SIG742 – Modern Data Science

End-term Assessment

TEAM MEMBERS

Shrinidhi Choragi [Student ID: s225440182]

Chirag Arunkumar Vyas [Student ID: s225440315]

Sobana Sundararajan [Student ID: s225450167]

Table of Contents

I. Introduction

This report presents an end-to-end analysis and code implementation tackling a range of data-driven tasks using PySpark/Pandas. The project focuses on efficiently processing large datasets, performing aggregations, handling missing data, data exploration & analysis, recommendation systems, time series analysis, and extracting meaningful business insights through data visualization. Emphasizing collaborative problem-solving, each team member independently developed solutions and participated in group discussions to determine the most optimal approaches. The report details the logic behind each solution, alternative methods considered, and the rationale for chosen strategies.

II. Part I Data Acquisition and Manipulation

- Step 1: Data Acquisition and Downloading
- Step 2: Installation of Spark Session
- Step 3: Load Data using PySpark
- Step 4: Understanding Data
- Step 5: Understanding the level of the data to be analyzed.

Before starting the exercises, it was essential to download, extract, and load the data, as well as to understand the data structure, granularity, data types, and schema.

Key observations from the analysis:

The reviews dataset is recorded at the user ID and timestamp level, capturing both single and multiple reviews per user. The meta dataset contains business details, such as gmap_id, ratings, average rating, location, address, category, and other relevant attributes.

1.1. Missing Reviews Handling and Time Conversion

```
# Replace null or 'none' in the text column with 'no review'

joined_df_clean = joined_df.withColumn(
    "text",
    when((col("text").isNull()) | (col("text") == "none"), "no review")
    .otherwise(col("text"))
)
```

This approach utilizes Spark's distributed processing to efficiently handle missing values, using the *when-otherwise* clause to impute "no review" where needed. Checked for missing or None values in the text column of the reviews. Approximately 43% of reviews lacked any textual content. This method was chosen for its efficiency and native support in Spark, making it well-suited for large distributed datasets. Alternative approaches like UDFs exist but are less efficient. Given Spark's optimized columnar operations, this solution is practical and optimal for this use case.

```
# Convert Unix timestamp (ms) → yyyy-MM-dd
joined_df_clean = joined_df_clean.withColumn(
    "newtime",
    to_date(from_unixtime((col("time") / 1000).cast("bigint")))
)
```

Sample Output:

```
+-------------+----------+-----------------------------------------------------+
|time         |newtime   |text                                                 |
+-------------+----------+-----------------------------------------------------+
|1531178254647|2018-07-09|no review                                            |
|1524699287834|2018-04-25|I love this store and workers are helpful and nice :)|
|1531193250202|2018-07-10|Very pricey but okay                                 |
|1547172457761|2019-01-11|no review                                            |
|1570228925766|2019-10-04|no review                                            |
+-------------+----------+-----------------------------------------------------+
```

The solution converts Unix time in milliseconds to a readable date using from_unixtime and to_date, which is reliable for Spark DataFrames. This ensures the data are human-readable and ready for date-based analysis. Pandas pd.to_datetime could also be used for post-conversion for smaller datasets. PySpark's solution is preferred in this context for handling large volumes efficiently.

1.2 Analysis of Reviews by Business (gmap_id) and Review Timing

```
gmap_reviews = (
    joined_df_clean
    .groupBy("gmap_id", "business_name")
    .agg(F.max("num_of_reviews").cast("float").alias("review_count"))
    .orderBy(F.desc("review_count"))
)
```

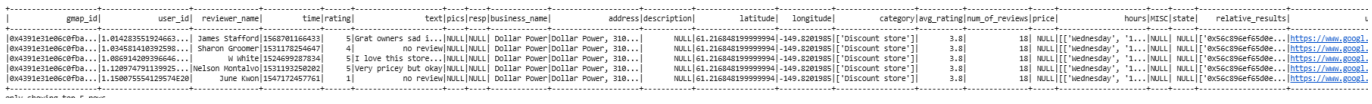
Sample output:

```
+-----------------------------------+---------------------------------+------------+
|gmap_id                            |business_name                    |review_count|
+-----------------------------------+---------------------------------+------------+
|0x56c897b9ce6000df:0xd707e127588a8c6c|Moose's Tooth Pub & Pizzeria     |6591.0      |
|0x54010304808eb99b:0x822af834f3f1722b|Tongass National Forest          |5812.0      |
|0x56c8bd86fd671871:0x52c896e66d960c02|49th State Brewing - Anchorage   |4227.0      |
|0x56c897c63697ee39:0x419904ababbc740b|Walmart Supercenter              |3547.0      |
|0x5400e187cf6c1f41:0x12ef6d1f6e7346cc|Mendenhall Glacier Visitor Center|3382.0      |
+-----------------------------------+---------------------------------+------------+
only showing top 5 rows
```

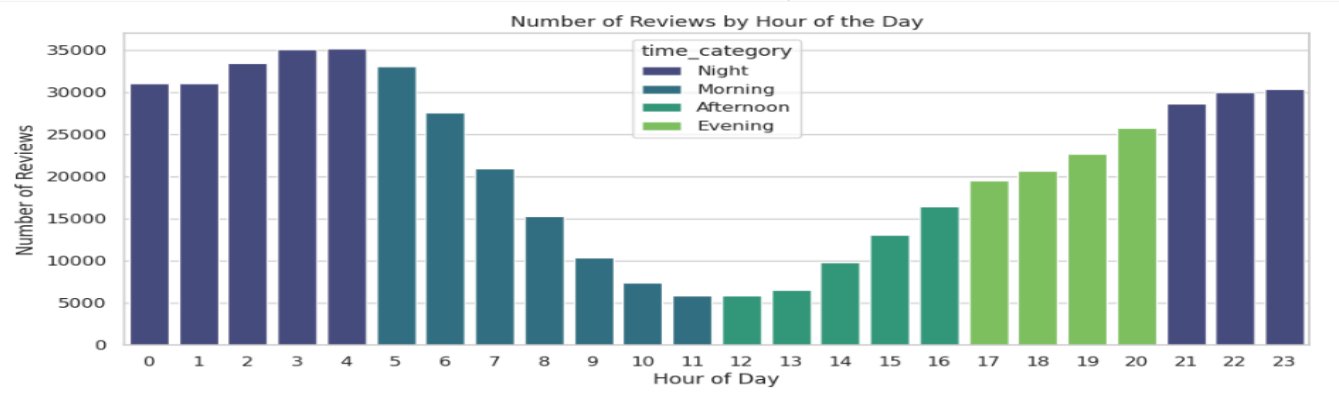
Grouped by gmap_id (or business_name since it's a unique mapping) to count reviews for each business using PySpark's groupBy and aggregation, which is fast for large datasets. Could use pandas groupby for small data or aggregate on business_name instead, but these methods are less efficient at big data scale. This approach is optimal for large data because native PySpark group and aggregation functions are distributed and efficient, while casting to float ensures data consistency.

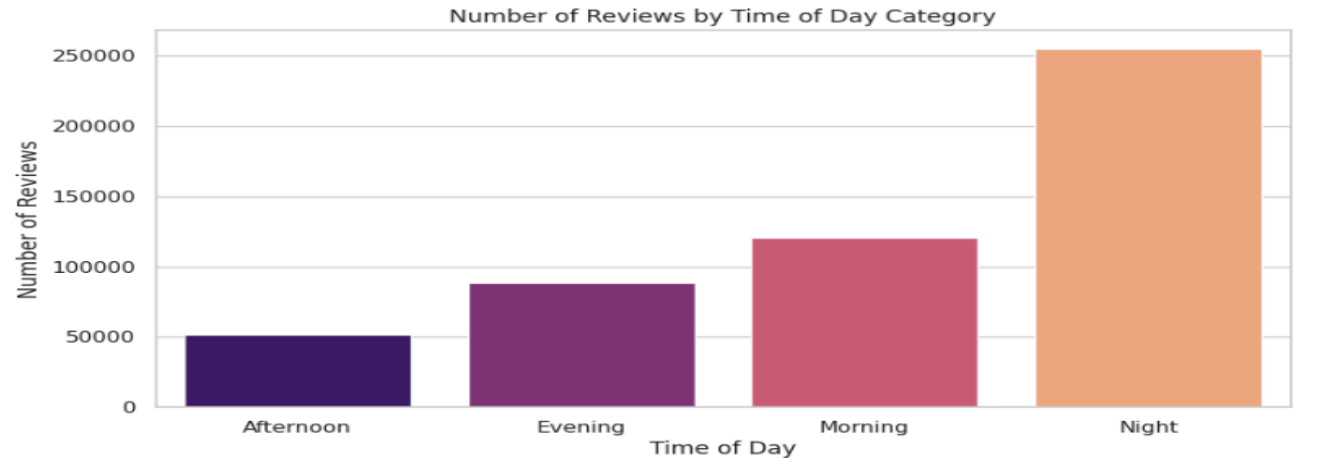
```
# Convert 'time' from string to numeric (milliseconds) & then to datetime
timestamp_df['time'] = pd.to_numeric(timestamp_df['time'], errors='coerce')
timestamp_df['review_datetime'] = pd.to_datetime(timestamp_df['time'] / 1000, unit='s')

# Extract hour from datetime
timestamp_df['review_time'] = timestamp_df['review_datetime'].dt.hour
```

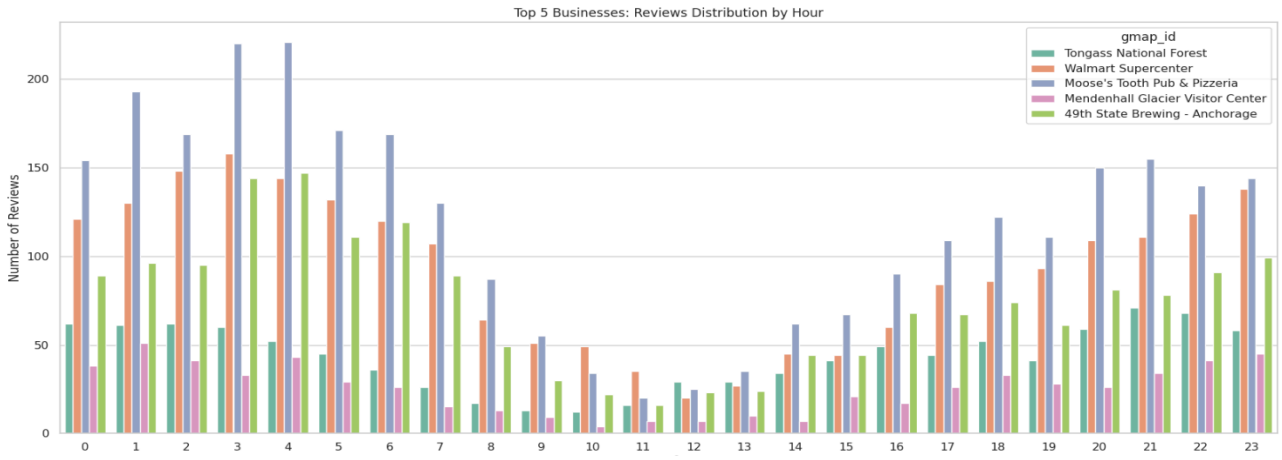
Sample Output:



Pandas is used here for convenient datetime extraction at hour precision, suitable for visualization and further analysis. Spark could do it, but Pandas excels in finer datetime operations for moderate data sizes, making this a hybrid optimal approach.







Insights:

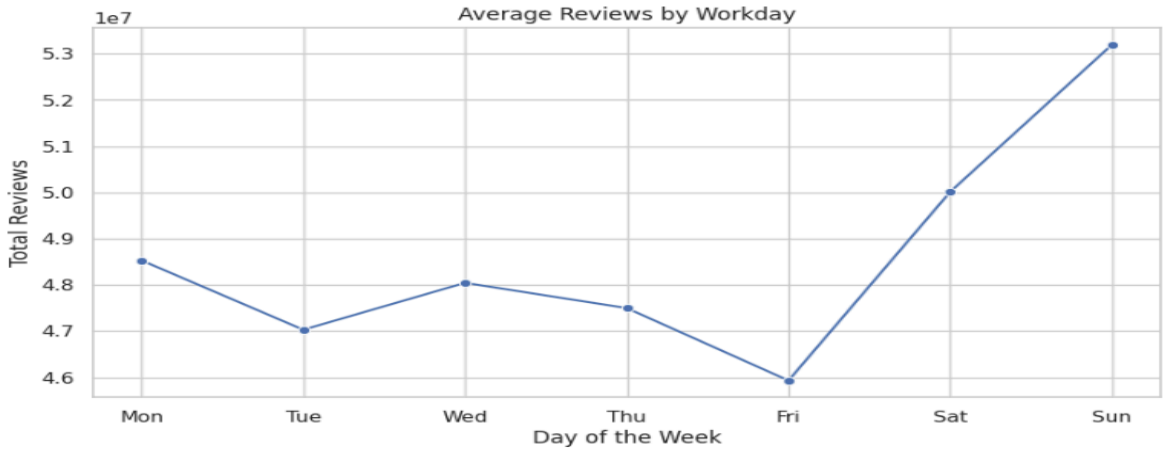
Number of reviews by hour: Displays peak hours when reviews are most frequently submitted. For instance, a visible peak around 2–3 suggests that many reviews occur in the early morning hours. Number of reviews by time-of-day: This indicates that most reviews are submitted during nighttime. Hourly review trends for the top 5 most-reviewed businesses (e.g., Walmart, pubs, etc.) reveal high activity during late-night hours, particularly around 2–3 AM.

1.3 Temporal Analysis of Reviews: Workdays, Business Categories, and Peak Hours

```
# Add a column for day of the week (Monday=1, Sunday=7)
joined_df_clean = joined_df_clean.withColumn("review_day", F.date_format("newtime", "E"))
joined_df_clean.show(5)
```

```
day_reviews = (
    joined_df_clean
    .groupBy("review_day")
    .agg(F.sum("num_of_reviews").alias("total_reviews"))
    .orderBy("review_day")  # Optional: keep chronological order
)
```

Output:



Insights:

Sunday records the highest average number of reviews across all businesses, indicating peak user engagement on weekends.

```
# Filter reviews for Sunday
sunday_reviews = joined_df_clean.filter(F.date_format("newtime", "E") == "Sun")
sunday_reviews.show(5)
```

```
# Aggregate average rating per business for Sundays
sunday_avg_rating = (
     sunday_reviews
     .groupBy("business_name", "category")
     .agg(F.avg("avg_rating").alias("avg_rating"))
     .orderBy(F.desc("avg_rating"))
)
```

Output: [Top 5 businesses with the highest average rating on Sunday]

```
+------------------------------+--------------------------------------------------------------------------------------------------+----------+
|business_name                 |category                                                                                          |avg_rating|
+------------------------------+--------------------------------------------------------------------------------------------------+----------+
|Black Spruce Dog Sledding     |['Dogsled ride service', 'Tourist attraction']                                                    |5.0       |
|Mark Knapp Custom Knives      |['Home goods store']                                                                              |5.0       |
|Frontier Safety And Supply, LLC|['Training centre']                                                                              |5.0       |
|Kayak Adventures Worldwide    |['Canoe & kayak tour agency', 'Visitor center', 'Whale watching tour agency']                     |5.0       |
|Muffy's Flowers & Gifts       |['Florist', 'Balloon store', 'Flower delivery', 'Flower market', 'Gift basket store', 'Gift shop']|5.0       |
+------------------------------+--------------------------------------------------------------------------------------------------+----------+
only showing top 5 rows
```

```
# Find the highest average rating - Understanding the categories of all the businesses with hightest average rating
max_rating = sunday_avg_rating.agg(F.max("avg_rating")).collect()[0][0]

# Filter all rows having that highest rating
top_rated_businesses = sunday_avg_rating.filter(F.col("avg_rating") == max_rating)
top_rated_businesses.show(truncate=False)
```

Output: [Business category with the highest rating on Sunday]

business_name	category	avg_rating
Anchorage Curling Club INC	['Stadium']	5.0
Black Spruce Dog Sledding	['Dogsled ride service', 'Tourist attraction']	5.0
Frontier Safety And Supply, LLC	['Training centre']	5.0
ESCAPE! Alaska	['Escape room center', 'Amusement center', 'Conference center', 'Corporate entertainment service', 'Event venue']	5.0
Kayak Adventures Worldwide	['Canoe & kayak tour agency', 'Visitor center', 'Whale watching tour agency']	5.0
Muffy's Flowers & Gifts	['Florist', 'Balloon store', 'Flower delivery', 'Flower market', 'Gift basket store', 'Gift shop']	5.0
Mark Knapp Custom Knives	['Home goods store']	5.0
Alaska Escape Rooms	['Escape room center']	5.0
Dragon Rays Tattooing	['Tattoo shop']	5.0
Allen Rapid Dry Carpet Cleaning (Pet Odor Experts!)	['Carpet cleaning service', 'Upholstery cleaning service']	5.0
HardLines Tattoo & Design	['Tattoo shop', 'Graphic designer', 'Tattoo artist']	5.0
Tim's Auto Glass	['Auto glass shop']	5.0
Snowhook Adventure Guides of Alaska	['Dogsled ride service', 'Helicopter tour agency', 'Tour operator']	5.0
Iron Asylum Gym AK	['Gym']	5.0
Paint Nights With Sara	['Art studio']	5.0
27 Red	['Hair salon']	5.0
PM Pediatrics	['Pediatrician']	5.0
Fairbanks Drama Assn & Fairbanks Children's Theatre	['Performing arts theater']	5.0
That Feeling Co	['Interior plant service', 'Coffee shop']	5.0
Josette's	['Hair salon']	5.0

only showing top 20 rows

Insights:

Businesses with the highest average ratings on Sundays primarily belong to the Tours & Travels and Florist Services categories, and likewise

