Reviews'})      axes[1].set\_title('Rating Distribution by Category (%)')      axes[1].set\_xlabel('Rating')      axes[1].set\_ylabel('Category')        # Subplot 3: Box Plot of Ratings by Category      top\_categories\_box = analysis\_df['clean\_category'].value\_counts().head(8).index      top\_cat\_data = analysis\_df[analysis\_df['clean\_category'].isin(top\_categories\_box)]      sns.boxplot(data=top\_cat\_data, y='clean\_category', x='rating', ax=axes[2])      axes[2].set\_title('Rating Distribution by Top Categories (Box Plot)')      axes[2].set\_xlabel('Rating')        # Subplot 4: Volume vs Quality Scatter      category\_stats = analysis\_df.groupby('clean\_category').agg({          'rating': ['mean', 'count', 'std']      }).round(2)      category\_stats.columns = ['avg\_rating', 'review\_count', 'rating\_std']      category\_stats = category\_stats.reset\_index()      scatter = axes[3].scatter(category\_stats['review\_count'], category\_stats['avg\_rating'],                                s=category\_stats['rating\_std']\*50, alpha=0.6, c=category\_stats['avg\_rating'],                                cmap='RdYlBu\_r')      axes[3].set\_xlabel('Number of Reviews')      axes[3].set\_ylabel('Average Rating')      axes[3].set\_title('Review Volume vs Average Rating\n(Bubble size = Rating Std Dev)')      for idx, row in category\_stats.iterrows():          if row['review\_count'] > 500:              axes[3].annotate(row['clean\_category'][:15],                               (row['review\_count'], row['avg\_rating']),                               xytext=(5, 5), textcoords='offset points', fontsize=8)        plt.tight\_layout(rect=[0, 0, 1, 0.98])      plt.show()        print("\n Statistical Summary by Category:")      print(category\_stats.sort\_values('avg\_rating', ascending=False))  else:      print(" Analysis data not available") |
| if 'analysis\_df' in locals():      print(" CREATING INTERACTIVE VISUALIZATIONS")      print("=" \* 40)        # 1. Interactive Rating Distribution by Category      fig = px.violin(analysis\_df, y='clean\_category', x='rating',                      title='Interactive Rating Distribution by Category',                      labels={'rating': 'Rating', 'clean\_category': 'Business Category'})      fig.update\_layout(height=600)      fig.show()        # 2. Rating Trends - Sunburst Chart      rating\_summary = analysis\_df.groupby(['clean\_category', 'rating\_category']).size().reset\_index(name='count')        fig = px.sunburst(rating\_summary, path=['clean\_category', 'rating\_category'], values='count',                        title='Rating Distribution Hierarchy by Category',                        color='count', color\_continuous\_scale='RdYlBu\_r')      fig.show()        # 3. Statistical Analysis Table      print("\n DETAILED STATISTICAL ANALYSIS")      print("=" \* 40)        detailed\_stats = analysis\_df.groupby('clean\_category').agg({          'rating': ['count', 'mean', 'median', 'std', 'min', 'max'],          'gmap\_id': 'nunique'  # Number of unique businesses      }).round(3)        detailed\_stats.columns = ['Total\_Reviews', 'Mean\_Rating', 'Median\_Rating',                               'Std\_Rating', 'Min\_Rating', 'Max\_Rating', 'Unique\_Businesses']      detailed\_stats = detailed\_stats.reset\_index()      detailed\_stats['Avg\_Reviews\_per\_Business'] = (detailed\_stats['Total\_Reviews'] /                                                    detailed\_stats['Unique\_Businesses']).round(1)        # Sort by mean rating      detailed\_stats = detailed\_stats.sort\_values('Mean\_Rating', ascending=False)        print(detailed\_stats.to\_string(index=False))        # 4. Low vs High Rating Analysis      low\_high\_analysis = analysis\_df.groupby('clean\_category')['rating\_category'].value\_counts(normalize=True).unstack(fill\_value=0) \* 100      low\_high\_analysis = low\_high\_analysis.round(1)        print("\n\nLOW vs HIGH RATING PERCENTAGES BY CATEGORY")      print("=" \* 50)      print(low\_high\_analysis.to\_string())  else:      print(" Analysis data not available") |

#### Visualizations

|  |
| --- |
| Figure 2 Relationship of Ratings with Business |

#### Output

|  |
| --- |
| Processing business categories...  Exploding categories to capture all category associations...  Original reviews: 516,289  After category explosion: 1,626,145 review-category combinations  Found 1379 unique categories  Top 10 categories (after explosion):  Restaurant: 509,134 review-category pairs (31.3%)  Tourist attraction: 40,041 review-category pairs (2.5%)  Grocery store: 35,478 review-category pairs (2.2%)  Coffee shop: 29,689 review-category pairs (1.8%)  Sandwich shop: 25,119 review-category pairs (1.5%)  Bar: 24,430 review-category pairs (1.5%)  Sporting goods store: 18,705 review-category pairs (1.2%)  Department store: 18,619 review-category pairs (1.1%)  Clothing store: 18,458 review-category pairs (1.1%)  Supermarket: 16,589 review-category pairs (1.0%)  Analysis dataset:  Categories with ≥100 reviews: 720  Review-category pairs in analysis dataset: 1,602,934 (98.6% of exploded total)  Unique reviews in analysis: 20,022  Unique businesses in analysis: 5,791  Data preprocessing completed  Ready for analysis with 1,602,934 review-category pairs across 720 categories  BENEFIT: By exploding categories, we capture ALL category associations.  Example: A review for 'Italian Restaurant' now contributes to both 'Restaurant' AND 'Italian' analysis.  This provides more comprehensive insights into category-specific patterns. |

**Top 10 Categories by Average Rating**

|  |  |  |  |
| --- | --- | --- | --- |
| **clean\_category** | **avg\_rating** | **review\_count** | **rating\_std** |
| Tourist attraction | 4.61 | 40041 | 0.73 |
| Park | 4.55 | 14431 | 0.77 |
| Cannabis store | 4.53 | 5296 | 0.93 |
| Warehouse store | 4.46 | 5064 | 0.86 |
| Brewpub | 4.43 | 8302 | 0.88 |
| Bakery | 4.4 | 7865 | 0.94 |
| Cafe | 4.39 | 16238 | 0.97 |
| Gift shop | 4.38 | 7441 | 0.93 |
| Food & Dining | 4.38 | 5511 | 0.89 |
| Movie theater | 4.38 | 7078 | 0.87 |