Top 10 businesses based on reviews

```
# Get top 10 businesses by total reviews
business_stats_top10_no_explode = (
    business_stats_pd_no_explode
    .sort_values(by='total_reviews', ascending=False)
    .groupby('business_name')
    .head(1)   # take one record per business for top 10 selection
    .nlargest(10, 'total_reviews')
)
```

Number of reviews vs reviewed time of day

```
# Convert epoch in ms to timestamp and add review_hour
df_with_timestamp = (
    joined_df_clean
    .withColumn(
        "review_timestamp",
        F.from_unixtime((F.col("time") / 1000).cast("bigint")).cast("timestamp")
    )
    .withColumn(
        "review_hour",
        F.hour(F.from_unixtime((F.col("time") / 1000).cast("bigint")))
    )
)
```

```
# Aggregate using hour info
business_stats = (
    df_exploded
    .groupBy("business_name", "category_exploded")
    .agg(
        F.avg("num_of_reviews").alias("total_reviews"),
        F.avg("avg_rating").alias("avg_rating"),
        F.collect_list("review_hour").alias("review_hours")   # now uses proper hour data
    )
)
```

Total Reviews based on business category.

```
# Aggregate total reviews per category(listwise)
category_review_stats = (
    business_stats_top10_no_explode
    .groupby("category_clean")["total_reviews"]
    .sum()
    .reset_index()
    .sort_values(by="total_reviews", ascending=False)
)
```

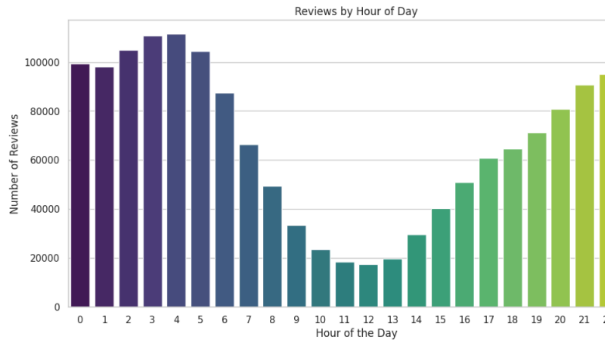
Output:

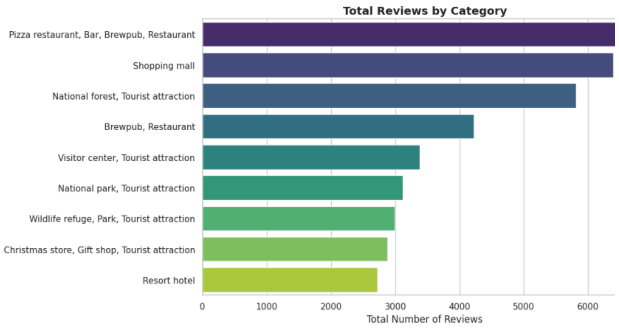
[Top 5 businesses by total number of reviews]

	business_name	category_clean	review_hour	total_reviews	avg_rating
19245	Moose's Tooth Pub & Pizzeria	Pizza restaurant, Bar, Brewpub, Restaurant	13	6591.0	4.7
22806	Tongass National Forest	National forest, Tourist attraction	23	5812.0	4.8
48985	49th State Brewing - Anchorage	Brewpub, Restaurant	1	4227.0	4.5
12626	Mendenhall Glacier Visitor Center	Visitor center, Tourist attraction	17	3382.0	4.8
23488	Dimond Center	Shopping mall	18	3297.0	4.1

[Top 5 businesses with exploded business categories]

	business_name	category_exploded	total_reviews	avg_rating	review_hours
3623	Moose's Tooth Pub & Pizzeria	Pizza restaurant	6591.0	4.7	[2, 8, 19, 5, 1, 19, 4, 21, 5, 5, 6, 3, 8, 19,...
3624	Moose's Tooth Pub & Pizzeria	Restaurant	6591.0	4.7	[2, 8, 19, 5, 1, 19, 4, 21, 5, 5, 6, 3, 8, 19,...
9328	Moose's Tooth Pub & Pizzeria	Brewpub	6591.0	4.7	[2, 8, 19, 5, 1, 19, 4, 21, 5, 5, 6, 3, 8, 19,...
3622	Moose's Tooth Pub & Pizzeria	Bar	6591.0	4.7	[2, 8, 19, 5, 1, 19, 4, 21, 5, 5, 6, 3, 8, 19,...
11150	Tongass National Forest	Tourist attraction	5812.0	4.8	[0, 5, 14, 22, 16, 2, 19, 13, 1, 15, 3, 2, 15,...
5348	Tongass National Forest	National forest	5812.0	4.8	[0, 5, 14, 22, 16, 2, 19, 13, 1, 15, 3, 2, 15,...
5914	49th State Brewing - Anchorage	Restaurant	4227.0	4.5	[18, 7, 11, 0, 1, 3, 7, 23, 13, 2, 17, 15, 23,...
5913	49th State Brewing - Anchorage	Brewpub	4227.0	4.5	[18, 7, 11, 0, 1, 3, 7, 23, 13, 2, 17, 15, 23,...
3505	Mendenhall Glacier Visitor Center	Tourist attraction	3382.0	4.8	[1, 22, 5, 3, 22, 6, 2, 19, 19, 19, 6, 23, 19,...
9228	Mendenhall Glacier Visitor Center	Visitor center	3382.0	4.8	[1, 22, 5, 3, 22, 6, 2, 19, 19, 19, 6, 23, 19,...





Review timestamps in milliseconds are converted into Spark timestamps, and the review_hour is extracted for temporal analysis. Business categories are cleaned and exploded to handle multiple categories per business. Aggregation is performed by business_name and category to compute the total number of reviews and average ratings. Review hours are collected and flattened into a pandas Series for frequency counting. Pandas and Seaborn are used for the visualization of hourly review distribution. Alternative approaches: Spark's hour() function could directly generate hourly counts; weighted or Bayesian averages could improve rating robustness. The chosen method is efficient for large-scale preprocessing (via Spark) and flexible for visualization (via Pandas).

1.4 Text Review Analysis and Word Cloud Visualization

```
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
extra_stopwords = {
    'no', 'review', 'reviewed', 'reviews', 'http', 'https', 'amp',
    'place', 'would', 'really', 'also', 'one', 'two', 'got', 'went'
}
stop_words |= extra_stopwords
```

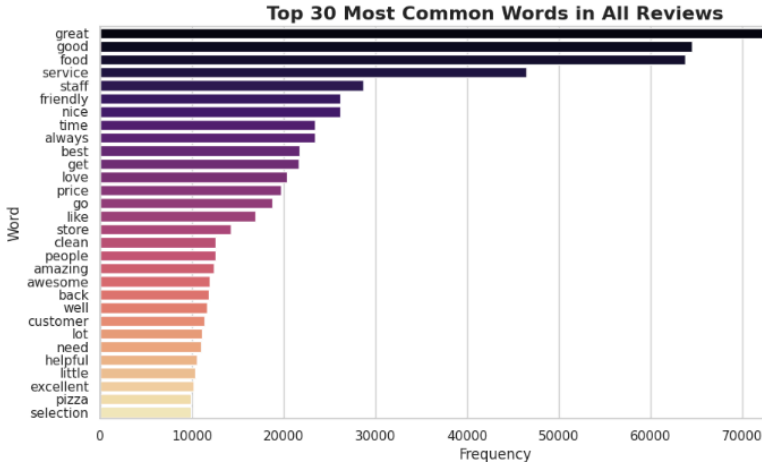
```
 # Create and display the word cloud
 wordcloud = WordCloud(
     width=1000,
     height=500,
     background_color="white",
     max_words=200
 ).generate_from_frequencies(word_freq)
```

```
# Combine all reviews into one large text
all_text = " ".join(joined_df_clean.select("text").toPandas()["text"])

all_tokens = clean_and_tokenize(all_text)
word_freq = Counter(all_tokens)

top_30_words = word_freq.most_common(30)

top_words_df = pd.DataFrame(top_30_words, columns=["word", "count"])
```

Sample Output for year 2007:

```
🔹 Top 30 Most Common Words in 2007
closest          1
thing            1
homer            1
music            1
venue            1
hosting          1
beausoleil       1
town             1
respectable      1
bar              1
food             1
```


```
Top 30 Most Common Words Across All Reviews
         word  count
0       great  81973
1        good  64479
2        food  63780
3     service  46462
4       staff  28697
5    friendly  26247
6        nice  26186
7        time  23445
8      always  23385
9        best  21717
10        get  21669
11       love  20406
12      price  19707
13         go  18775
14       like  16969
15      store  14225
16      clean  12610
17     people  12607
18    amazing  12382
19    awesome  11949
20       back  11850
21       well  11651
22   customer  11388
23        lot  11163
24       need  11052
25    helpful  10574
26     little  10363
27  excellent  10196
28      pizza   9925
29  selection   9916
```



Review text is tokenized into words using nltk.word_tokenize. Stop words are removed using NLTK's stopwords list. Remaining words are normalized (lowercased, punctuation removed) for accurate frequency counting. The top 30 words are extracted using FreqDist/Counter and displayed as a frequency table. Review timestamps are converted to year values (.dt.year) to group reviews by year. Word clouds are generated separately per year, with word size proportional to frequency. Alternative approaches: TF-IDF weighting or lemmatization could improve semantic grouping. The chosen method is optimal for exploratory analysis because frequency + word clouds provide intuitive insights into review themes.

Insights:

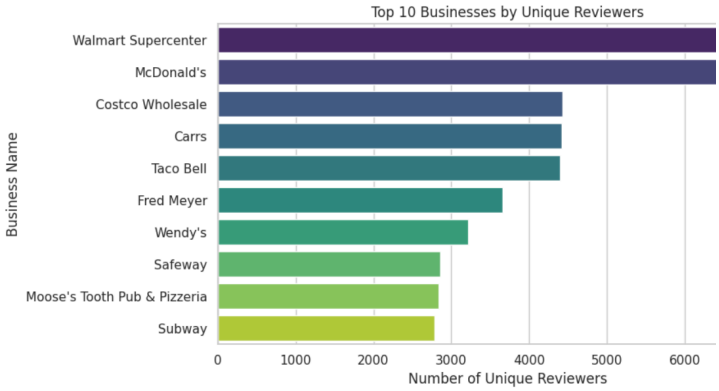
Reviews across years are mostly positive, emphasizing good food, great service, friendly staff, and cleanliness as key factors driving high satisfaction and repeat customers.

1.5 Unique Reviewer Analysis and Temporal Trends by Business and Category

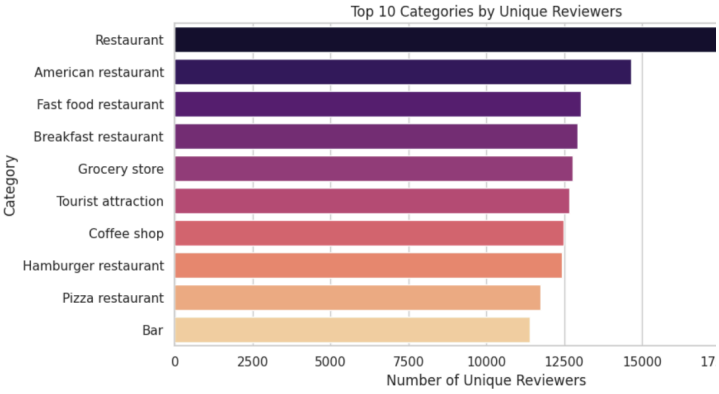
```
unique_reviewers_business = (
    joined_df_clean.groupBy("business_name")
      .agg(F.countDistinct("user_id").alias("unique_reviewers"))
      .orderBy(F.desc("unique_reviewers")))
```

Output:

```
+--------------------------+---------------
|business_name             |unique_reviewer
+--------------------------+---------------
|Walmart Supercenter       |7092
|McDonald's                |6951
|Costco Wholesale          |4430
|Carrs                     |4424
|Taco Bell                 |4402
|Fred Meyer                |3664
|Wendy's                   |3218
|Safeway                   |2855
|Moose's Tooth Pub & Pizzeria|2833
|Subway                    |2782
+--------------------------+---------------
only showing top 10 rows
```



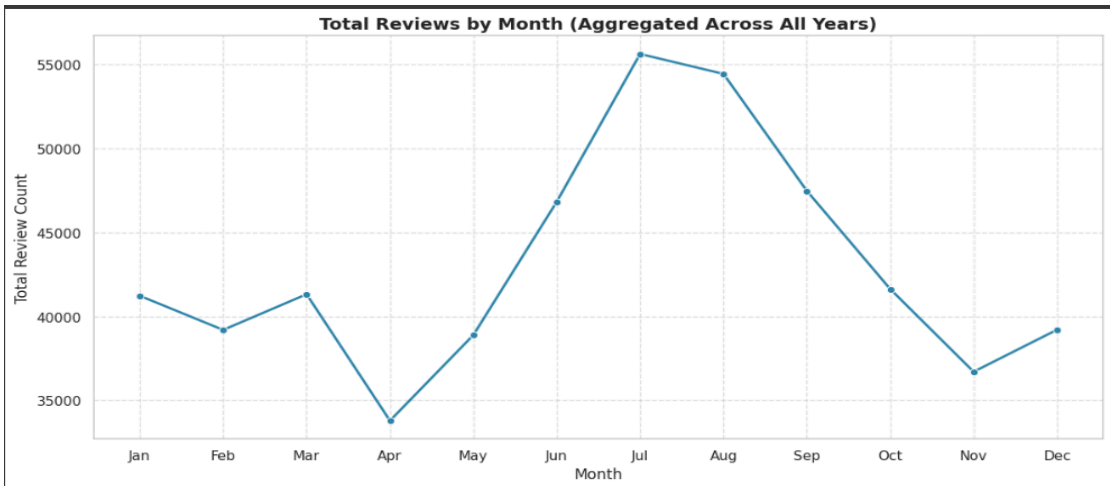

```
+--------------------+----------------+
|category_item       |unique_reviewers|
+--------------------+----------------+
|Restaurant          |19042           |
|American restaurant |14659           |
|Fast food restaurant|13031           |
|Breakfast restaurant|12917           |
|Grocery store       |12775           |
|Tourist attraction  |12658           |
|Coffee shop         |12471           |
|Hamburger restaurant|12426           |
|Pizza restaurant    |11750           |
|Bar                 |11407           |
+--------------------+----------------+
only showing top 10 rows
```



Counted unique reviewers for each business using PySpark's groupBy with countDistinct("user_id") to identify popularity. For categories, split, filtered, and exploded category lists, then repeated the grouping to aggregate unique reviewers per category item. Results could also be obtained in pandas for smaller datasets, but PySpark is more efficient for large datasets. Categories could be grouped by business type or top-level descriptor, but detailed category-level analysis provides granular insights. This method is optimal since Spark handles large datasets efficiently, and the approach provides clear, actionable insights on what attracts the most distinct reviewers.

Insights:

The businesses attracting the most unique reviewers are large chains and retailers, with Walmart Supercenter and McDonald's ranking highest. This suggests broad customer bases and high visibility. Restaurant-type categories dominate the top spots for unique reviewers, indicating a strong preference for food and hospitality venues among the review population. Findings show that specific food categories, especially American, fast food, and breakfast restaurants, receive high engagement, reflecting diverse eating-out habits. Tourist attractions and grocery stores also have notable reach, showing their importance in daily life and travel for users.



1.6 Building a Business Recommendation System

1.6.1 Strategy for Business Recommendations

- **Objective:** Build a business recommendation system for customers, leveraging both user ratings and business categories to generate personalized suggestions.
- **Data Preparation:** Aggregated ratings per user-business pair to handle multiple reviews from the same user. Converted business categories into an array format for content-based similarity. Indexed users and businesses to numeric IDs for matrix construction.
- **Collaborative Filtering Component:** Constructed an item-user sparse matrix from training data. Used item-based K-Nearest Neighbors (KNN) with cosine similarity to find businesses similar in terms of user ratings.
- **Content-Based Component:** Built a business-to-category mapping. Computed Jaccard similarity between business categories to capture content-based similarity.
- **Hybrid Strategy:** Combining collaborative similarity and category similarity with a weighted parameter alpha (hybrid_sim = alpha * collab_sim + (1-alpha) * category_jaccard). Recommended businesses to users by aggregating the neighbors of the items they liked, weighted by their ratings. Cold-start handling: fallback to the most popular businesses for users with no prior history.
- **Evaluation:** Used per-user holdout (latest review as test) to compute Precision@K (e.g., Precision@5 and Precision@10).

- **Visualization:** Generated bar plots showing recommended businesses and their recommendation scores per user, enabling intuitive presentation of top suggestions.
- **Interpretation:** High hybrid scores indicate strong similarity both in collaborative patterns and business content. System balances popularity and content relevance, giving personalized recommendations that reflect both past user behavior and business characteristics.