**Bottom 10 Categories by Average Rating**

|  |  |  |  |
| --- | --- | --- | --- |
| **clean\_category** | **avg\_rating** | **review\_count** | **rating\_std** |
| Electronics store | 3.88 | 13314 | 1.18 |
| Discount store | 3.83 | 10767 | 1.19 |
| Home goods store | 3.83 | 10925 | 1.19 |
| Craft store | 3.9 | 13621 | 1.16 |
| Cell phone store | 3.95 | 4629 | 1.31 |
| Toy store | 3.95 | 13516 | 1.15 |
| Clothing store | 3.97 | 18458 | 1.16 |
| Sandwich shop | 3.98 | 25119 | 1.17 |
| Supermarket | 4 | 16589 | 1.12 |
| Department store | 4.03 | 18619 | 1.14 |

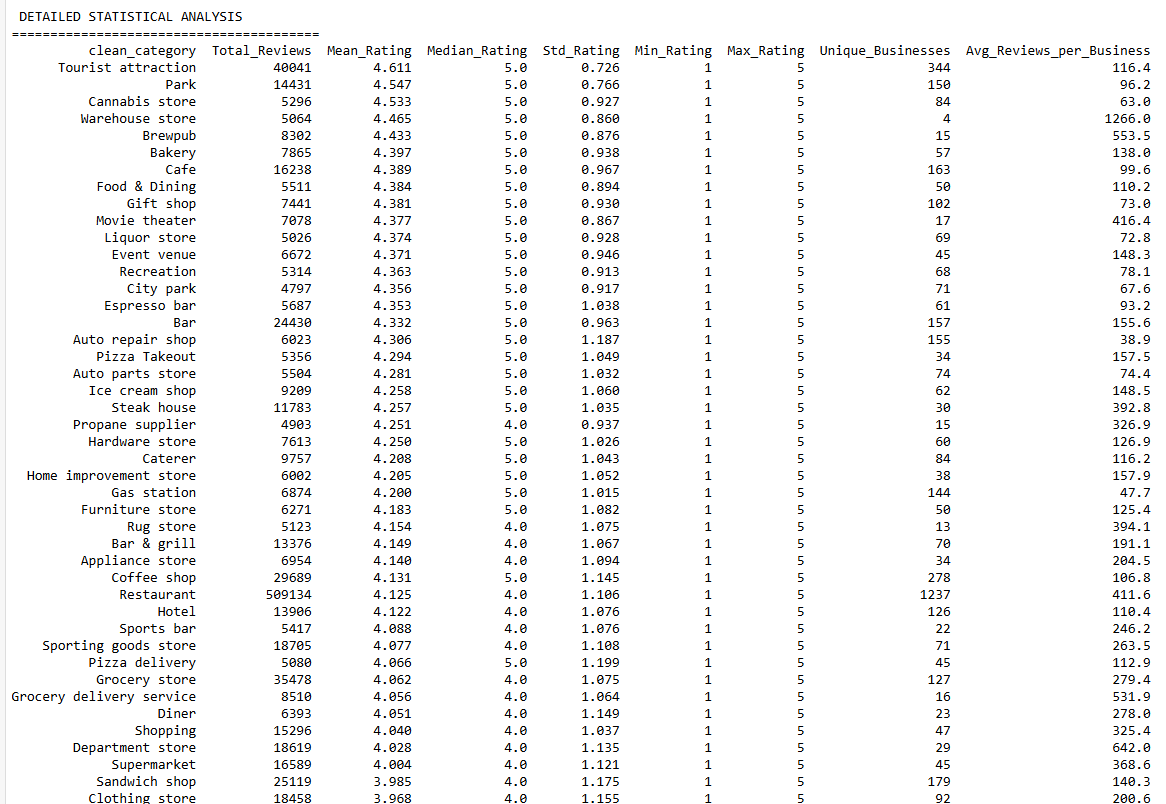


Figure 3 Category Rating Visualization

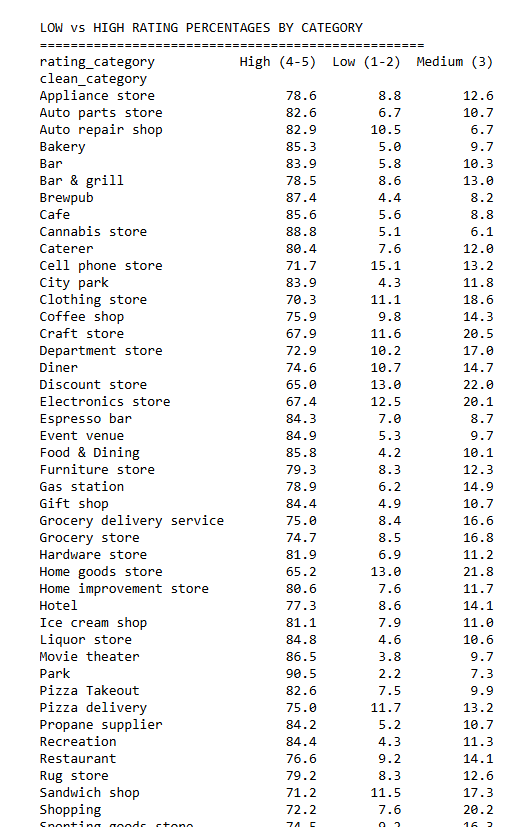


Figure 4 User Sentiment (by Category)

#### Insights

**Category Coverage and Explosion**

* Exploding multi-category businesses increased review-category pairs from about 516,000 to 1.6 million across 720 categories. This ensures each review contributes to all relevant categories.

**Review Volume Distribution**

* Restaurants account for 31% of reviews, followed by tourist attractions at 2.5% and grocery stores at 2.2%. The top 10 categories dominate review activity, highlighting where customers focus.

**Average Ratings and Variability**

* Tourist attractions have the highest average rating at 4.61, with low variability at 0.73. This indicates consistently positive experiences. Clothing stores and sandwich shops have lower average ratings, around 3.97 to 3.98, with higher variability, around 1.15 to 1.17. This shows more mixed customer experiences.