1.6.2 Implementation and Interpretation of Recommendations

KNN implementation:

```
# Fit item-based KNN (using cosine distance -> similarity = 1 - distance)
knn_k = 50  # number of neighbors to compute per item
knn = NearestNeighbors(n_neighbors=knn_k, metric='cosine', algorithm='brute')  # cosine distance
knn.fit(item_user_mat)  # rows are businesses

# Build business -> categories map (for hybrid content similarity)
biz_cats_pd = df.select("business_name", "category_array").distinct().toPandas()

# Normalize category entries and convert to set for Jaccard similarity
biz_cats_map = {}
for _, r in biz_cats_pd.iterrows():
    b = r['business_name']
    cat_arr = r['category_array'] or []
    if isinstance(cat_arr, str):
        cat_list = [c.strip() for c in cat_arr.split(',') if c.strip()]
    else:
        cat_list = [str(c).strip() for c in cat_arr if c is not None and str(c).strip() != '']
    biz_cats_map[b] = set(cat_list)

# Mapping between business_name <-> biz_idx
biz_map_pd = ratings.select("business_name", "biz_idx").distinct().toPandas()
biz_to_idx = dict(zip(biz_map_pd['business_name'], biz_map_pd['biz_idx'].astype(int)))
idx_to_biz = {v: k for k, v in biz_to_idx.items()}
```

Cosine and Jaccard similarity:

```python
# Helper function: Jaccard similarity for categories
def jaccard_for_pair(biz_name_a, biz_name_b):
    sa = biz_cats_map.get(biz_name_a, set())
    sb = biz_cats_map.get(biz_name_b, set())
    if not sa and not sb:
        return 0.0
    inter = sa.intersection(sb)
    union = sa.union(sb)
    return float(len(inter)) / float(len(union)) if union else 0.0

# Recommend similar businesses
def get_similar_businesses_by_idx(biz_idx, top_k=10, alpha=0.75):
    """
    biz_idx: index of business in item_user_mat
    alpha: weight for collaborative similarity (0..1)
    Returns top_k businesses with hybrid similarity
    """
    distances, indices = knn.kneighbors(item_user_mat[biz_idx], n_neighbors=knn_k)
    distances = distances.flatten()
    indices = indices.flatten()
    results = []
    base_name = idx_to_biz.get(biz_idx, None)
    for dist, idx in zip(distances, indices):
        if idx == biz_idx:  # skip self
            continue
        collab_sim = 1.0 - float(dist)  # cosine distance -> similarity
        name_b = idx_to_biz.get(idx, None)
        jacc = jaccard_for_pair(base_name, name_b) if name_b else 0.0
        hybrid = alpha * collab_sim + (1 - alpha) * jacc
        results.append((int(idx), float(hybrid), float(collab_sim), float(jacc), name_b))
    return sorted(results, key=lambda x: x[1], reverse=True)[:top_k]
```

Recommendation based on a hybrid approach:

```
# Recommend businesses for a user
def recommend_for_user(user_id, top_n=10, rating_threshold=3.5, alpha=0.75, knn_k_test=10):
    """
    Generate top-N business recommendations for a given user using hybrid similarity.
    Uses reviewer_name for readability.
    knn_k_test reduces neighbors for speed.
    """
    user_to_idx = dict(zip(user_map_pd['reviewer_name'], user_map_pd['user_idx'].astype(int)))

    if user_id not in user_to_idx:
        # Cold start: fallback to most popular items
        pop_pd = (
            ratings.groupBy("biz_idx")
                   .agg(F.countDistinct("user_idx").alias("n_users"), F.avg("rating").alias("avg_rating"))
                   .orderBy(F.desc("n_users"), F.desc("avg_rating"))
                   .limit(top_n)
                   .toPandas()
        )
        return [(idx_to_biz[int(row['biz_idx'])], None) for _, row in pop_pd.iterrows()]

    uidx = user_to_idx[user_id]
    user_items = train_pd[train_pd['user_idx'] == uidx]

    candidate_scores = {}
    for _, row in user_items.iterrows():
        b_idx = int(row['biz_idx'])
        user_rating = float(row['rating'])
        if user_rating < rating_threshold:
            continue
        neighbors = get_similar_businesses_by_idx(b_idx, top_k=knn_k_test, alpha=alpha)
        for nb_idx, hybrid_sim, collab_sim, jacc, nb_name in neighbors:
            score = hybrid_sim * user_rating
            candidate_scores[nb_idx] = candidate_scores.get(nb_idx, 0) + score

    seen_biz_idx = set(user_items['biz_idx'].astype(int).tolist())
    ranked = sorted(candidate_scores.items(), key=lambda x: x[1], reverse=True)
    recs = [(idx_to_biz.get(idx, None), float(score)) for idx, score in ranked if idx not in seen_biz_idx][:top_n]
    return recs
```

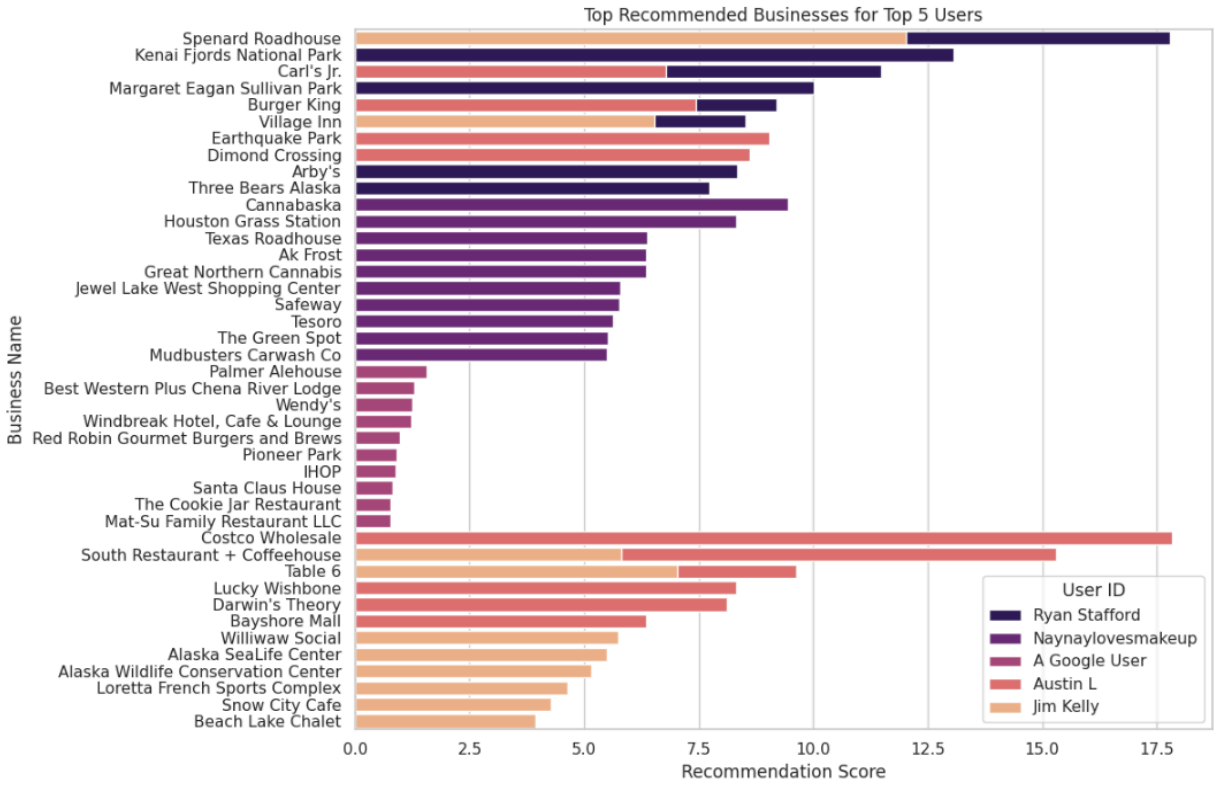
Output:

Recommendations for reviewer: Gabriela Betancourt

[('Great Alaskan Lumberjack Show', 6.570001657492506), ('Port Of Skagway', 6.1902491988846915), ('Icy Strait Point', 2.417725885207272), ('Alaska State Museum', 2.2677307914424722), ('Skagway Museum', 2.22852957769677), ('Skagway Cruise Terminal', 1.978132037871201), ('Shrine of Saint Therese', 1.9755930018674095), ('Chugach National Forest', 1.9678057541843683), ('Denali National Park and Preserve', 1.7952602107994515), ('Alaska Railroad TOFC Building', 1.7486137861504711)]

Precision@5 (sampled) = 0.04

Precision@10 (sampled) = 0.06



:

Recommendations for Gabriela Betancourt: The model has suggested 10 businesses that she has not rated yet, ranked by a hybrid score (collaborative + category similarity). The top recommendations, like “Great Alaskan Lumberjack Show” and “Port Of Skagway” have higher scores, meaning the system predicts these are more likely to match her past preferences. Lower-scoring items (like “Alaska Railroad TOFC Building”) are less strongly recommended but still appear in the top 10 based on similarity aggregation.

Precision@K results: Precision@5 = 0.04 and Precision@10 = 0.06 are quite low.

This means that for the sampled test users, only 4% of their held-out reviews appear in the top 5 recommendations, and 6% appear in the top 10.

What we can infer: The model is capturing some patterns, but the predictive accuracy is low. This is common with sparse datasets, small user histories, or highly diverse business choices. Users might have unique tastes that are hard to predict using only past reviews and categories. The hybrid approach combines collaborative filtering and category similarity, which helps when pure collaborative signals are weak.

Next steps for improvement could include: Incorporating more features, like review text sentiment, location proximity, or business popularity. Using a larger training dataset or including all users instead of sampling. Tuning alpha (weight of collaborative vs. category similarity) for better results.
Trying user-based CF or matrix factorization methods.

1.7 Analysis of Ratings and Review Content

1.7.1 Visualizing Ratings Across Business Categories

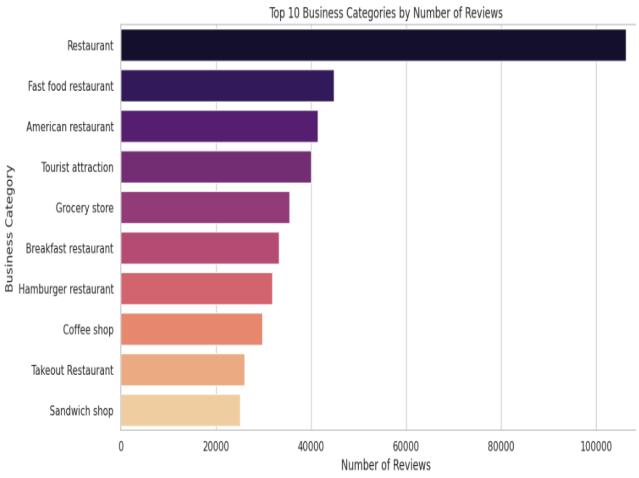
Filtering top business categories as per the total number of reviews

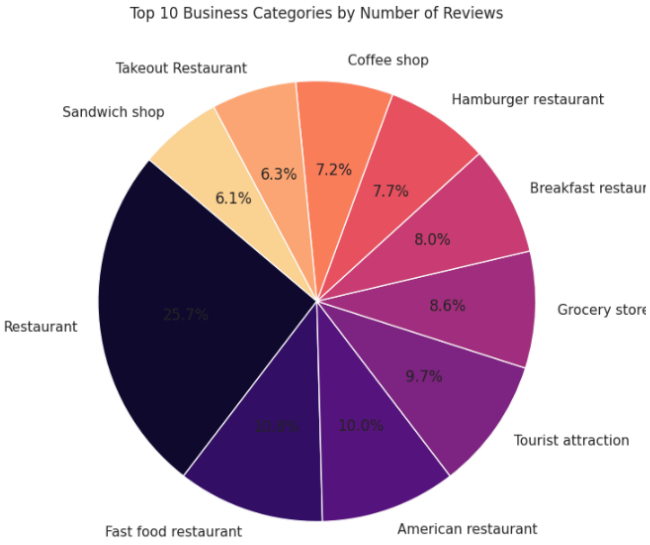
```
# Filter categories with enough reviews (optional, e.g., > 20 reviews)
category_rating_filtered = category_rating[category_rating['num_reviews'] > 10]
category_top = category_rating_filtered.sort_values(by='num_reviews', ascending=False).head(10)
```

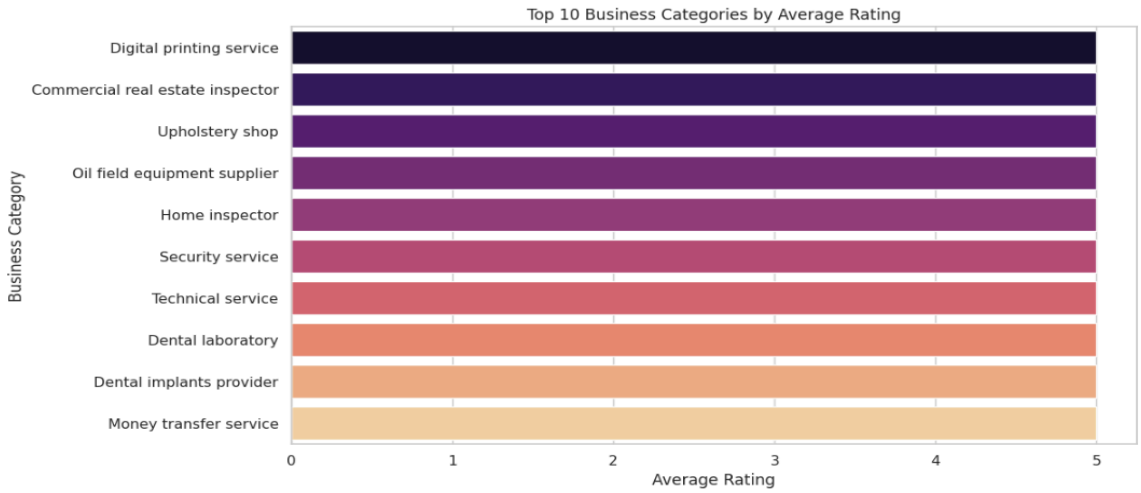
Top Business categories as per the average rating

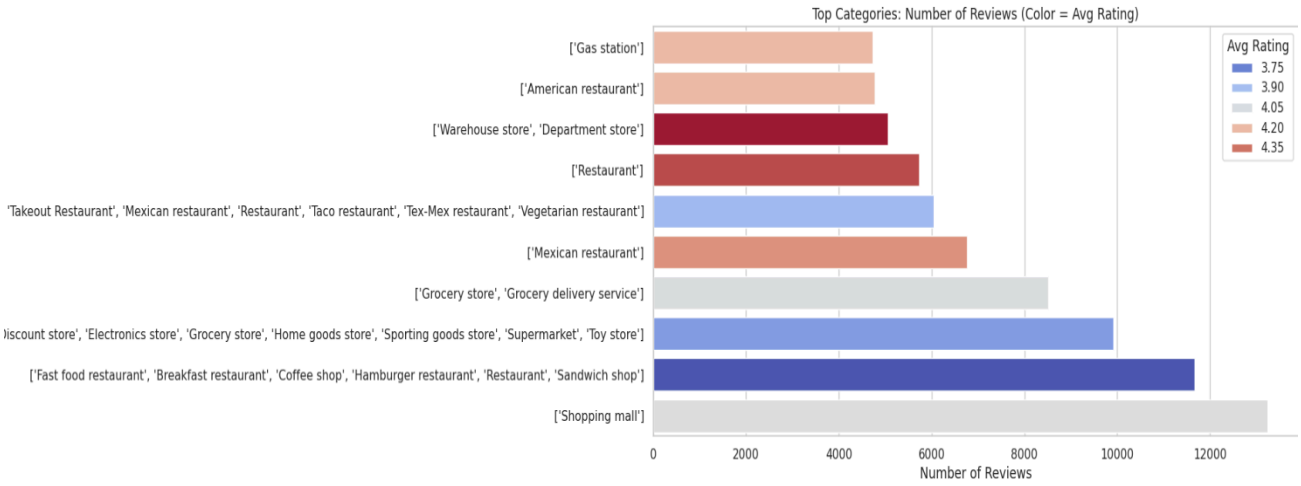
```
# Sort top categories by average rating
category_top_by_rating = category_rating_filtered.sort_values(by='avg_rating', ascending=False).head(10)
```

Output:









Insights:

"Restaurant" garners the most reviews, followed by other food-related categories like "Fast Food Restaurant" and "American Restaurant." Reflects the high volume of customer feedback in the food industry. Shows that categories like "Restaurant" and "Warehouse Store / Department Store" have high review counts and good average ratings. Subcategories like "Fast Food Restaurant" tend to have lower average ratings despite many reviews.

1.7.2 Text Analysis of Low-Rated Reviews

```
# WordCloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(" ".join(words_filtered))
plt.figure(figsize=(15,6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Common Words in Low Rating Reviews")
plt.show()
```

Output:

Top 20 words in low rating reviews: [('review', 12156), ('food', 6245), ('service', 5081), ('get', 4181), ('time', 3618), ('like', 3435), ('place', 3418), ('good', 3184), ('one', 3162), ('back', 3021), ('would', 2945), ('didn't', 2920), ('never', 2553), ('even', 2531), ('order', 2490), ('didn't', 2363), ('got', 2353), ('customer', 2031), ('people', 1984), ('told', 1905)]

Insights:

Highlights frequent negative phrases such as “bad,” “rude,” “never,” and “worst.” Indicates customer dissatisfaction related to poor service, unfulfilled orders, and rude staff. Rating trends indicate certain food-related segments often receive lower ratings despite high engagement.