**High vs Low Ratings**

* Most categories have over 70% high ratings, which are between 4 and 5. Tourist attractions show 92% high ratings and only 1.9% low ratings. In contrast, clothing stores and sandwich shops have around 11 to 12% low ratings, suggesting areas that need improvement.

**Business-Level Insights**

* Average reviews per business vary widely: department stores have about 642, restaurants around 412, and sporting goods stores approximately 264. High-volume categories maintain strong ratings, suggesting quality experiences can scale well.

**Strategic Takeaways**

* Categories with lower ratings or higher variability indicate potential service gaps. High-rated, high-volume categories, like tourist attractions and bars, may reveal operational best practices worth following. The exploded category approach allows for multi-dimensional analysis of customer satisfaction across overlapping categories.

#### Explanation

**Why we chose this solution**

* We decided to break down business categories and clarify their names. This ensures that every review is counted in all relevant categories. For instance, a review for a "Mexican Restaurant" counts for both 'Mexican' and 'Restaurant.' This method gives a clear view of rating distributions across various business types. It also prevents the loss of information that would happen if we only used the primary category.

**Alternative solutions**

* A simpler option is to use the get\_primary\_category function to select just the first listed category for each business. While this approach is quicker and creates a simpler dataset, it overlooks secondary category links. As a result, it offers less detailed insights for businesses that fall into multiple types, like a "Café and Bookstore."

**Optimality of our solution**

* This solution works best for Exploratory Data Analysis (EDA). By breaking down categories and filtering to include only popular ones (≥100 reviews), we enhance our analytical depth while keeping statistical significance. The resulting dataset of multi-category associations captures user patterns more accurately than focusing solely on the primary category. This makes it very useful for understanding category-specific trends.

### Answer 1.7.2 Analysis of Low-Rating Reviews

#### Code

|  |
| --- |
| if 'analysis\_df' in locals():      print(" PREPARING LOWER RATING ANALYSIS")      print("=" \* 40)        # Filter for lower ratings (1-2 stars)      low\_ratings = analysis\_df[analysis\_df['rating'] <= 2].copy()      print(f"Total low rating reviews (1-2 stars): {len(low\_ratings):,}")      print(f"Percentage of total reviews: {len(low\_ratings)/len(analysis\_df)\*100:.1f}%")        # Check for text column      text\_columns = [col for col in low\_ratings.columns if 'text' in col.lower() or 'review' in col.lower() or 'comment' in col.lower()]        if text\_columns:          text\_col = 'text'          print(f"\nUsing text column: '{text\_col}'")            # Filter for reviews with text          low\_ratings\_with\_text = low\_ratings[low\_ratings[text\_col].notna() & (low\_ratings[text\_col].str.len() > 10)].copy()          print(f"Low rating reviews with text: {len(low\_ratings\_with\_text):,}")          print(f"Text availability: {len(low\_ratings\_with\_text)/len(low\_ratings)\*100:.1f}%")            if len(low\_ratings\_with\_text) > 0:              # Basic text statistics              low\_ratings\_with\_text['text\_length'] = low\_ratings\_with\_text[text\_col].str.len()              low\_ratings\_with\_text['word\_count'] = low\_ratings\_with\_text[text\_col].str.split().str.len()                print(f"\n Text Statistics for Low Ratings:")              print(f"Average text length: {low\_ratings\_with\_text['text\_length'].mean():.0f} characters")              print(f"Average word count: {low\_ratings\_with\_text['word\_count'].mean():.0f} words")              print(f"Median word count: {low\_ratings\_with\_text['word\_count'].median():.0f} words")                # Sample some low rating reviews              print(f"\n Sample Low Rating Reviews:")              sample\_reviews = low\_ratings\_with\_text.sample(min(10, len(low\_ratings\_with\_text)))              for idx, row in sample\_reviews.iterrows():                  print(f"\n {row['rating']} stars - {row['clean\_category']}:")                  print(f"   {row[text\_col][:200]}{'...' if len(row[text\_col]) > 200 else ''}")          else:              print("No low rating reviews with sufficient text found")              low\_ratings\_with\_text = None      else:          print(" No text column found for review analysis")          low\_ratings\_with\_text = None          text\_col = None        # Category breakdown of low ratings      print(f"\n Low Ratings by Category:")      low\_rating\_by\_category = low\_ratings['clean\_category'].value\_counts()      low\_rating\_percentages = (low\_rating\_by\_category / analysis\_df['clean\_category'].value\_counts() \* 100).round(1)        category\_low\_rating\_analysis = pd.DataFrame({          'Low\_Rating\_Count': low\_rating\_by\_category,          'Low\_Rating\_Percentage': low\_rating\_percentages      }).fillna(0).sort\_values('Low\_Rating\_Percentage', ascending=False)        print(category\_low\_rating\_analysis.head(10).to\_string())  else:      print(" Analysis data not available") |

|  |
| --- |
| # Text analysis of low rating reviews  if 'low\_ratings\_with\_text' in locals() and low\_ratings\_with\_text is not None and text\_col is not None:      print(" TEXT ANALYSIS OF LOW RATING REVIEWS")      print("=" \* 45)        # Text preprocessing function      def preprocess\_text(text):          """Clean and preprocess text for analysis"""          if pd.isna(text):              return ""            # Convert to lowercase          text = str(text).lower()            # Remove special characters but keep spaces          text = re.sub(r'[^a-zA-Z\s]', ' ', text)            # Remove extra whitespace          text = ' '.join(text.split())            return text        # Preprocess all low rating review texts      low\_ratings\_with\_text['clean\_text'] = low\_ratings\_with\_text[text\_col].apply(preprocess\_text)        # Extract all words      all\_words = []      for text in low\_ratings\_with\_text['clean\_text']:          words = text.split()          # Filter out very short words and common stop words          stop\_words = {'the', 'and', 'or', 'but', 'in', 'on', 'at', 'to', 'for', 'of', 'with', 'by',                       'a', 'an', 'is', 'was', 'are', 'were', 'be', 'been', 'have', 'has', 'had',                       'do', 'does', 'did', 'will', 'would', 'could', 'should', 'may', 'might',                       'this', 'that', 'these', 'those', 'i', 'you', 'he', 'she', 'it', 'we', 'they',                       'my', 'your', 'his', 'her', 'its', 'our', 'their', 'me', 'him', 'her', 'us', 'them'}            filtered\_words = [word for word in words if len(word) > 2 and word not in stop\_words]          all\_words.extend(filtered\_words)        # Count word frequencies      word\_counts = Counter(all\_words)        print(f"\n MOST COMMON WORDS IN LOW RATING REVIEWS:")      print("=" \* 50)        top\_words = word\_counts.most\_common(30)      for i, (word, count) in enumerate(top\_words, 1):          percentage = count / len(all\_words) \* 100          print(f"{i:2d}. {word:15s} - {count:4d} times ({percentage:.1f}%)")        # Create word cloud      if len(word\_counts) > 0:          print("\n Generating word cloud...")            # Create word cloud          wordcloud = WordCloud(width=800, height=400,                               background\_color='white',                               max\_words=100,                               colormap='Reds').generate\_from\_frequencies(word\_counts)            # Display word cloud          plt.figure(figsize=(12, 6))          plt.imshow(wordcloud, interpolation='bilinear')          plt.axis('off')          plt.title('Word Cloud: Most Common Words in Low Rating Reviews', fontsize=16, fontweight='bold')          plt.tight\_layout()          plt.show()  else:      print(" Text analysis not possible - no text data available") |

|  |
| --- |
| if 'low\_ratings\_with\_text' in locals() and low\_ratings\_with\_text is not None and text\_col is not None:      print(" ADVANCED ANALYSIS: COMPLAINT PATTERNS")      print("=" \* 45)        # Define complaint categories based on common keywords      complaint\_categories = {          'Service Issues': ['service', 'staff', 'rude', 'slow', 'wait', 'waiting', 'employee', 'server', 'attitude'],          'Food Quality': ['food', 'taste', 'cold', 'burnt', 'stale', 'fresh', 'quality', 'flavor', 'meal'],          'Cleanliness': ['dirty', 'clean', 'hygiene', 'mess', 'filthy', 'sanitary', 'bathroom', 'table'],          'Price/Value': ['expensive', 'price', 'money', 'cost', 'value', 'overpriced', 'cheap', 'worth'],          'Location/Accessibility': ['location', 'parking', 'access', 'far', 'close', 'traffic', 'address'],          'Wait Time': ['wait', 'time', 'long', 'quick', 'fast', 'delay', 'hour', 'minute'],          'Product/Item Issues': ['product', 'item', 'broken', 'defective', 'wrong', 'missing', 'damaged']      }        # Analyze complaint patterns      complaint\_analysis = {}        for category, keywords in complaint\_categories.items():          # Count reviews mentioning keywords from this category          pattern = '|'.join(keywords)          mentions = low\_ratings\_with\_text['clean\_text'].str.contains(pattern, case=False, na=False).sum()          percentage = mentions / len(low\_ratings\_with\_text) \* 100          complaint\_analysis[category] = {'count': mentions, 'percentage': percentage}        # Create DataFrame for better visualization      complaint\_df = pd.DataFrame(complaint\_analysis).T      complaint\_df = complaint\_df.sort\_values('percentage', ascending=False)        print("\n COMPLAINT CATEGORY ANALYSIS:")      print("=" \* 40)      print(f"{'Category':<20} {'Count':<8} {'Percentage':<12}")      print("-" \* 40)        for category, row in complaint\_df.iterrows():          print(f"{category:<20} {row['count']:<8.0f} {row['percentage']:<12.1f}%")        # Visualize complaint categories      plt.figure(figsize=(12, 8))        # Bar plot of complaint categories      plt.subplot(2, 1, 1)      bars = plt.bar(complaint\_df.index, complaint\_df['percentage'], color='lightcoral')      plt.title('Complaint Categories in Low Rating Reviews', fontsize=14, fontweight='bold')      plt.ylabel('Percentage of Reviews (%)')      plt.xticks(rotation=45, ha='right')        # Add value labels on bars      for bar, percentage in zip(bars, complaint\_df['percentage']):          plt.text(bar.get\_x() + bar.get\_width()/2, bar.get\_height() + 0.5,                  f'{percentage:.1f}%', ha='center', va='bottom')        # Pie chart of top complaint categories      plt.subplot(2, 1, 2)      top\_complaints = complaint\_df.head(5)      plt.pie(top\_complaints['percentage'], labels=top\_complaints.index, autopct='%1.1f%%',              colors=['lightcoral', 'lightsalmon', 'peachpuff', 'moccasin', 'wheat'])      plt.title('Top 5 Complaint Categories Distribution', fontsize=14, fontweight='bold')        plt.tight\_layout()      plt.show()        # Category-specific complaint analysis      print("\n\n COMPLAINT PATTERNS BY BUSINESS CATEGORY:")      print("=" \* 50)        category\_complaint\_analysis = {}        for business\_cat in low\_ratings\_with\_text['clean\_category'].unique():          cat\_reviews = low\_ratings\_with\_text[low\_ratings\_with\_text['clean\_category'] == business\_cat]          if len(cat\_reviews) >= 10:  # Only analyze categories with sufficient data              cat\_complaints = {}              for complaint\_cat, keywords in complaint\_categories.items():                  pattern = '|'.join(keywords)                  mentions = cat\_reviews['clean\_text'].str.contains(pattern, case=False, na=False).sum()                  percentage = mentions / len(cat\_reviews) \* 100                  cat\_complaints[complaint\_cat] = percentage                # Find top complaint for this business category              top\_complaint = max(cat\_complaints, key=cat\_complaints.get)              category\_complaint\_analysis[business\_cat] = {                  'top\_complaint': top\_complaint,                  'percentage': cat\_complaints[top\_complaint],                  'review\_count': len(cat\_reviews)              }        # Display results      for business\_cat, analysis in sorted(category\_complaint\_analysis.items(),                                         key=lambda x: x[1]['percentage'], reverse=True):          print(f"{business\_cat:25s} - Top Issue: {analysis['top\_complaint']:15s} "                f"({analysis['percentage']:.1f}% of {analysis['review\_count']} reviews)")  else:      print(" Advanced analysis not possible - no text data available") |