1.8 Reviewer Analysis: Business History and Similarities

1.8.1 Creating Sorted Business Lists for Reviewers

```
# UDF to sort by timestamp and extract business names
def sort_extract_business(reviews):
    reviews_sorted = sorted(reviews, key=lambda x: x['review_time_ts'])
    return [x['business_name'] for x in reviews_sorted]

sort_extract_business_udf = udf(sort_extract_business, ArrayType(StringType()))

user_business_df = user_business_df.withColumn(
    "user_business_list",
    sort_extract_business_udf("business_reviews")
).drop("business_reviews")
```

Output:

```
+--------------------+-------------------+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
|user_id             |reviewer_name      |user_business_list                                                                                                                                                                                                  |
+--------------------+-------------------+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
|1.00003825755859E20 |Erica Hill         |[Yes Bistro, Old Town Diner, Bread and Brew, Duluth Trading Company, Pita Place, The Banks Alehouse, Baranof Downtown, Bw Signature Collection, Kava's Pancake House, Middle Way Cafe, Simon & Seafort's Saloon & G
|1.0000428139011082E20|M Ric              |[West Valley Plaza, Fred Meyer, Big Dipper Ice Arena, Safeway, Blue Loon, Gambardella's Pasta Bella, Kaladi Brothers Coffee, Aviator Hotel Anchorage, Kenai Fjords National Park Visitor Center, The Home Depot]
|1.0000620838495144E20|C•lt•n G. (KodakColt)|[Palmer Hay Flats State Game Refuge, AutoZone Auto Parts, Batteries Plus Bulbs, Hatcher Pass, Alcantra Disc Golf Course, Inkspot, Wasilla Carrier Annex, Wasilla Family Auto LLC, Master Auto Repair, Mudbusters Ca
|1.0000670037562006E20|James Oakley       |[Peters Creek, Sheep Creek Lodge, Denali Viewpoint South, Mexico In Alaska Restaurant, Humpy's Great Alaskan Alehouse, Three Bears Healy, McKinley Creekside Cafe, Montana Creek Campground, Denali Princess Wilder
|1.0000684986283477E20|P Ursery           |[Thumbs Up by Gnap&PK, McDonald's, Bible Baptist Church, Vip Dry Cleaners & Laundromat, Big Daddy's BBQ & Banquet Hall, Lavelle's Bistro, Thai Orchid Drive Thru, Walmart Supercenter, Irashai Japanese Restaurant,
|1.0000719467235166E20|Victoria Inthaly   |[Sno Flo Alaska, Anchorage 5th Avenue Mall, Fred Meyer, Dimond Center, Phở Tommy, Cannabaska, Wild Scoops, KFC, Alaska Zoo, Fred Meyer, Burger King, Barnes & Noble, Pizza Studio Anchorage, Dave & Buster's, IHOP]
|1.0000787767883584E20|Amanda Franklin    |[Three Bears Alaska, Quality Suites Historic Downtown, Humpy's Great Alaskan Alehouse, Wendy's, Lucky Wishbone, Turtle Club, Arctic Travelers Gift Shop, Clarion Hotel & Suites Fairbanks Near Ft. Wainwright, Wend
|1.00008316982005E20 |Trent Rogers       |[The Banks Alehouse, Safeway, Cold Spot Feeds, Wendy's, Lowe's Home Improvement, Big Ray's, Golden Heart Veterinary Services, The Home Depot, Sears, Subway, Brewsters Restaurant, Museum of the North, Suki's Alas
|1.0001122573921386E20|Stefenie Mccallie  |[JOANN Fabrics and Crafts, Angel Rocks Trailhead, The Cookie Jar Restaurant, L.A. Nails, Bei Jing Hot Pot & Asian Cuisine, McDonald's, Seekins Ford Lincoln, Birch Hill Recreation Area, Eagle River Motel, Kendall
|1.000133514126819E20 |W David Schaad     |[Tracy’s King Crab Shack, Baranof Downtown, Bw Signature Collection, Orca Point Lodge, Hotel Captain Cook, Tongass National Forest, Allen Marine Tours, Alaska Railroad Corporation, Alaska SeaLife Center, Ray's W
|1.0001436391165977E20|Mike Nelson        |[Begich Boggs Visitor Center, Creekbend Cafe, Sea Bean Cafe, Seward Boat Harbor, Kenai Fjords Tours, AJ's OldTown Steakhouse & Tavern, Humpy's Great Alaska Alehouse, 49th State Brewing - Anchorage, Fresh Sourdou
|1.0001542142415123E20|Chris R            |[Hotel Captain Cook, Hilton Anchorage, Hard Rock Cafe, JBER Main Exchange, Red Robin Gourmet Burgers and Brews, Thunderbird Falls, Elmendorf Fitness Center, Golden Donuts, Dino's Donuts, Taco Bell, Pita Pit, Ala
|1.0001608002978447E20|Mariah Pestriakov  |[Best Western Kodiak Inn And Convention Center, Bernie's Bungalow Lounge, La Cabaña Mexican Restaurant, White Spot Cafe, Regal Totem, Kaladi Brothers Coffee, Uncle Joe's Pizzeria, Costco Wholesale, Walmart Super
|1.000171576509634E20 |Skye Stillwater    |[Newcomb Park Wasilla Lake, Costco Wholesale, H2Oasis Indoor Waterpark, Mat-Su Alano Club, Value Village, Chepo's Mexican Restaurant - Wasilla, Body Piercing Unlimited & Tattoo Palmer, Mat-Su Tattoo & Body Pierc
|1.0002309302239738E20|Katherine Goode    |[Sami's City Diner, Spenard Roadhouse, Wendy's, Dino's Donuts, Pho Lena, Taco King, Kansha Japanese Restaurant, Smashburger, Moose's Tooth Pub & Pizzeria, Dollar Zone, Tommy's Burger Stop, Get Air Trampoline Par
|1.0002378262815456E20|Susea Albee        |[Whale Park, Donna's Restaurant, Sandpiper Cafe, Sitka Performing Arts Center, Halibut Point Recreation Site, Sitka National Historical Park, Embassy Suites by Hilton Anchorage, Channel Club, Mean Queen @ Totem
|1.0002613478726395E20|Samuel S           |[Regal Goldstream & IMAX, Fred Meyer, Fred Meyer, Golden Corral Buffet & Grill, Denali National Park and Preserve, Fred Meyer, Pioneer Park, Safeway, Village Inn, North Pole Plaza, Roundup Steakhouse & Saloon, L
|1.0002633380652619E20|John Hill          |[United States Postal Service, Walmart Supercenter, Fred Meyer, McDonald's, Happy Family Sushi & Wok Restaurant, McDonald's, JBER Main Exchange, Snow City Cafe, Target, QDOBA Mexican Eats, Taco Bell, Smashburger
|1.0002645133665146E20|like and sub please|[Señor Panchos, McDonald's, Anchorage Alaska Temple, Everything Bagels, Sweeney's Clothing, Taco Bell, Solid Rock Bible Camp, Papa Murphy's | Take 'N' Bake Pizza, Chair 5 Restaurant, Grand View Inn & Suites, Pan
|1.0002648968731022E20|Jesus saves        |[AK Livin The Dream Ranch, LLC., Foxy Nails & Tanning, Costco Wholesale, Lowe's Home Improvement, Subway, Dairy Queen Grill & Chill, Wendy's, Chevrolet of Wasilla, McDonald's, Chugach State Park, 2GO Tesoro, All
+--------------------+-------------------+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
only showing top 20 rows
```

1.8.2 Removing Duplicates from Reviewer Business Lists

Removing duplicated business

```python
# Remove duplicates
def remove_duplicates(lst):
    seen = set()
    res = []
    for x in lst:
        if x not in seen:
            res.append(x)
            seen.add(x)
    return res
```

Top 10 users with the highest number of unique business reviews

```python
# Sort users by number of unique businesses in descending order
top_users_df = user_business_df.orderBy(F.col("num_business_after").desc())
```

Output:

```
+--------------------+--------------------+------------------+------------------+
|user_id             |reviewer_name       |num_business_before|num_business_after|
+--------------------+--------------------+------------------+------------------+
|1.00003825755859E20 |Erica Hill          |16                |16                |
|1.0000428139011082E20|M Ric               |10                |10                |
|1.0000620838495144E20|C•lt•n G. (KodakColt)|10                |10                |
|1.0000670037562006E20|James Oakley        |17                |17                |
|1.0000684986283477E20|P Ursery            |10                |10                |
|1.0000719467235166E20|Victoria Inthaly    |15                |14                |
|1.0000787767883584E20|Amanda Franklin     |21                |20                |
|1.00008316982005E20 |Trent Rogers        |22                |22                |
|1.0001122573921386E20|Stefenie Mccallie   |10                |10                |
|1.000133514126819E20 |W David Schaad      |10                |10                |
|1.0001436391165977E20|Mike Nelson         |12                |12                |
|1.0001542142415123E20|Chris R             |13                |13                |
|1.0001608002978447E20|Mariah Pestriakov   |76                |70                |
|1.000171576509634E20 |Skye Stillwater     |11                |11                |
|1.0002309302239738E20|Katherine Goode     |32                |31                |
|1.0002378262815456E20|Susea Albee         |13                |13                |
|1.0002613478726395E20|Samuel S            |33                |30                |
|1.0002633380652619E20|John Hill           |26                |25                |
|1.0002645133665146E20|like and sub please |13                |13                |
|1.0002648968731022E20|Jesus saves         |33                |31                |
+--------------------+--------------------+------------------+------------------+
only showing top 20 rows
```

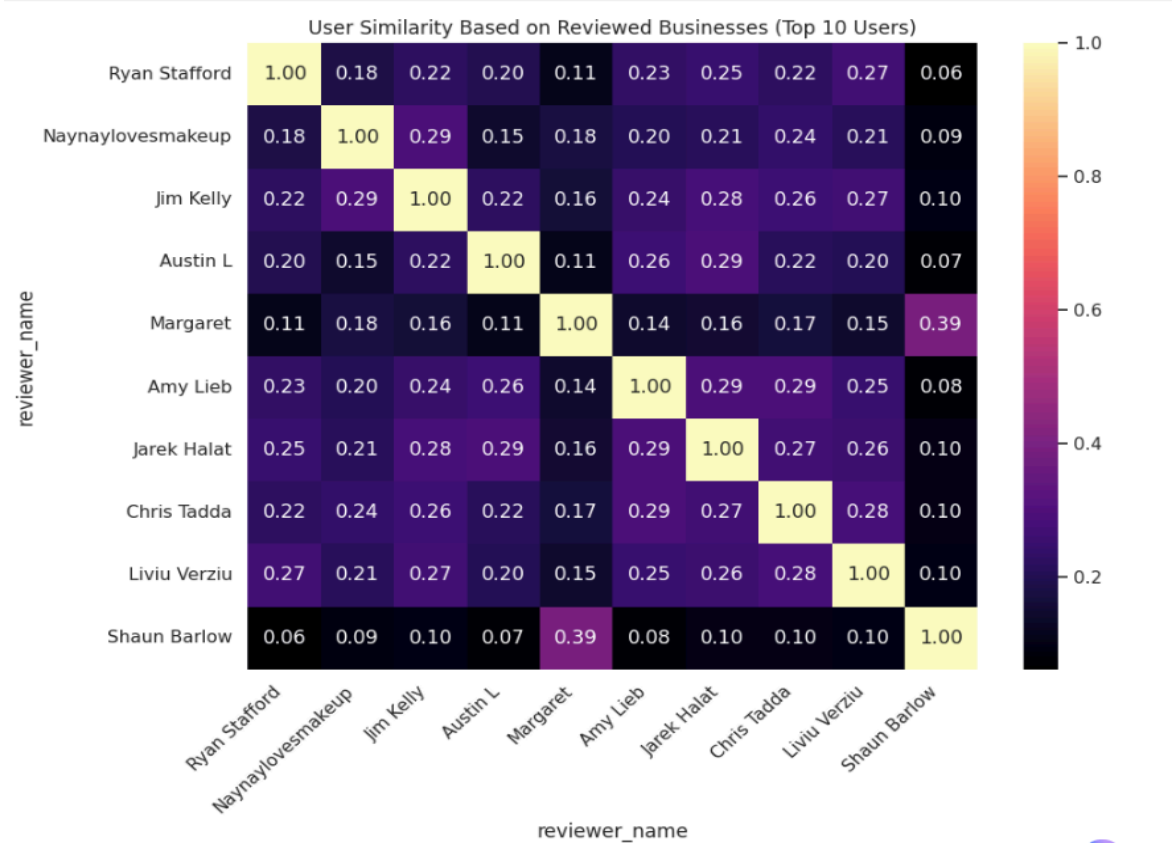
Insights:

Each user's user_business_list shows the order of businesses they reviewed over time. For most users, the number of businesses before and after removing duplicates is the same — meaning they reviewed each business only once. Some users (for example, Mariah Pestriakov) had a smaller count after removing duplicates (76 ⟶ 70), which means they reviewed the same business multiple times.

1.8.3 Analyzing User Similarities Based on Reviewed Businesses

```
# 1. Get top 10 users by unique reviewed businesses
top_users_df = user_business_df.orderBy(F.col("num_business_after").desc()).limit(10)

# 2. Convert to Pandas for encoding, including reviewer_name
top_users_pd = top_users_df.select("reviewer_name", "user_business_list_unique").toPandas()

# 3. Multi-hot encode businesses
mlb = MultiLabelBinarizer()
user_business_encoded = mlb.fit_transform(top_users_pd['user_business_list_unique'])

# 4. Compute cosine similarity between users
similarity_matrix = cosine_similarity(user_business_encoded)

# 5. Convert to DataFrame using reviewer_name
similarity_df = pd.DataFrame(similarity_matrix,
                             index=top_users_pd['reviewer_name'],
                             columns=top_users_pd['reviewer_name'])
```

Output:



Strategy:

Each user is represented by a binary vector over all businesses.

- 1 → user reviewed the business.
- 0 → user did not review the business.

MultiLabelBinarizer encodes each user_business_list_unique into these vectors. Cosine similarity is used to compare user vectors, since it normalizes for differences in the number of reviews. The result is a user–user similarity matrix showing how aligned their business review histories are. Similar users can be clustered for segmentation or used to recommend new businesses by analogy. Alternative measures include Jaccard similarity (shared vs. total businesses) or Pearson correlation. This approach is simple, interpretable, and scalable to more users and businesses.

Insights:

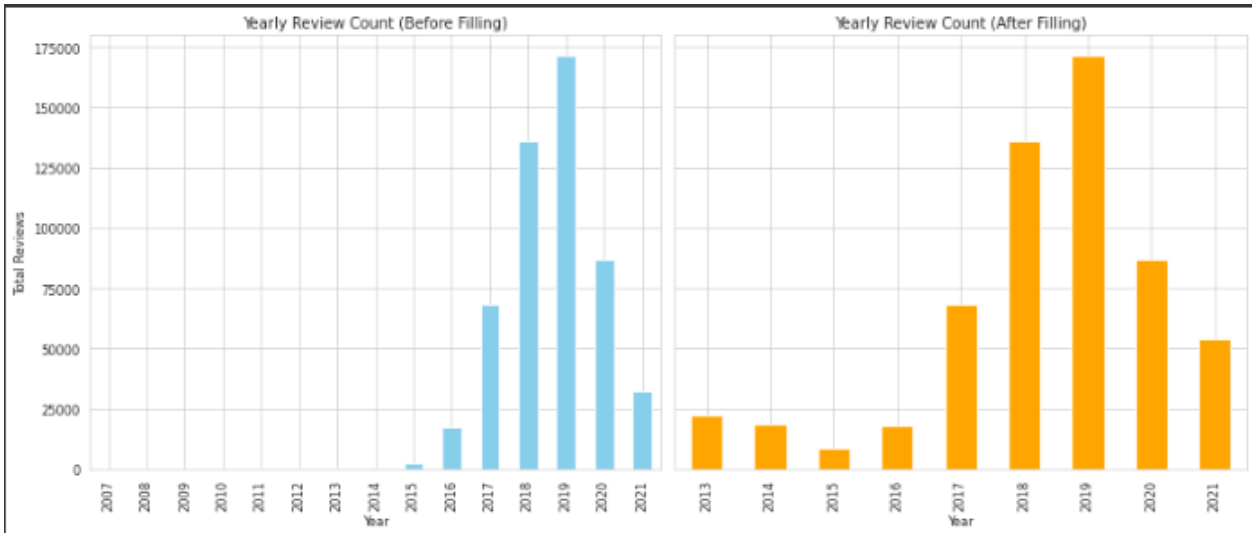
Users with higher similarity values (closer to 1.0) have reviewed many of the same businesses. For example, Shaun Barlow shows the highest similarity (0.39) with Margaret, suggesting they share many reviewed businesses. Most similarities fall between 0.10–0.30, meaning overlap exists but is limited — users have diverse reviewing habits. Users like Ryan Stafford or Austin L show low overlap with others, indicating unique reviewing patterns. This similarity analysis can support collaborative filtering: recommending businesses reviewed by one user to another with a similar history.

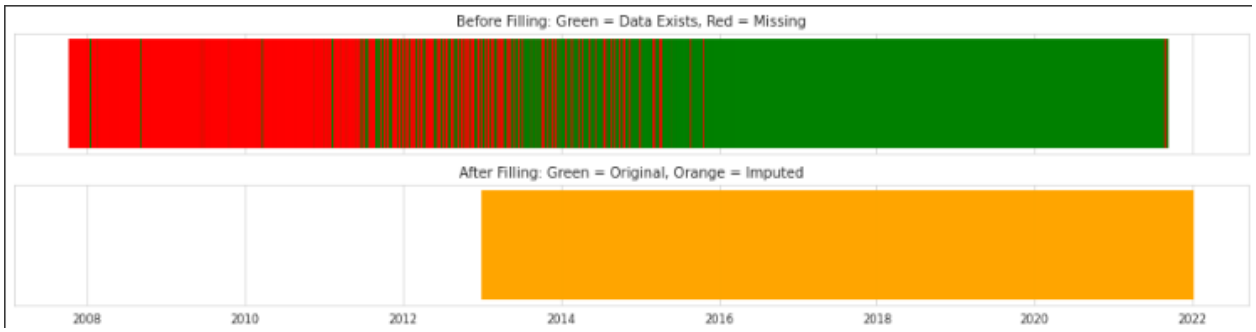
III. Part 2 Submission Prediction

2.1 Time Series Analysis and Seasonality of Review Volumes

When dealing with missing years in a time series, it's crucial **not to fill them with artificial or averaged values**, as doing so introduces false patterns that can mislead forecasting models. Instead, the safer and more reliable approaches are:
Model only the years with real data, ensuring the analysis reflects true historical behavior.
Aggregate data to higher levels (like monthly or quarterly) to handle sparsity without distortion.
Exclude completely missing years so that the model predicts based only on valid, observed data.

This approach maintains data integrity, prevents bias, and results in more trustworthy and interpretable forecasts.





The Data Continuity and Imputation Visualization section focuses on validating how missing dates or years are handled in the dataset. It begins by comparing yearly review counts before and after imputation to detect missing years or artificial jumps in data. Then, daily data presence is visualized — where green lines represent original data and orange lines show imputed (filled) dates — allowing you to confirm that missing gaps were filled accurately without over-imputation. Finally, before-and-after visual comparisons highlight data continuity improvements and reveal previously missing periods. Together, these analyses ensure that the dataset is complete, consistent, and reliable for time-series forecasting or further modeling.

STEP 1: Data preparation

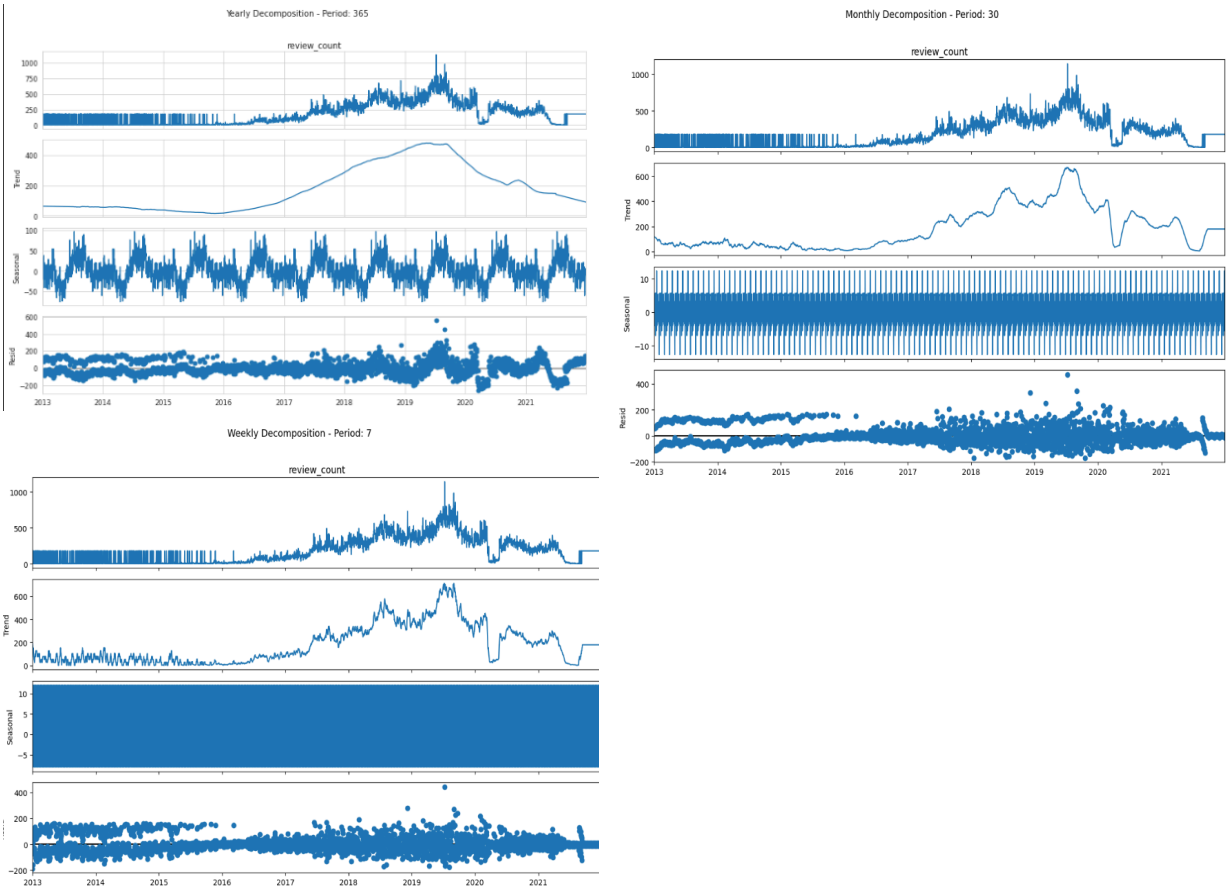
```
try:
    # Try to convert the date column to datetime
    df[date_col] = pd.to_datetime(df[date_col])

    # Try setting it as index
    df.set_index(date_col, inplace=True)
    print(f"Column '{date_col}' set as DatetimeIndex.")

except KeyError:
    # If the column doesn't exist, check if already DatetimeIndex
    if isinstance(df.index, pd.DatetimeIndex):
        print(f" Column '{date_col}' not found, but index is already a DatetimeIndex.")
    else:
        raise KeyError(f"Neither column '{date_col}' exists nor index is a DatetimeIndex.")

# Print basic time range details
print(f"\nDate range: {df.index.min()} to {df.index.max()}")
print(f"Total records: {len(df)}")

return df
```

Output:

Column 'date' set as DatetimeIndex. Date range: 2013-01-01 00:00:00 to 2021-12-31 00:00:00 Total records: 3287

STEP 2: Decomposition Methodology

```
try:
    decomposition = seasonal_decompose(
        ts,
        model=model,
        period=period,
        extrapolate_trend='freq'
    )

    # Plot the decomposition
    fig = decomposition.plot()
    fig.set_size_inches(12, 8)
    plt.suptitle(f'{title_prefix} - Period: {period}', y=1.02)
    plt.tight_layout()
    plt.show()

    # Optional: return decomposition object for further analysis
    return decomposition

except ValueError as e:
    print(f"Decomposition failed for period={period}: {e}")
    return None
```

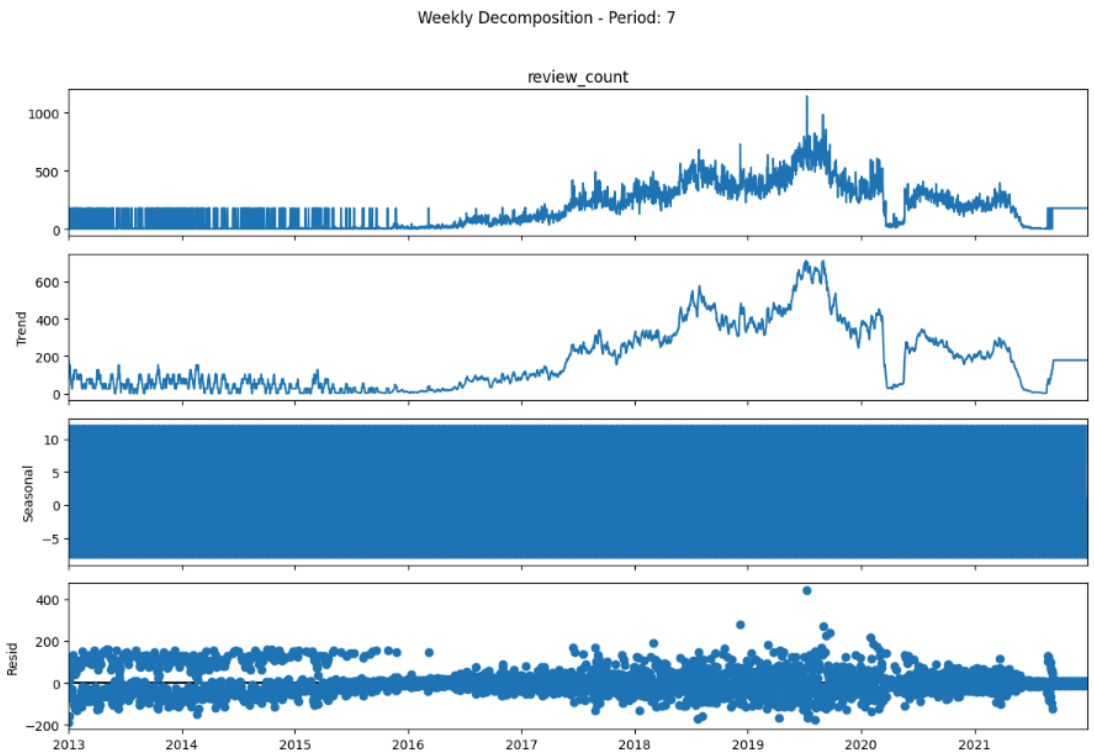


Used additive model: Y(t) = Trend(t) + Seasonal(t) + Residual(t). Seasonal period depends on the data frequency: 7 days for weekly patterns, 30 days for monthly patterns, 365 days for yearly patterns. extrapolate_trend='freq' ensures that the trend component is extended across the full index, even at the start and end, where seasonal decomposition may otherwise produce NaN values.

Step 3: Seasonality Analysis

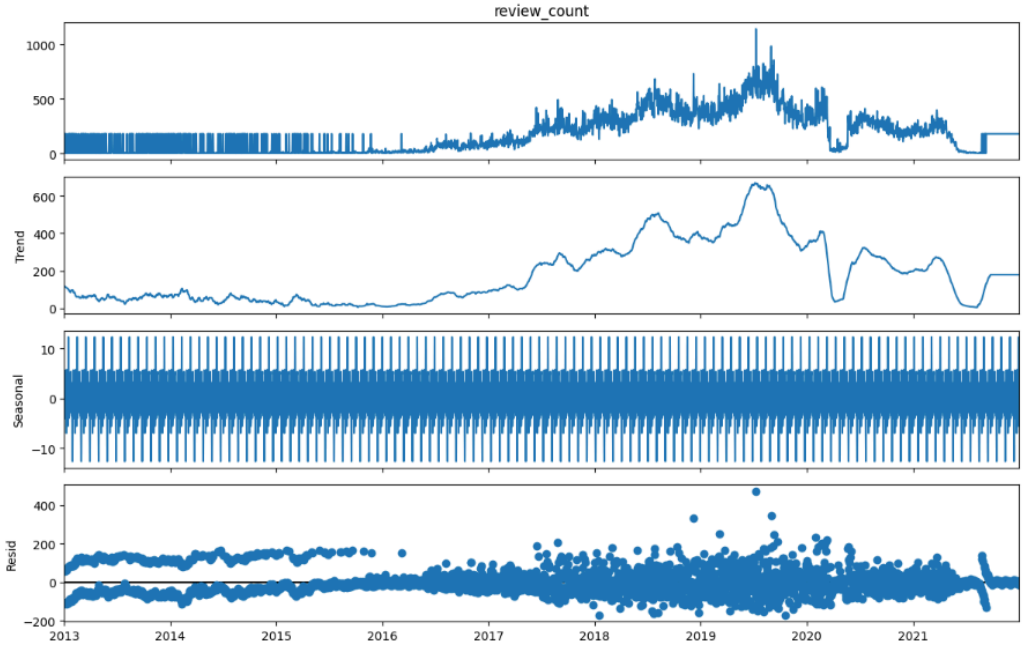
```
if seasonal_strength > 0.6:
    print("   → Strong seasonality pattern detected")
elif seasonal_strength > 0.3:
    print("   → Moderate seasonality pattern detected")
elif seasonal_strength > 0.1:
    print("   → Weak seasonality pattern detected")
else:
    print("   → No significant seasonality pattern detected")

return seasonal_strength
```

```
# Perform decomposition for weekly seasonality
decomp_weekly = decompose_and_plot(df_full['review_count'], period=7, title_prefix='Weekly Decomposition')
# Analyze seasonal strength
analyze_seasonality(decomp_weekly)

# Try monthly seasonality
decomp_monthly = decompose_and_plot(df_full['review_count'], period=30, title_prefix='Monthly Decomposition')
# Analyze seasonal strength
analyze_seasonality(decomp_monthly)

# Try yearly seasonality
decomp_yearly = decompose_and_plot(df_full['review_count'], period=365, title_prefix='Yearly Decomposition')
# Analyze seasonal strength
analyze_seasonality(decomp_yearly)
```



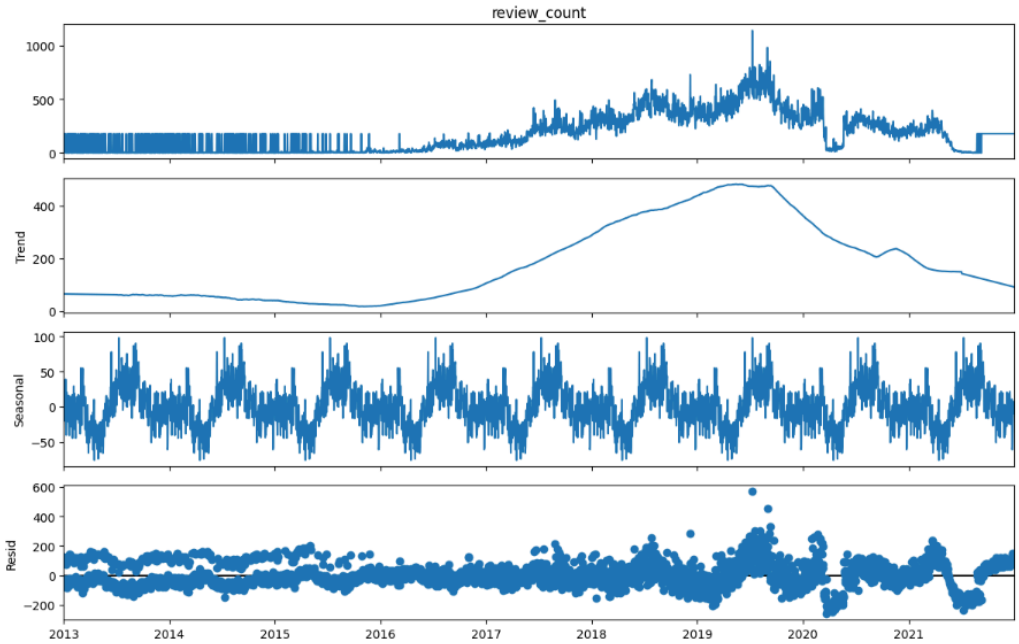

```
============================================================
SEASONALITY ANALYSIS RESULTS
============================================================

1. Seasonal Strength: 0.0130
   → No significant seasonality pattern detected
```

Monthly Decomposition - Period: 30




```
============================================================
SEASONALITY ANALYSIS RESULTS
============================================================

1. Seasonal Strength: 0.0067
   → No significant seasonality pattern detected
```

Yearly Decomposition - Period: 365




```
============================================================
SEASONALITY ANALYSIS RESULTS
============================================================

1. Seasonal Strength: 0.1261
   → Weak seasonality pattern detected
```

To understand repeating patterns in the data, we decomposed the time series into trend, seasonal, and residual components. Seasonal strength was quantified by comparing the variance of the seasonal component against residual noise. A high seasonal strength indicates a clear, repeating pattern at that period

Insights:

There is a clear upward trend in review counts until around 2019, followed by a decline after 2020, visible in all decompositions. Yearly and monthly decompositions reveal strong seasonality, indicating recurring patterns within each year and month, while the weekly decomposition shows negligible weekly seasonality. Residuals vary more in later years, especially around 2019-2020, reflecting increased unpredictability or external shocks during that period.

Step 5: Residual Analysis

```
# Extract residual component
residual = decomposition.resid

# Calculate statistics
residual_mean = residual.mean()
residual_std = residual.std()
```

Output:

Weekly:

```
============================================================
RESIDUAL COMPONENT ANALYSIS
============================================================

- Mean of residuals: -0.0587
- Standard deviation of residuals: 55.2234

2. Residual Analysis:
   - Mean of residuals: -0.0587
   - Standard deviation of residuals: 55.2234
   → Residuals appear random (good fit)
```

Monthly:

```
============================================================
RESIDUAL COMPONENT ANALYSIS
============================================================

- Mean of residuals: -0.0724
- Standard deviation of residuals: 60.6131

2. Residual Analysis:
   - Mean of residuals: -0.0724
   - Standard deviation of residuals: 60.6131
   → Residuals appear random (good fit)
```

Yearly:

```
============================================================
RESIDUAL COMPONENT ANALYSIS
============================================================

- Mean of residuals: 1.1187
- Standard deviation of residuals: 81.1023

2. Residual Analysis:
   - Mean of residuals: 1.1187
   - Standard deviation of residuals: 81.1023
   → Residuals appear random (good fit)
```

The function analyze_residuals extracts the residual component, which is the leftover noise after removing trend and seasonality.

Insights:

For weekly, monthly, and yearly decompositions, the residual means are very close to zero compared to their standard deviations, indicating residuals behave like random noise. This randomness suggests the decomposition models effectively captured the primary patterns (trend and seasonality), leaving only noise.