#### Visualizations

|  |
| --- |
| Figure 5 Word clouds showing common words in low ratings |

|  |
| --- |
| Figure 6 Top Negative Themes in Reviews |

#### Output

|  |
| --- |
| PREPARING LOWER RATING ANALYSIS  ========================================  Total low rating reviews (1-2 stars): 65,237  Percentage of total reviews: 8.9%  Using text column: 'text'  Low rating reviews with text: 41,318  Text availability: 63.3%  Text Statistics for Low Ratings:  Average text length: 191 characters  Average word count: 36 words  Median word count: 21 words  Sample Low Rating Reviews:  1 stars - Restaurant:  There was a dirty cup and dirty silverware, the server was very polite and switched it out however. The food was good but not worth the money spent. Have been there several times and it is always very...  1 stars - Department store:  Way way way overpriced for a Kohl's I live in Seattle Washington and I went up here to visit for Christmas because why not rent out of place in a snowy Wonderland for Christmas well Alaska ended up ju...  1 stars - Restaurant:  We were denied entry because we did not have masks. Not one customer in the establishment was wearing masks, but because they had masks, they were allowed in. I have never experienced anything so du...  2 stars - Restaurant:  3 stars because they grill everything perfectly, crunchy but no burnt taste, not too soft either. I'm tempted to give 2 because this is the one Taco Bell in town that either makes errors or forgets it...  1 stars - Restaurant:  McDonald's would have been better. Taco Bell also would have been better  1 stars - Restaurant:  This particular location was horrible. Walked in the front door and the only employee there didn't turn to greet or even see who was there. She continued to ignore me, favoring trying to refill their ...  1 stars - Coffee shop:  Found a hair in my food  1 stars - Restaurant:  9-4-2020  Wow went there three times in a few month span and messed up most of the orders even the fix they messed up on. Never going here again.  1 stars - Restaurant:  Place was trashed garbage all over parking lot, trash inside was full, tables were dirty. Went threw drive through and ordered 2 extra value meals and they messed up order and I had to go in. Crew was...  2 stars - Grocery store:  was going to get groceries and bedding , got to the meat department and the first package of meat I picked up(angus) , was green on edges and all over the top, very dried out looking, was growing som...  Low Ratings by Category:  Low\_Rating\_Count Low\_Rating\_Percentage  clean\_category  Sandwich shop 2877 11.5  Clothing store 2042 11.1  Department store 1891 10.2  Coffee shop 2915 9.8  Supermarket 1631 9.8  Restaurant 46948 9.2  Sporting goods store 1717 9.2  Grocery store 3026 8.5  Bar 1414 5.8  Tourist attraction 776 1.9    Figure 7 Principal Features of Negative Review Text    Figure 8 Key Issues Across Main Business Segments    Figure 9 Breakdown of Negative Themes per Business Category |
|  |

#### Insights

**Low-Rating Proportion**

* Total low-rating reviews: 65,237 (8.9% of all reviews).
* Text is available for 41,318 reviews (63%), with an average length of 36 words.

**Categories with Most Low Ratings**

* Highest low-rating percentages: Sandwich shop (11.5%), Clothing store (11.1%), Department store (10.2%).
* Lowest low-rating percentages: Tourist attraction (1.9%) and Bar (5.8%).

**Common Complaint Themes**

* Food Quality: This is the most common issue overall (41%) and the top complaint for Bar, Restaurant, and Sandwich shop.
* Service Issues: These are a major concern in Coffee shop, Department store, Grocery store, and Clothing store (31-35%).
* Wait Time: This is especially notable for Tourist attraction (16.4%).
* Other complaints include Price/Value (16.7%), Cleanliness (11.6%), and Product/Item Issues (8.5%).

**Frequent Words in Low-Rating Reviews**

* Common words include: food, not, service, order, time, good, back, never.
* This shows a primary focus on food quality, service, and order problems.

**Actionable Takeaways**

* Food-related businesses (restaurants, sandwich shops, bars) should work on improving food quality and order accuracy.
* Service issues (staff behaviour, responsiveness, wait times) are crucial for coffee shops, department stores, and grocery stores.

#### Explanation

**Why we chose this solution**

* The analysis looks at 1- and 2-star reviews to pinpoint key areas where the business is failing. We start by cleaning and processing the review texts. Then, we count word frequencies and create a Word Cloud to show the most common words. We also categorize complaints based on keywords, identifying main drivers of negative feedback, like "service" or "food quality." This method gives clear insights for stakeholders in a simple, understandable way.

**Alternative solutions**

* Other possible approaches include:
  + Sentiment Scoring: Giving sentiment scores to each review with lexicon-based or pre-trained sentiment models to measure negative sentiment.
  + N-gram Analysis: Looking at bigrams or trigrams instead of single words to capture more context-specific phrases like "poor service" or "cold food."
  + Machine Learning Classification: Using supervised learning models to sort reviews into preset complaint categories based on text features.
* These methods can offer deeper insights or more detailed analysis. However, they often need more data processing, computing power, or labelled training data.

**Optimality of our solution**

* Our keyword-based method is quick and straightforward. It aligns with project goals and offers practical insights into important business complaints. This approach avoids the complications of complex machine learning models.

## Answer 1.8

### Answer 1.8.1 Compile User Business History

#### Code

|  |
| --- |
| if 'merged\_df' in locals():      time\_col = 'newtime'      print(" ANALYZING REVIEWER BUSINESS HISTORY")      print("=" \* 50)        # Create user business history      print(f" Creating chronological business lists for each reviewer...")        user\_business\_list = {}      user\_id\_to\_name = {}        # Group by user and process each user's reviews      for user\_id, user\_reviews in merged\_df.groupby('user\_id'):          user\_reviews\_sorted = user\_reviews.sort\_values(time\_col)          business\_names = user\_reviews\_sorted['name\_business'].tolist()          user\_business\_list[user\_id] = business\_names          user\_id\_to\_name[user\_id] = user\_reviews\_sorted['name\_review'].iloc[0]        print(f" Created business history for {len(user\_business\_list):,} reviewers")        # Display statistics      business\_counts = [len(businesses) for businesses in user\_business\_list.values()]        print(f"\n REVIEWER ACTIVITY STATISTICS:")      print(f"Total reviewers: {len(user\_business\_list):,}")      print(f"Average businesses per reviewer: {np.mean(business\_counts):.1f}")      print(f"Median businesses per reviewer: {np.median(business\_counts):.1f}")      print(f"Max businesses reviewed by single user: {max(business\_counts):,}")      print(f"Min businesses reviewed by single user: {min(business\_counts):,}")      print(f"\n SAMPLE REVIEWER BUSINESS HISTORIES:")      sample\_users = list(user\_business\_list.keys())[:3]      for i, user\_id in enumerate(sample\_users, 1):          businesses = user\_business\_list[user\_id]          user\_name = user\_id\_to\_name[user\_id]          print(f"\n{i}. User :=> {user\_name}({user\_id}) :")          print(f"   Total businesses: {len(businesses)}")          print(f"   Business sequence: {businesses[:5]}{'...' if len(businesses) > 5 else ''}")    if 'merged\_df' in locals():          # Distribution of review counts      plt.figure(figsize=(15, 6))        plt.subplot(1, 3, 1)      plt.hist(business\_counts, bins=50, alpha=0.7, color='skyblue', edgecolor='black')      plt.xlabel('Number of Businesses Reviewed')      plt.ylabel('Number of Reviewers')      plt.title('Distribution of Business Review Counts\nper Reviewer (All)')      plt.yscale('log')        plt.subplot(1, 3, 2)      # Focus on reviewers with reasonable activity (1-50 businesses)      filtered\_counts = [count for count in business\_counts if 1 <= count <= 50]      plt.hist(filtered\_counts, bins=30, alpha=0.7, color='lightcoral', edgecolor='black')      plt.xlabel('Number of Businesses Reviewed (1-50)')      plt.ylabel('Number of Reviewers')      plt.title('Distribution (Filtered: 1-50 businesses)')        plt.subplot(1, 3, 3)      # Box plot for better understanding      plt.boxplot(business\_counts, vert=True)      plt.ylabel('Number of Businesses Reviewed')      plt.title('Box Plot of Review Activity')      plt.yscale('log')        plt.tight\_layout()      plt.show() |

#### Visualization

|  |
| --- |
|  |

#### Output

|  |
| --- |
| ANALYZING REVIEWER BUSINESS HISTORY  ==================================================  Creating chronological business lists for each reviewer...  Created business history for 20,022 reviewers  REVIEWER ACTIVITY STATISTICS:  Total reviewers: 20,022  Average businesses per reviewer: 25.8  Median businesses per reviewer: 17.0  Max businesses reviewed by single user: 394  Min businesses reviewed by single user: 8  SAMPLE REVIEWER BUSINESS HISTORIES:  1. User :=> orlando taporco(1.0000266958784964e+20) :  Total businesses: 22  Business sequence: ["McDonald's", 'New Sagaya Midtown Market', 'Lucky Market', 'Walmart Supercenter', 'Extended Stay America - Anchorage - Midtown']...  2. User :=> Erica Hill(1.00003825755859e+20) :  Total businesses: 16  Business sequence: ['Yes Bistro', 'Pita Place', 'Old Town Diner', 'The Banks Alehouse', "Kava's Pancake House"]...  3. User :=> M Ric(1.0000428139011082e+20) :  Total businesses: 10  Business sequence: ['West Valley Plaza', 'Fred Meyer', 'Big Dipper Ice Arena', 'Blue Loon', 'Safeway']... |

#### Insights

* Total number of reviewers: 20,022; average business reviews per reviewer: ~26.
* Review activity varies greatly, min = 8 to max = 394 businesses per reviewer.
* The lists of businesses are ordered and recorded chronologically, allowing for trends, loyalty, and engagement metrics to be analyzed.
* The samples of sequences of how reviewers interact with a variety of chain businesses, local businesses, and specialty businesses indicates the user behaviors were similar.
* The user-business-list allows for the detection of repeated reviews, accounts of similarities across reviewers, and recommendations building trigger.

#### Explanation

**Why you decided to choose your solution**

* We used pandas.groupby('user\_id') with sort\_values(time\_col) and a dictionary (user\_business\_list). This efficiently creates chronological business sequences for each reviewer. It is simple, scalable, and maintains the correct review order for all users.

**Are there any other solutions that could solve the question**

* Parallel frameworks like Spark could speed up grouping and sorting for large datasets, but require additional setup and resources. A manual iteration through the DataFrame is also possible but slower and more error-prone.

**Whether your solution is the optimal or not? why?**

* The solution is near-optimal for a standard single-machine Python setup. It efficiently leverages pandas, but its main limitation is memory usage, as storing the full merged\_df and user\_business\_list dictionary can be demanding. For much larger datasets, this approach may become impractical, and a distributed framework like Spark or Dask would be more suitable. For the current dataset, however, pandas groupby provides the best balance between simplicity, readability, and performance.