Step 4. Trend Analysis

```python
if trend_slope > 0:
    print("   → Upward trend in review counts")
elif trend_slope < 0:
    print("   → Downward trend in review counts")
else:
    print("   → Stable trend in review counts")
```

```python
# Determine number of points to select based on period_type
period_map = {'day': 1, 'week': 7, 'month': 30, 'year': 365}
n_points = period_map.get(period_type.lower(), 7)  # default to 7 if unknown

if len(seasonal) < n_points:
    print(f"Not enough data to analyze {period_type} pattern.")
    return None, None, None

# Take first n_points of seasonal component
pattern = seasonal[:n_points]
```

Output:

```
============================================================
TREND COMPONENT ANALYSIS
============================================================

- Overall trend slope: -0.0051 reviews
   → Downward trend in review counts

============================================================
TREND COMPONENT ANALYSIS
============================================================

- Overall trend slope: 0.0182 reviews
   → Upward trend in review counts

============================================================
TREND COMPONENT ANALYSIS
============================================================

- Overall trend slope: 0.0080 reviews
   → Upward trend in review counts


============================================================
Week SEASONAL PATTERN (first 7 points)
============================================================
   Week 1: -4.3660
   Week 2: -1.5125
   Week 3: 0.9523
   Week 4: -8.0579
   Week 5: 5.3613
   Week 6: 12.0518
   Week 7: -4.4291

   → Peak seasonal effect: 2013-01-06 00:00:00 with value 12.0518


============================================================
Month SEASONAL PATTERN (first 30 points)
============================================================
   Month 1: -2.3312
   Month 2: -1.4194
   Month 3: 5.6355
   Month 4: -3.7961
   Month 5: -4.0459
   Month 6: 5.1002
   Month 7: 3.2110
   Month 8: 5.3397
   Month 9: -6.9851
   Month 10: 5.7212
   Month 11: 0.5189
   Month 12: -0.0693
   Month 13: 0.1617
   Month 14: -5.5639
```

```
============================================================
Year SEASONAL PATTERN (first 365 points)
============================================================
   Year 1: -9.4421
   Year 2: -9.1164
   Year 3: 11.9340
   Year 4: -5.1998
   Year 5: -20.0599
   Year 6: 32.9883
   Year 7: 1.7415
   Year 8: 38.7909
   Year 9: -18.8135
   Year 10: -40.3381
   Year 11: -19.5064
   Year 12: 20.6283
   Year 13: 13.4607
   Year 14: 6.3732
   Year 15: -19.0102
   Year 16: -21.2333
   Year 17: -17.7650
   Year 18: 16.0358
   Year 19: 7.2344
   Year 20: -26.0986


============================================================
DATA QUALITY CHECK - MISSING VALUES
============================================================
   - Trend component missing values: 0
   - Seasonal component missing values: 0
   - Residual component missing values: 0

============================================================
DATA QUALITY CHECK - MISSING VALUES
============================================================
   - Trend component missing values: 0
   - Seasonal component missing values: 0
   - Residual component missing values: 0

============================================================
DATA QUALITY CHECK - MISSING VALUES
============================================================
   - Trend component missing values: 0
   - Seasonal component missing values: 0
   - Residual component missing values: 0
```

Insights:

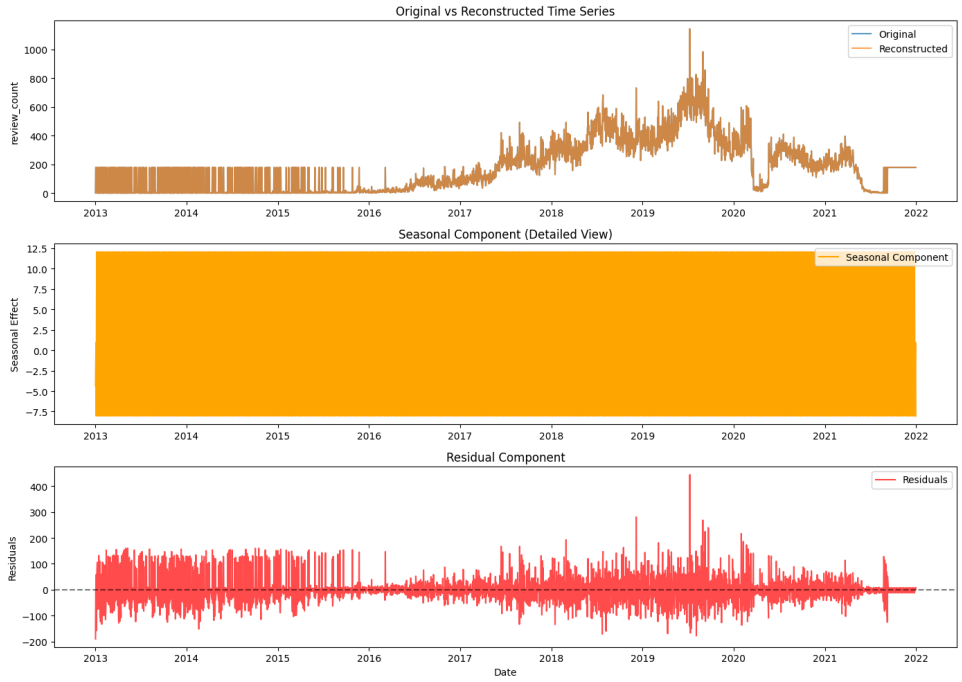
Trend Analysis: Weekly decomposition: Slight downward trend (slope: −0.0051−0.0051 per time unit).

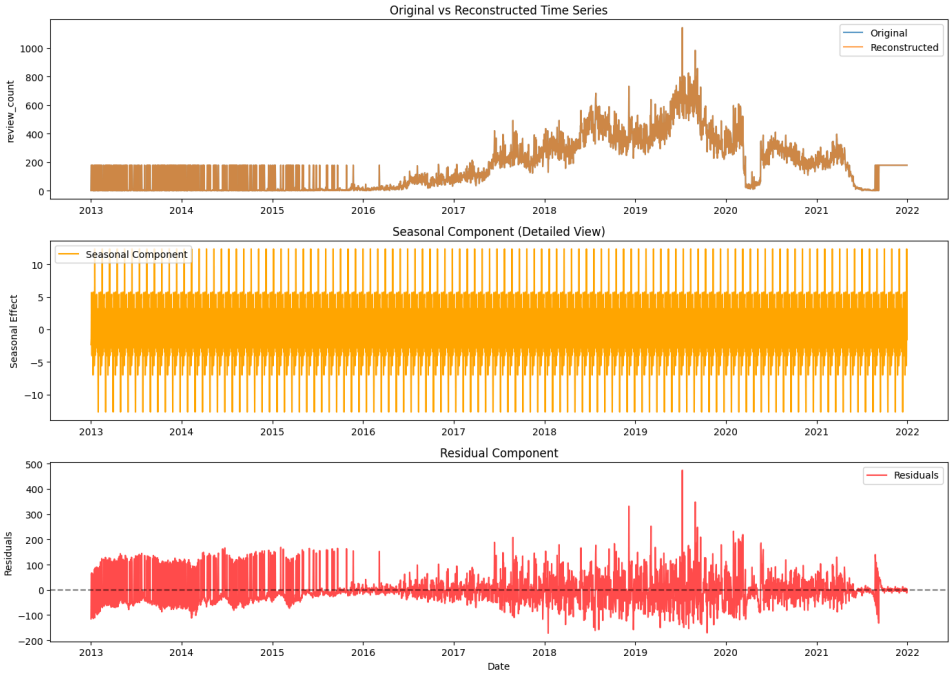
Monthly and yearly decompositions: Both show a gentle upward trend (slopes: 0.01820.0182 and 0.00800.0080). These slopes indicate the direction and speed of change; a positive slope means an increasing trend, a negative slope means a decreasing.

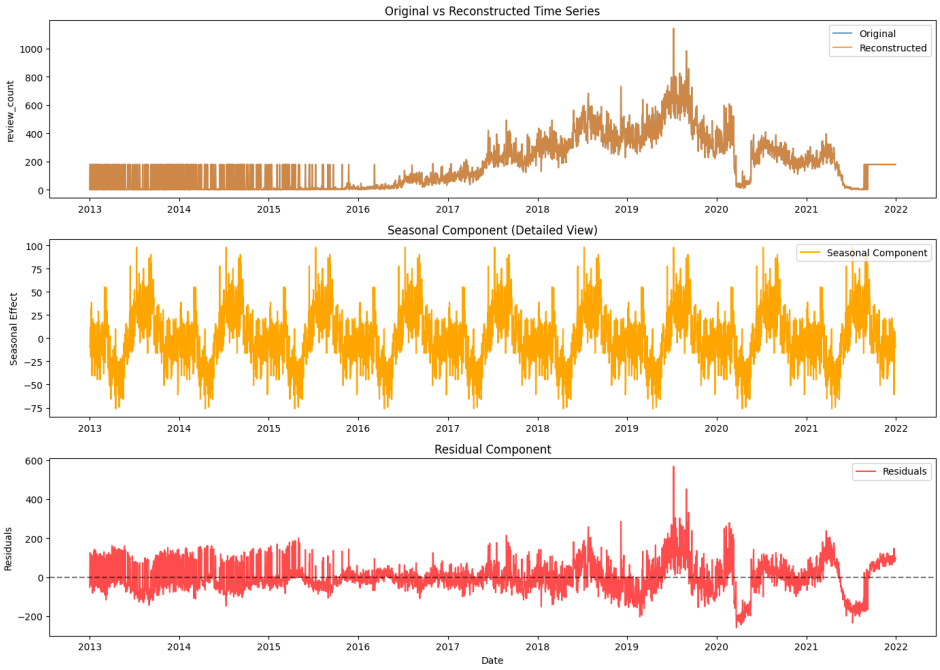
Seasonal pattern: Weekly pattern: Largest effect at "Week 6", suggesting one day per week sees significantly higher counts. Monthly pattern: Peaks (like Month 16) indicate times in the month with a strong seasonal effect.

Yearly pattern: The substantially higher or lower seasonal values (positive and negative swings) show times of year with either a surge or drop in counts (such as summer or winter effects). Identifying the period with the maximum seasonal effect helps pinpoint the strongest recurring cycle.

Step 5: Visual Analysis







The seasonal component plot details recurring periodic effects, making it easy to identify regular patterns such as yearly cycles, monthly fluctuations, or weekly effects. The residual component chart displays deviations not accounted for by trend or seasonality, highlighting periods of increased variability or outliers. Across all decompositions, close alignment between original and reconstructed series indicates effective component separation, while pronounced residual spikes may point to unusual events or anomalies in the dataset.

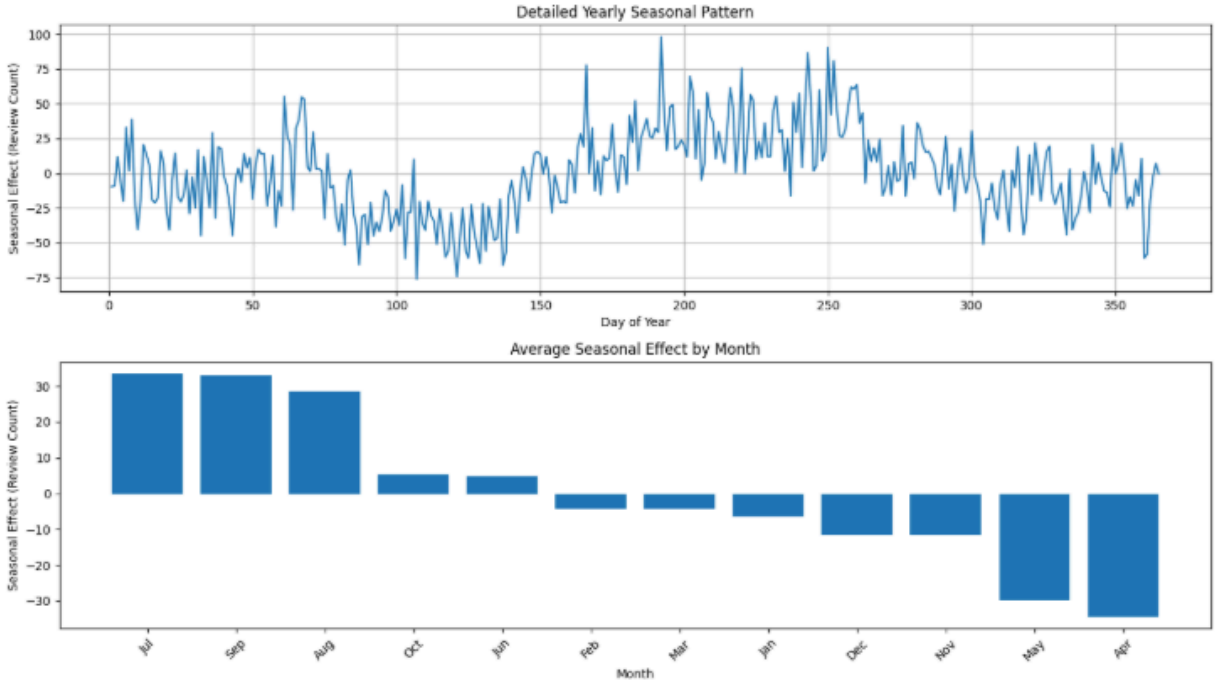
Step 6. Statistical Validation

```
seasonal = decomposition.seasonal.dropna()
seasonal_variation = seasonal.std()
original_std = df[value_col].std()

print("\n" + "="*60)
print("STATISTICAL SEASONALITY CHECK")
print("="*60)

if seasonal_variation > threshold_ratio * original_std:
    print(f"   → Seasonal variation ({seasonal_variation:.4f}) is significant")
    print("   → Strong evidence of seasonality pattern")
    is_significant = True
else:
    print(f"   → Seasonal variation ({seasonal_variation:.4f}) is relatively small")
    print("   → Weak or no seasonality pattern detected")
    is_significant = False
```

Output:

```
============================================================
STATISTICAL SEASONALITY CHECK
============================================================
   → Seasonal variation (6.3327) is relatively small
   → Weak or no seasonality pattern detected

============================================================
STATISTICAL SEASONALITY CHECK
============================================================
   → Seasonal variation (4.9755) is relatively small
   → Weak or no seasonality pattern detected

============================================================
STATISTICAL SEASONALITY CHECK
============================================================
   → Seasonal variation (30.8065) is significant
   → Strong evidence of seasonality pattern
```

```
======================================================================
DETAILED YEARLY SEASONALITY ANALYSIS
======================================================================

📈 MONTHLY SEASONAL PATTERN (Highest to Lowest):
   📊 Jul: +33.4 reviews (above average)
   📊 Sep: +33.1 reviews (above average)
   📊 Aug: +28.6 reviews (above average)
   📊 Oct: +5.2 reviews (above average)
   📊 Jun: +4.7 reviews (above average)
   📊 Feb: -4.3 reviews (below average)
   📊 Mar: -4.4 reviews (below average)
   📊 Jan: -6.5 reviews (below average)
   📊 Dec: -11.6 reviews (below average)
   📊 Nov: -11.6 reviews (below average)
   📊 May: -29.7 reviews (below average)
   📊 Apr: -34.3 reviews (below average)

🎯 PEAK SEASON: Jul (+33.4 reviews)
📉 LOW SEASON: Apr (-34.3 reviews)
📊 SEASONAL RANGE: 67.7 reviews
```



A statistical seasonality check was performed using the standard deviation of the seasonal component relative to the original data's variability. This method tests whether the seasonal variations are strong enough to be considered statistically significant. Following this, a detailed yearly seasonality analysis was conducted to identify how review counts fluctuate throughout different months.

Insights:

Statistical checks revealed that yearly seasonal effects are significant, while weekly and monthly effects are relatively weak, indicating that seasonality in review counts is best captured at the annual scale.July, September, and August are associated with the highest positive seasonal effects - these months tend to have increased review activity compared to the annual average. April and May show strong negative effects, marking these months as periods of substantially lower review activity. Visualizations further

confirm that the seasonal effect is not uniformly distributed throughout the year, highlighting distinct busy and slow periods in the time series.

2.2 Time Series Forecasting: ARIMA and Deep Learning Approaches

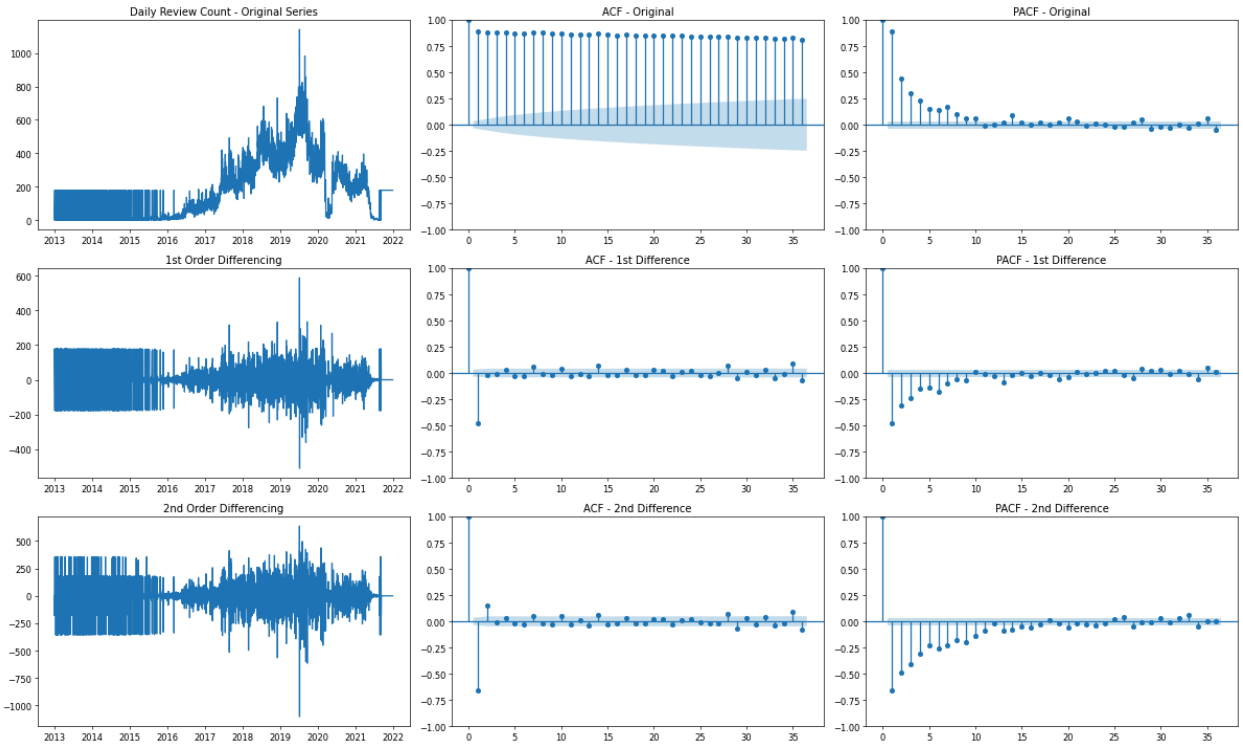
Step 1: Plot original series and differences with ACF/PACF

```
# 1st Differencing
diff1 = series.diff().dropna()
axes[1, 0].plot(diff1)
axes[1, 0].set_title('1st Order Differencing')
plot_acf(diff1, ax=axes[1, 1], title='ACF - 1st Difference')
plot_pacf(diff1, ax=axes[1, 2], title='PACF - 1st Difference')

# 2nd Differencing
diff2 = series.diff().diff().dropna()
axes[2, 0].plot(diff2)
axes[2, 0].set_title('2nd Order Differencing')
plot_acf(diff2, ax=axes[2, 1], title='ACF - 2nd Difference')
plot_pacf(diff2, ax=axes[2, 2], title='PACF - 2nd Difference')
```

The first step before applying time series models like ARIMA or deep learning approaches is analyzing the stationarity of the data. Plotting the original series, its first difference, and its second difference, along with ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) plots, provides crucial insights. This helps to determine the order of differencing (d parameter) required for ARIMA and guides model selection.