### Answer 1.8.2 Remove Duplicate Businesses per User

#### Code

|  |
| --- |
| if 'user\_business\_list' in locals():      print(" ANALYZING DUPLICATE BUSINESSES IN USER HISTORIES")      print("=" \* 55)        # Create deduplicated version      user\_business\_list\_deduplicated = {}      duplicate\_stats = []        print("Processing user business lists...")        for user\_id, business\_list in user\_business\_list.items():          # Original list length          original\_length = len(business\_list)            # Remove duplicates while preserving order (first occurrence)          seen = set()          deduplicated\_list = []          for business in business\_list:              if business not in seen:                  seen.add(business)                  deduplicated\_list.append(business)            # Store deduplicated list          user\_business\_list\_deduplicated[user\_id] = deduplicated\_list            # Calculate statistics          deduplicated\_length = len(deduplicated\_list)          duplicates\_removed = original\_length - deduplicated\_length          duplicate\_percentage = (duplicates\_removed / original\_length \* 100) if original\_length > 0 else 0            duplicate\_stats.append({              'user\_id': user\_id,              'original\_length': original\_length,              'deduplicated\_length': deduplicated\_length,              'duplicates\_removed': duplicates\_removed,              'duplicate\_percentage': duplicate\_percentage          })        # Convert to DataFrame for analysis      duplicate\_stats\_df = pd.DataFrame(duplicate\_stats)    else:      print(" User business list not available for duplicate analysis")  if 'user\_business\_list' in locals():      print(f"\n DUPLICATE ANALYSIS RESULTS:")      print(f"Total users analyzed: {len(user\_business\_list):,}")        # Overall statistics      total\_original = duplicate\_stats\_df['original\_length'].sum()      total\_deduplicated = duplicate\_stats\_df['deduplicated\_length'].sum()      total\_duplicates = duplicate\_stats\_df['duplicates\_removed'].sum()        print(f"\n OVERALL STATISTICS:")      print(f"Total business visits (original): {total\_original:,}")      print(f"Unique business visits (deduplicated): {total\_deduplicated:,}")      print(f"Total duplicate visits removed: {total\_duplicates:,}")      print(f"Overall duplicate percentage: {total\_duplicates/total\_original\*100:.1f}%")        # User-level statistics      users\_with\_duplicates = duplicate\_stats\_df[duplicate\_stats\_df['duplicates\_removed'] > 0]        print(f"\n USER-LEVEL STATISTICS:")      print(f"Users with duplicate visits: {len(users\_with\_duplicates):,} ({len(users\_with\_duplicates)/len(duplicate\_stats\_df)\*100:.1f}%)")      print(f"Users with no duplicates: {len(duplicate\_stats\_df) - len(users\_with\_duplicates):,}")        if len(users\_with\_duplicates) > 0:          print(f"Average duplicates per user (among users with duplicates): {users\_with\_duplicates['duplicates\_removed'].mean():.1f}")          print(f"Max duplicates by single user: {users\_with\_duplicates['duplicates\_removed'].max():,}")          print(f"Average duplicate percentage per user: {users\_with\_duplicates['duplicate\_percentage'].mean():.1f}%")        # Show examples      print(f"\nEXAMPLES OF USERS WITH DUPLICATES:")      top\_duplicate\_users = users\_with\_duplicates.nlargest(3, 'duplicates\_removed')        for idx, row in top\_duplicate\_users.iterrows():          user\_id = row['user\_id']          user\_name = user\_id\_to\_name[user\_id]          print(f"\nUser :=> {user\_name} ({user\_id}):")          print(f"  Original list length: {row['original\_length']}")          print(f"  After removing duplicates: {row['deduplicated\_length']}")          print(f"  Duplicates removed: {row['duplicates\_removed']} ({row['duplicate\_percentage']:.1f}%)")            # Show original vs deduplicated          original\_list = user\_business\_list[user\_id][:10]          deduplicated\_list = user\_business\_list\_deduplicated[user\_id][:10]          print(f"  Original (first 10): {original\_list}")          print(f"  Deduplicated (first 10): {deduplicated\_list}")    else:      print(" User business list not available for duplicate analysis")  # Visualization for duplicate analysis  if 'duplicate\_stats\_df' in locals():      print(" VISUALIZING DUPLICATE PATTERNS")      print("=" \* 35)        plt.figure(figsize=(15, 10))        # Subplot 1: Distribution of duplicate percentages      plt.subplot(2, 3, 1)      plt.hist(duplicate\_stats\_df['duplicate\_percentage'], bins=30, alpha=0.7, color='orange', edgecolor='black')      plt.xlabel('Duplicate Percentage (%)')      plt.ylabel('Number of Users')      plt.title('Distribution of Duplicate Percentages')        # Subplot 2: Original vs Deduplicated lengths      plt.subplot(2, 3, 2)      plt.scatter(duplicate\_stats\_df['original\_length'], duplicate\_stats\_df['deduplicated\_length'],                 alpha=0.6, s=20)      plt.plot([0, duplicate\_stats\_df['original\_length'].max()], [0, duplicate\_stats\_df['original\_length'].max()],               'r--', label='No duplicates line')      plt.xlabel('Original List Length')      plt.ylabel('Deduplicated List Length')      plt.title('Original vs Deduplicated Lengths')      plt.legend()        # Subplot 3: Number of duplicates removed      plt.subplot(2, 3, 3)      plt.hist(duplicate\_stats\_df['duplicates\_removed'], bins=30, alpha=0.7, color='red', edgecolor='black')      plt.xlabel('Number of Duplicates Removed')      plt.ylabel('Number of Users')      plt.title('Distribution of Duplicates Removed')      plt.yscale('log')        # Subplot 4: Relationship between list length and duplicate percentage      plt.subplot(2, 3, 4)      plt.scatter(duplicate\_stats\_df['original\_length'], duplicate\_stats\_df['duplicate\_percentage'],                 alpha=0.6, s=20, color='purple')      plt.xlabel('Original List Length')      plt.ylabel('Duplicate Percentage (%)')      plt.title('List Length vs Duplicate Percentage')        # Subplot 5: Box plot of duplicates by activity level      plt.subplot(2, 3, 5)      # Create activity level categories      duplicate\_stats\_df['activity\_level'] = pd.cut(duplicate\_stats\_df['original\_length'],                                                    bins=[0, 5, 15, 50, float('inf')],                                                    labels=['Low (1-5)', 'Medium (6-15)', 'High (16-50)', 'Very High (50+)'])        sns.boxplot(data=duplicate\_stats\_df, x='activity\_level', y='duplicate\_percentage')      plt.xlabel('Activity Level')      plt.ylabel('Duplicate Percentage (%)')      plt.title('Duplicates by Activity Level')      plt.xticks(rotation=45)        # Subplot 6: Summary statistics      plt.subplot(2, 3, 6)      summary\_data = [          duplicate\_stats\_df['duplicate\_percentage'].mean(),          duplicate\_stats\_df['duplicate\_percentage'].median(),          duplicate\_stats\_df[duplicate\_stats\_df['duplicates\_removed'] > 0]['duplicate\_percentage'].mean()      ]      labels = ['Overall Mean', 'Overall Median', 'Mean (Users with Duplicates)']        plt.bar(labels, summary\_data, color=['skyblue', 'lightgreen', 'coral'])      plt.ylabel('Duplicate Percentage (%)')      plt.title('Summary Statistics')      plt.xticks(rotation=45)        plt.tight\_layout()      plt.show()    else:      print(" Duplicate statistics not available for visualization") |

#### Visualization

|  |
| --- |
| Figure 10 Impact of Duplicate Reviews on Reviewer History Metrics |

#### Output

|  |
| --- |
| DUPLICATE ANALYSIS RESULTS:  Total users analyzed: 20,022  OVERALL STATISTICS:  Total business visits (original): 516,289  Unique business visits (deduplicated): 492,494  Total duplicate visits removed: 23,795  Overall duplicate percentage: 4.6%  USER-LEVEL STATISTICS:  Users with duplicate visits: 7,132 (35.6%)  Users with no duplicates: 12,890  Average duplicates per user (among users with duplicates): 3.3  Max duplicates by single user: 105  Average duplicate percentage per user: 7.9%  EXAMPLES OF USERS WITH DUPLICATES:  User :=> Naynaylovesmakeup (1.0188830200557922e+20):  Original list length: 394.0  After removing duplicates: 289.0  Duplicates removed: 105.0 (26.6%)  Original (first 10): ['Fred Meyer', 'Anchorage 5th Avenue Mall', 'Midtown Mall', 'Dimond Center', 'Jalapenos Mexican Restaurant & Cantina', 'Walmart Supercenter', 'Palmer Veterinary Clinic', 'Grizzly Family Fitness', 'Donut King', 'Pioneer Pizza']  Deduplicated (first 10): ['Fred Meyer', 'Anchorage 5th Avenue Mall', 'Midtown Mall', 'Dimond Center', 'Jalapenos Mexican Restaurant & Cantina', 'Walmart Supercenter', 'Palmer Veterinary Clinic', 'Grizzly Family Fitness', 'Donut King', 'Pioneer Pizza']  User :=> DJ Wallace-Blatchford (1.03692833808364e+20):  Original list length: 304.0  After removing duplicates: 236.0  Duplicates removed: 68.0 (22.4%)  Original (first 10): ['Verizon Authorized Retailer - GoWireless', 'Regal Totem', 'Natural Pantry', 'Thai Village Restaurant', 'The Flats Bistro', 'The Cookery', 'Once Upon A Child Anchorage', "Ray's Waterfront", 'Apollo Restaurant', 'Hops Hallmark Shop']  Deduplicated (first 10): ['Verizon Authorized Retailer - GoWireless', 'Regal Totem', 'Natural Pantry', 'Thai Village Restaurant', 'The Flats Bistro', 'The Cookery', 'Once Upon A Child Anchorage', "Ray's Waterfront", 'Apollo Restaurant', 'Hops Hallmark Shop']  User :=> Chris Tadda (1.1220090844478289e+20):  Original list length: 311.0  After removing duplicates: 250.0  Duplicates removed: 61.0 (19.6%)  Original (first 10): ['Kendall Ford of Anchorage', 'Walmart Supercenter', "McDonald's", 'The Motorcycle Shop', 'Chugach State Park', 'Delaney Park', 'Cuddy Family Midtown Park', 'Golden Corral Buffet & Grill', 'Walmart Supercenter', 'Midtown Mall']  Deduplicated (first 10): ['Kendall Ford of Anchorage', 'Walmart Supercenter', "McDonald's", 'The Motorcycle Shop', 'Chugach State Park', 'Delaney Park', 'Cuddy Family Midtown Park', 'Golden Corral Buffet & Grill', 'Midtown Mall', 'Magic Wand Car Wash']  VISUALIZING DUPLICATE PATTERNS  ===================================  IMPLEMENTING USER SIMILARITY ANALYSIS  ==================================================  Users with ≥3 businesses: 20,022 (from 20,022 total)  APPROACH 1: JACCARD SIMILARITY ANALYSIS  =============================================  Calculating Jaccard similarities for sample of 500 users...  Found 58,278 user pairs with shared businesses  Top 10 most similar user pairs (Jaccard):  Users Don(1.0131743591050556e+20) & Carl Redding(1.020477549205772e+20) : 0.333 (9.0 shared / 27.0 total)  Users Aiden 2345 verney.(1.10463128812272e+20) & Emily Stanker(1.0637785892171973e+20) : 0.316 (6.0 shared / 19.0 total)  Users Melisa Olson(1.0001748952137774e+20) & Carl Redding(1.020477549205772e+20) : 0.306 (11.0 shared / 36.0 total) |

#### Insights

* **Duplicate Occurrence:**Of 20,022 users, 7,132 (35.6%) experienced duplicate visits to the same businesses in their review history.
* **Volume of Duplicates:** In total, we removed 23,795 duplicate visits, which is 4.6% of the total visits to businesses after logging all visits.
* **User Impact:** For used affected by duplicate reviews, the average number of duplicates was 3.3, with the highest number of duplicates for an individual user at 105.
* **Effect on Business Lists:** Removing duplicates impacted the user\_business\_list length for affected users, so that each business is listed once.
* **Example Patterns:** High-activity users returned to popular businesses (e.g., Walmart, McDonald's) repeatedly, showing their interest in these businesses, but counted as visits that inflated raw review totals.
* **Key Takeaway**: Deduplication prepares the data for subsequent analysis, and provides assurance that user-business sequences are based on a unique user-business interaction, rather than a repeat visit to a business.

#### Explanation

**Why we chose this solution**

* The solution eliminates duplicate business names while maintaining the order of the initial visit by using a set-based strategy with list traversal. For sequential recommender systems, where item order is important, this is essential. The requirement for a clear, non-redundant sequence is met by iterating through the original time-sorted list and only adding a business name to the new list if it hasn't been seen (checked against the fast-lookup set). This keeps the history chronologically based on the first visit to that business.

**Are there any other solutions that could solve the question**

* Yes. In Python an easier, one-line solution for removing duplicates is to convert the list to a dict.fromkeys(business\_list) and then back to a list. This is faster but less explicit about the order-preserving mechanism. Another easy solution would be converting the original list to a pandas Series and using the .drop\_duplicates(keep='first') method, which has a very clean and readable form, but will introduce the computational overhead of converting between standard Python data structures and a pandas Series on repeated calls inside the loop.

**Whether your solution is the optimal or not? why?**

* In a typical Python environment, the solution offers the best performance and clarity. The average complexity for determining whether a business has been viewed is O(1) (constant time) when we use a set for lookup. Because of this, the entire process is quick: the time complexity for a list of length N is O(N). Compared to list comprehensions or nested loops, which produce slower O(N 2) complexity, this linear approach is far superior. It successfully strikes a balance between memory usage, speed, and the fundamental need to maintain the first chronological event.

### Answer 1.8.3 Analyse User Similarities Based on Reviewed Businesses

#### Code

|  |
| --- |
|  |

#### Visualization

|  |
| --- |
|  |

#### Output

|  |
| --- |
|  |

#### Insights

#### Explanation

# Part 2

## Answer 2.1

## Answer 2.2

#### Code

|  |
| --- |
|  |

#### Visualization

|  |
| --- |
|  |

#### Output

|  |
| --- |
|  |

#### Insights

#### Explanation

## Answer 2.3