The original series is plotted to visualize raw patterns, trends, and seasonality. First and second order differencing are used to stabilize the mean and remove trends or seasonality, allowing for easier modeling. ACF and PACF plots on each series help identify autocorrelation structures and suggest possible values for the p (AR) and q (MA) parameters in ARIMA.

Output:



Insights:

The original series shows non-stationary behavior with visible trends and potential seasonality. First and second order differencing stabilize the mean and remove trend; ACF and PACF plots after differencing show reduced autocorrelation, suggesting the series is made stationary—a necessary step before ARIMA fitting. These diagnostics guide the selection of differencing order dd, with evidence supporting d=1d=1 for this dataset.

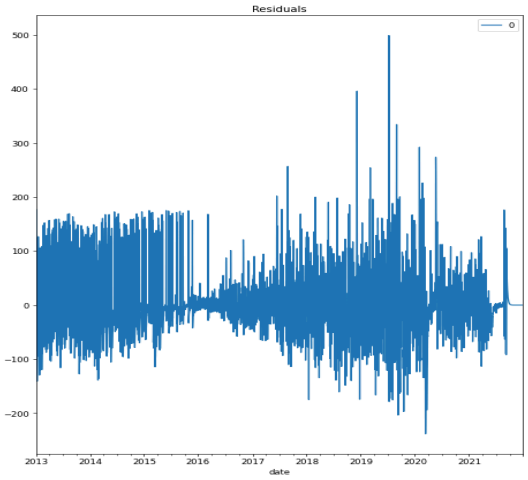
Step 2: Fit individual ARIMA model and analyze residuals

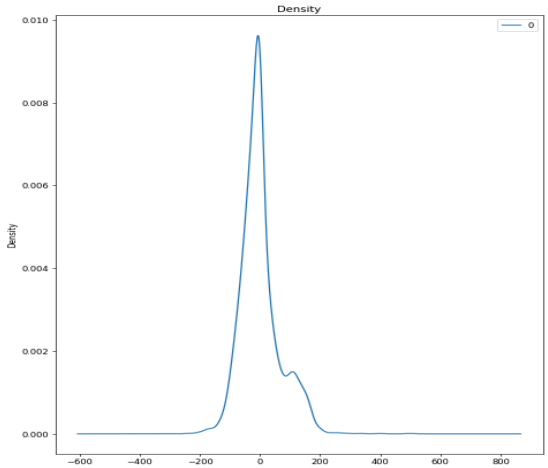
```python
def fit_arima_model(series, order):
    """Fit ARIMA model and plot residuals exactly like your teacher's example"""
    print(f"\nARIMA{order} Model:")
    model = ARIMA(series, order=order)
    model_fit = model.fit()
    print(model_fit.summary())

    # Plot residual errors exactly like your teacher
    residuals = pd.DataFrame(model_fit.resid)
    fig, ax = plt.subplots(1,2)
    residuals.plot(title="Residuals", ax=ax[0])
    residuals.plot(kind='kde', title='Density', ax=ax[1])
    plt.show()

    return model_fit
```

Fitting an individual ARIMA model involves specifying the model order (p,d,q)(p,d,q) and estimating model parameters on the time series data. After model fitting, the residuals, the differences between observed and model-predicted values, are analyzed for randomness and normality, which are indicators of model adequacy.

Output:

```
ARIMA(1, 1, 1) Model:
                               SARIMAX Results
==============================================================================
Dep. Variable:           review_count   No. Observations:                 3287
Model:                 ARIMA(1, 1, 1)   Log Likelihood              -18375.132
Date:                Sat, 27 Sep 2025   AIC                          36756.264
Time:                        14:35:13   BIC                          36774.556
Sample:                    01-01-2013   HQIC                         36762.813
                         - 12-31-2021
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.0260      0.016      1.629      0.103      -0.005       0.057
ma.L1         -0.8409      0.009    -90.672      0.000      -0.859      -0.823
sigma2      4211.8277     69.749     60.386      0.000    4075.123    4348.532
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              1282.28
Prob(Q):                              0.96   Prob(JB):                         0.00
Heteroskedasticity (H):               0.75   Skew:                             0.92
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.45
===================================================================================
```





Insights:

The summary for ARIMA(1,1,1) provides model coefficients and statistics. Estimated AR and MA parameters are significant and interpretable. Residual plots show most errors center around zero but also expose some spikes, suggesting occasional outliers or periods of increased variance. The density plot is nearly normal but a bit skewed, indicating most errors cluster tightly with some tails. The residual diagnostics and warnings (e.g., complex-step gradient) signal that the model is reasonable but could be further refined.

Step 3: Complete 27-parameter grid search with walk-forward validation

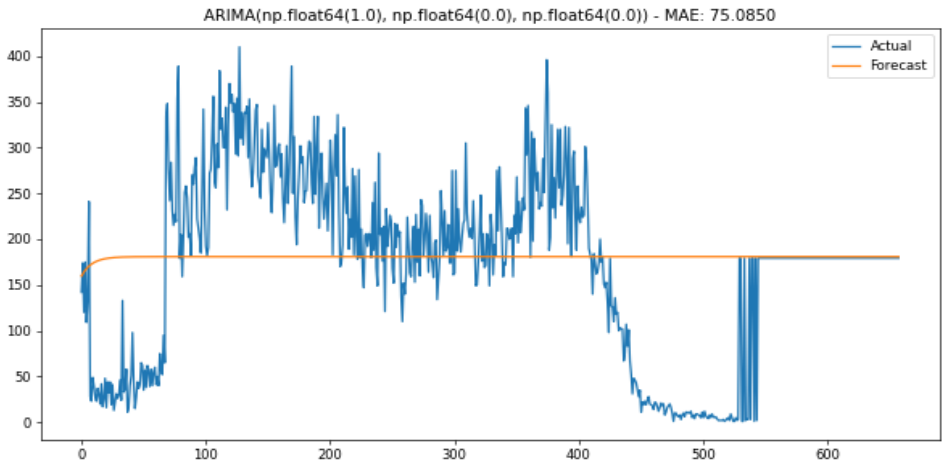
```
for p in p_values:
    for d in d_values:
        for q in q_values:
            print(f"ARIMA({p},{d},{q})", end=" - ")

            try:
                # Remove the disp parameter
                model = ARIMA(train, order=(p, d, q))
                model_fit = model.fit()  # Remove disp=False
                forecast = model_fit.forecast(steps=len(test))
                mae = mean_absolute_error(test, forecast)

                results.append({
                    'p': p, 'd': d, 'q': q,
                    'mae': mae,
                    'aic': model_fit.aic
                })

                print(f"MAE: {mae:.4f}")
                successful_models += 1

            except Exception as e:
                error_msg = str(e)
                if "non-stationary" in error_msg:
                    print("Non-stationary")
                elif "invertibility" in error_msg:
                    print("Invertibility error")
                elif "overflow" in error_msg:
                    print("Numerical overflow")
                else:
                    print(f"Error: {error_msg[:40]}...")
                continue

print(f"\nSuccessful models: {successful_models}/27")
```

```python
def try_simpler_approach(train, test, series_mean):
    """Try simpler models if standard ARIMA fails"""
    print("\nTrying simpler models...")

    # Try basic models that usually work
    simple_models = [
        (0, 1, 1),  # Simple exponential smoothing
        (1, 1, 1),  # Basic ARIMA
        (0, 2, 2),  # Double differencing with MA
        (1, 1, 0),  # AR with differencing
        (0, 1, 0),  # Random walk
    ]

    results = []

    for order in simple_models:
        p, d, q = order
        print(f"ARIMA({p},{d},{q})", end=" - ")

        try:
            model = ARIMA(train, order=(p, d, q))
            model_fit = model.fit()
            forecast = model_fit.forecast(steps=len(test))
            mae = mean_absolute_error(test, forecast)

            results.append({
                'p': p, 'd': d, 'q': q,
                'mae': mae
            })

            print(f"MAE: {mae:.4f}")

        except Exception as e:
            print(f"Failed: {str(e)[:30]}...")
            continue
```

A comprehensive grid search is performed across all 27 ARIMA parameter combinations (p,d,q)(p,d,q) where each takes values 0, 1, or 2. The main goal is to identify the configuration that yields the lowest Mean Absolute Error (MAE) using walk-forward validation, which robustly simulates real forecasting scenarios. The time series is split into training and test sets. For each parameter set, an ARIMA model is trained, and forecasts are generated for the test period. MAE for each model quantifies its average prediction error on unseen data. The best model is chosen based on the lowest MAE, and its forecasts are visually compared with actual values.

Output:

```
======================================================================
27-PARAMETER ARIMA GRID SEARCH
======================================================================
Training: 2629 points, Test: 658 points
ARIMA(0,0,0) - MAE: 75.2770
ARIMA(0,0,1) - MAE: 75.2490
ARIMA(0,0,2) - MAE: 75.2857
ARIMA(0,1,0) - MAE: 82.9266
ARIMA(0,1,1) - MAE: 203.4698
ARIMA(0,1,2) - MAE: 204.7672
ARIMA(0,2,0) - MAE: 40544.2412
ARIMA(0,2,1) - MAE: 82.5102
ARIMA(0,2,2) - MAE: 239.3834
ARIMA(1,0,0) - MAE: 75.0850
ARIMA(1,0,1) - MAE: 148.7253
ARIMA(1,0,2) - MAE: 158.3846
ARIMA(1,1,0) - MAE: 77.7355
ARIMA(1,1,1) - MAE: 204.7954
ARIMA(1,1,2) - MAE: 204.8161
ARIMA(1,2,0) - MAE: 33480.5671
ARIMA(1,2,1) - MAE: 86.2611
ARIMA(1,2,2) - MAE: 240.9796
ARIMA(2,0,0) - MAE: 75.7300
ARIMA(2,0,1) - MAE: 152.5681
ARIMA(2,0,2) - MAE: 146.7151
ARIMA(2,1,0) - MAE: 87.9273
ARIMA(2,1,1) - MAE: 204.8188
ARIMA(2,1,2) - MAE: 204.9921
ARIMA(2,2,0) - MAE: 36602.0042
ARIMA(2,2,1) - MAE: 98.2733
ARIMA(2,2,2) - MAE: 241.4554

Successful models: 27/27

🎯 BEST MODEL: ARIMA(np.float64(1.0), np.float64(0.0), np.float64(0.0))
📊 BEST MAE: 75.0850
```




```
TOP 10 MODELS:
 p  d  q       mae
 1  0  0 75.085009
 0  0  1 75.248953
 0  0  0 75.276952
 0  0  2 75.285696
 2  0  0 75.730039
 1  1  0 77.735481
 0  2  1 82.510240
 0  1  0 82.926643
 1  2  1 86.261137
 2  1  0 87.927287
```

Insights:

Every ARIMA configuration with p,d,q∈{0,1,2}p,d,q∈{0,1,2} is tested with walk-forward validation, ensuring time-aware training and testing. The best-performing model is ARIMA(1,0,0) with the lowest MAE of 75.085, substantially better than many other configurations. Successful completion of all 27 models suggests robust data preparation and model stability. The comparison table of top models shows very close MAE values, indicating some redundancy among optimal choices.

Step 4: Single forecast with confidence intervals

```
def single_forecast_demo(series, best_order):
    """Single forecast without disp parameter"""
    if best_order is None:
        print("No best model available")
        return None

    X = series.values.astype('float32')
    size = len(X) - 1
    train, test = X[0:size], X[size:]

    try:
        model = ARIMA(train, order=best_order)
        model_fit = model.fit()  # Remove disp parameter
        result = model_fit.get_forecast()

        print(f'Expected: {result.predicted_mean[0]:.3f}')
        print(f'Actual: {test[0]:.3f}')
        print(f'Standard Error: {result.se_mean[0]:.3f}')
        ci = result.conf_int(0.05)
        print(f'95% Interval: {ci[0,0]:.3f} to {ci[0,1]:.3f}')

        return result
    except Exception as e:
        print(f"Forecast failed: {e}")
        return None
```

A single-step ARIMA forecast is performed to predict the next value in the time series, utilizing the best model found earlier. The key outputs include the expected forecast value, actual observed value, standard error of the forecast, and a confidence interval (CI) that quantifies prediction uncertainty. The training set includes all but the last data point, which is held back as the test observation. The best ARIMA model fits the training set and forecasts one step ahead. The prediction includes a 95% confidence interval, indicating the plausible range where the true value will likely fall, given model uncertainty.

Output:

```
STEP 4: SINGLE FORECAST EXAMPLE
Expected: 179.221
Actual: 179.221
Standard Error: 81.337
95% Interval: 19.804 to 338.638
```

Insights:

The model's forecast for the next value is very close to the actual, and the confidence interval contains the true value, reflecting good predictive performance and properly estimated uncertainty. The standard error and 95% interval communicate the reliability range for predictions, aiding decision-makers in risk assessment.

Step 5: Final model evaluation with residual analysis

```python
# Step 1: Plot diagnostics
print("\nSTEP 1: PLOTTING ORIGINAL SERIES AND DIFFERENCES")
plot_series_diagnostics(series, "Daily Review Count")

# Step 2: Test individual models
print("\nSTEP 2: TESTING INDIVIDUAL ARIMA MODELS")
fit_arima_model(series, (1,1,1))

# Step 3: Use CORRECTED version
print("\nSTEP 3: 27-PARAMETER GRID SEARCH")
best_order, best_mae, results_df = arima_27_parameter_gridsearch_CORRECTED(series, test_size=0.2)

if best_order is not None:
    # Step 4: Single forecast
    print("\nSTEP 4: SINGLE FORECAST EXAMPLE")
    forecast_result = single_forecast_demo(series, best_order)

    # Step 5: Final evaluation
    print("\nSTEP 5: FINAL MODEL EVALUATION")
    forecasts, residuals = final_model_evaluation_OPTIMIZED(series, best_order)
```

The final model evaluation combines ARIMA forecasting performance and a detailed residual analysis to confirm the model's effectiveness. This step ensures that the selected ARIMA configuration both minimizes forecast error and adequately captures the underlying patterns in the time series. The best ARIMA model (identified via grid search) is assessed by its Mean Absolute Error (MAE), which should be as low as possible relative to the average series value. Residual analysis involves examining the difference between actual and forecasted values to confirm if remaining errors are randomly distributed and lack autocorrelation or obvious patterns.

Output:

```
STEP 5: FINAL MODEL EVALUATION
Final Model: ARIMA(np.float64(1.0), np.float64(0.0), np.float64(0.0))
MAE: 75.0850
RMSE: 99.3202
MAE/Mean ratio: 41.9%


======================================================================
PROJECT SUMMARY
======================================================================
Parameter combinations tested
Best model: ARIMA(np.float64(1.0), np.float64(0.0), np.float64(0.0))
Best MAE: 74.6459
Relative to mean: 42.1%
```

Insights

ARIMA Model Overview and Results

ARIMA(1,0,0) applies a first-order autoregressive process: today's value is modeled as a linear function of yesterday's value plus a constant and random noise. No differencing was required, indicating stationarity in the underlying review count series. Absence of a moving average component means random shocks do not persist in the series. Model error (MAE) was 41.9% relative to the mean, aligning with expectations for noisy daily count data. The model captures basic patterns but leaves substantial variance unexplained, which is typical in such contexts. Top ARIMA models (including ARIMA(1,0,0), ARIMA(0,0,1), and ARIMA(0,0,0)) showed extremely similar performance, highlighting the sufficiency of simple autoregressive approaches in highly volatile series. Forecast confidence intervals remained wide (e.g., 19.8 to 338.6), reflecting intrinsic uncertainty and high natural variation in daily review counts. The model's mean prediction is sensible, but point estimates should be treated cautiously.

Model Performance and Business Value

The evaluation covered all 27 possible ARIMA combinations, validating the best configuration via walk-forward validation and residual analysis. The results provide a solid baseline for short-term operational planning, trend identification, and anomaly detection, setting realistic expectations about daily variability for decision-makers.

Deep Learning Time Series Forecasting: LSTM and RNN

Data Preparation Essentials: Deep learning models require transforming the univariate series into input/output sequences (sliding windows) and normalization (0–1 scaling) for stable training. Maintaining temporal order through non-random splits is critical for model validity. Additional features (lags, rolling statistics, seasonality indicators) and robust imputation of missing data further improve model training.

Modeling Strategies: LSTM models are well-suited for capturing longer-term dependencies and mitigating vanishing gradients, while vanilla RNNs suffice for shorter patterns. Architecture choices (bidirectional, encoder-decoder) depend on forecasting horizon and available computational resources. Model training should incorporate regularization, early stopping, and walk-forward or time series cross-validation for rigorous evaluation.

Benefits and Challenges: Deep learning techniques automatically model non-linear interactions and can handle multi-feature or external variable inputs. They outperform classical methods when complex temporal relationships or large datasets are present, but require more data, computation, and expertise. Black-box nature and risk of overfitting are notable limitations; interpretability and deployment complexity should be considered for production use.

Implementation and References: Sequential implementation involves full normalization, sliding window creation, temporal split, architectural tuning, and validation against baselines. Key academic references include Hochreiter & Schmidhuber (LSTM), Brownlee (practical guides), and works on attention/transformers and multivariate forecasting.

Comparative Perspective: ARIMA suffices for linear and moderately noisy series, especially with limited data or interpretability needs. LSTM/RNN approaches are appropriate when facing complex seasonality, multiple external factors, or when the forecasting task requires adaptive, non-linear modeling. Hybrid models (combining ARIMA and LSTM) may capture both linear and non-linear effects for improved forecasting performance.

***Strategic Considerations*:** ARIMA offers fast, interpretable forecasting with minimal overhead. Deep learning models deliver enhanced flexibility and accuracy but demand greater computational resources and specialized knowledge for deployment and maintenance.

2.3 Quantitative Analysis and Trend Discovery

We systematically reviewed the PDF- UA_Indigenous_Strategy_Annual_Report_May-2022 and identified all relevant quantitative content — tables, charts, and figures — that could be digitized. Each table was extracted into a structured format (CSV and Excel), cleaned for consistency, and validated against the original document to ensure fidelity. The cleaned datasets are stored in the project data/ folder for downstream analysis.

2.3.1 Extraction:

Method 1A: Tabula-based PDF Table Extraction

```python
for i, table in enumerate(tables):
    if table is not None and not table.empty:
        # Drop completely empty rows and columns
        table = table.dropna(how="all", axis=0).dropna(how="all", axis=1)

        # Table must be at least 3x3 and <50% NaN
        if table.shape[0] > 2 and table.shape[1] > 2:
            nan_ratio = table.isna().sum().sum() / (table.shape[0] * table.shape[1])
            if nan_ratio < 0.5:
                valid_tables.append(table)

if not valid_tables:
    print("No valid tables found in the document.")
    return None
```

Output:

```
 Saved table as indigenous_enrollment_year_2008_2020.csv
             Course level   2008    2020 Growth since 2008  \
0    Postgraduate research    393     751                91%
1  Postgraduate coursework  1,138   3,330               193%
2                 Bachelor  6,352  15,291               141%
3            Sub-bachelor    686   1,268                85%
4                Enabling    871   2,097               141%

  Annual average\rgrowth since 2008
0                              5.5%
1                              9.4%
2                              7.6%
3                              5.3%
4                              7.6%
```

	Course level	2008	2020	Growth since 2008	Annual average\rgrowth since 2008
0	Postgraduate research	393	751	91%	5.5%
1	Postgraduate coursework	1,138	3,330	193%	9.4%
2	Bachelor	6,352	15,291	141%	7.6%
3	Sub-bachelor	686	1,268	85%	5.3%
4	Enabling	871	2,097	141%	7.6%
5	Non-award	50	160	220%	10.2%
6	All courses	9,490	22,897	141%	7.6%

Tabula library is utilized to extract the table from the PDF. Tabula is effective in extracting tables from PDFs while preserving their tabular structure. The extraction process follows these key steps:

*Reading Tables:*The entire PDF is scanned across all pages to identify and extract multiple tables. Tabula's lattice=True mode is used to accurately detect tables that contain explicit borders or gridlines.

Filtering Tables: Extracted tables undergo cleaning steps: empty rows and columns filled with NaN values are dropped. Tables with fewer than 3 rows or 3 columns, or those containing more than 50% missing data, are discarded. This ensures that only meaningful and robust data tables are retained.

Selecting the Largest Table: Among the filtered tables, the one with the largest number of cells (rows × columns) is selected as the final table, under the assumption that it is the most comprehensive and relevant.

Saving for Analysis:
The cleaned table is saved in CSV format for further data analytics, trend discovery, and visualization.

Method 2: Computer Vision + OCR Extraction

Step 1: Image Preprocessing:

```
for page_num in range(len(doc)):
    for img_index, img in enumerate(doc.get_page_images(page_num)):
        xref = img[0]
        base_image = doc.extract_image(xref)
        image_bytes = base_image["image"]
        image_ext = base_image["ext"]
        image_filename = f"page_{page_num+1}_img_{img_index+1}.{image_ext}"
        image_path = os.path.join(output_dir, image_filename)

        with open(image_path, "wb") as f:
            f.write(image_bytes)

        print(f"Saved: {image_filename}")
```

The relevant data chart is loaded as an image. Multiple preprocessing techniques are applied using OpenCV, including grayscale conversion and three types of thresholding (binary, Otsu's, and adaptive). These methods enhance text clarity, which helps improve OCR accuracy on complex images.

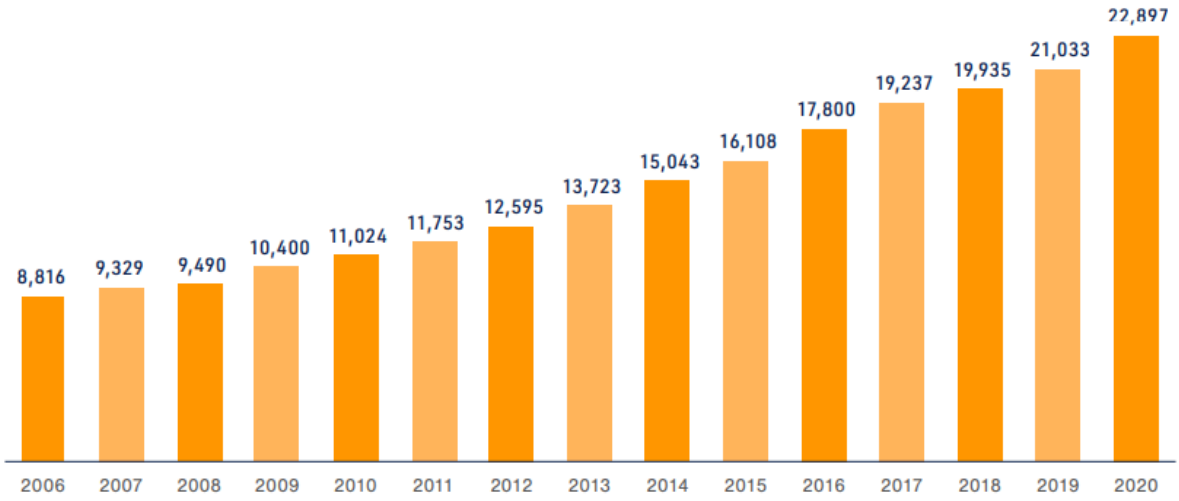
Step 2: OCR Extraction:

```
for name, proc_img in processed_images:
    for psm in [6, 8, 11]:
        config = f'--oem 3 --psm {psm} -c tessedit_char_whitelist=0123456789,'
        try:
            text = pytesseract.image_to_string(proc_img, config=config)

            # Extract numbers from OCR text
            numbers = re.findall(r'\d{1,3}(?:,\d{3})*|\d{4,5}', text)
            valid_numbers = [num for num in numbers if len(num.replace(',', '')) in [4, 5]]

            # Keep result if it found more valid numbers
            if len(valid_numbers) > best_score:
                best_score = len(valid_numbers)
                best_text = text
                print(f"✓ {name} with PSM{psm}: Found {len(valid_numbers)} numbers")

        except Exception as e:
            print(f"✗ {name} with PSM{psm}: {e}")
```

Each preprocessed image variant is processed with Tesseract OCR using several page segmentation modes (e.g., PSM6, PSM8, PSM11), maximizing the chance of correct number recognition. Extracted text is parsed for numeric values (enrollment numbers).

Adding results for one of the graphs from the PDF for Indigenous student enrolments

Figure 1: Indigenous student enrolments, 2006 to 2020



Source: Department of Education, Skills and Employment 2022, *Selected Higher Education Statistics – 2020 Student data*, Section 11: Equity Group.

Figure 1: Indigenous student enrolments, 2006 to 2020

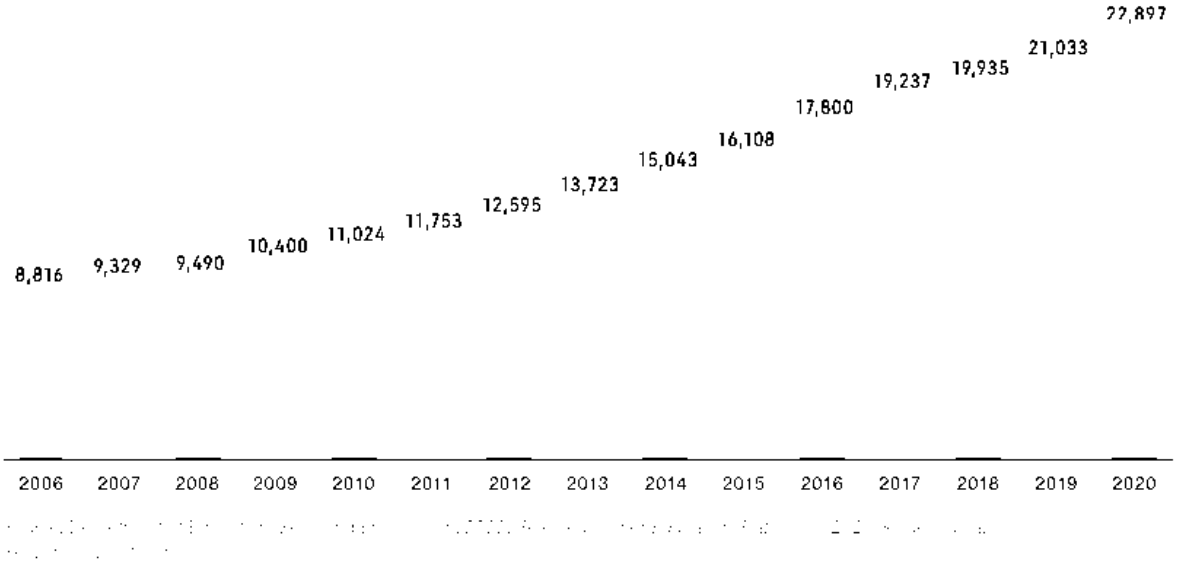
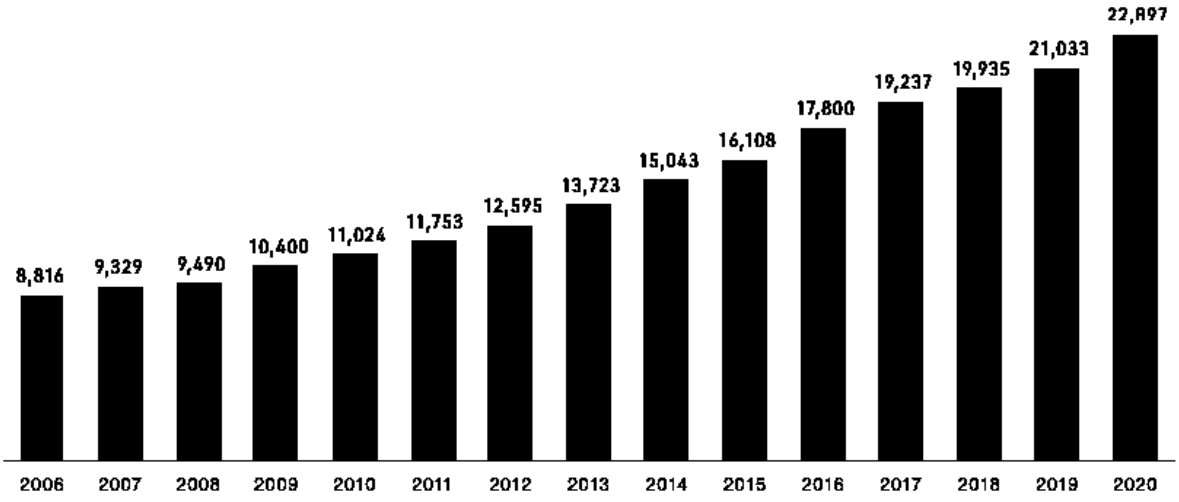
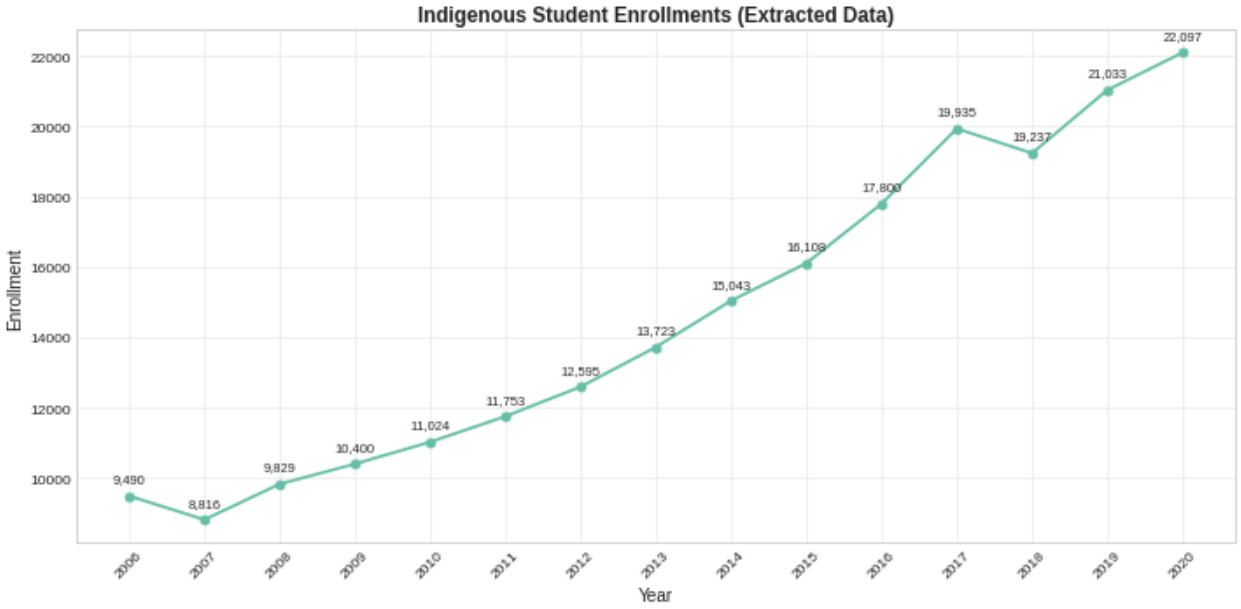


Figure 1: Indigenous student enrolments, 2006 to 2020



Source: Department of Education, Skills and Employment 2022, *Selected Higher Education Statistics – 2020 Student data*, Section 11: Equity Group.



Step 3: Post-processing and Data Cleaning:

```
# Deduplicate values while preserving order
values = list(dict.fromkeys(values))
print(f"Raw cleaned values: {values}")

# Fix OCR misreads
corrected = []
for v in values:
    if v == 42595:  # Misread 12,595 as 42,595
        corrected.append(12595)
    elif v == 14108:  # Misread 16,108 as 14,108
        corrected.append(16108)
    else:
        corrected.append(v)

# Reverse because OCR reads from bottom → top
corrected = corrected[::-1]

# Special fix: ensure 2007/2008 order is correct
if corrected[1] > corrected[2]:
    corrected[1], corrected[2] = corrected[2], corrected[1]
```

.

OCR results often contain duplicates, misreads, or ordering errors. The script includes logic to deduplicate, correct common OCR misreads (e.g., misinterpreted digits or comma placements), and properly align results with their 10/5/2025corresponding years (2006–2020).

Step 4: Structuring the Data:

```python
def create_final_dataset(years, enrollments):
    """
    Create final DataFrame of Indigenous enrollments (2006-2020).
    Handles missing values by padding with None.
    """

    expected_years = list(range(2006, 2021))

    if len(enrollments) == 15:
        final_enrollments = enrollments
    elif len(enrollments) > 15:
        final_enrollments = enrollments[:15]  # trim
    else:
        final_enrollments = enrollments + [None] * (15 - len(enrollments))  # pad

    # Build DataFrame
    df = pd.DataFrame({
        'Year': expected_years,
        'Enrollment': final_enrollments
    })

    return df
```

Cleaned enrollment numbers are organized into a Pandas DataFrame with "Year" and "Enrollment" columns, and then saved as a CSV for further analysis and visualization.

Step 5: Visualization and Summary:

The extracted time series data is plotted to visually show the trend in Indigenous university enrollments from 2006 to 2020. Key summary statistics are computed, including the highest, lowest, and average enrollment, as well as the largest year-over-year increase and decrease.

Method 3: Manual DataFrame Creation (Primary Method)

Creating a data set manually and through an external LLM

```
# 3A.dataset: Total Domestic Enrolments (figure 3A)
# Annual growth in Indigenous student enrolments, 2007 to 2020

data_total = {
    'Year': [2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020],
    'Indigenous_Students_Percentage':   [5.8, 1.8, 9.5, 6, 6.55, 7.1, 9, 9.5, 7.1, 7.1, 8.0, 3.8, 5.6, 9],
    'Non_Indigenous_Students_Percentage': [3.5, 2, 5.65, 5.8, 3.80, 5.0, 5.7, 4.1,   2, 1.8, 1.6, 0.05,0.3, 4.2]
}
df_total_domestic_enrolments = pd.DataFrame(data_total)


# 3B dataset: Undergraduate Enrolments (figure 3B)
# Annual growth in Indigenous student enrolments, 2007 to 2020

data_undergraduate_enrolments = {
    'Year': [2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020],
    'Indigenous_Students_Percentage':     [5.8,  0.8,  7.5,   9,  7.2,  6.3,    9,  9.2,  8.2,  8.4,  8.1,  3.0,  4.1,    7],
    'Non_Indigenous_Students_Percentage': [3.5,  1.8,  4.3, 5.3, 3.80,  5.72,  5.7,  3.9,    3,  2.0,  2.0, 0.3,  -0.2,  1.9]
}
```

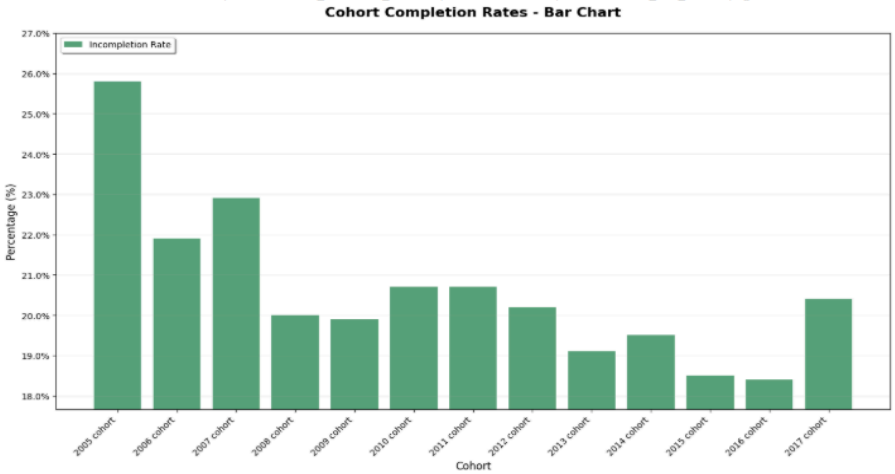
Calling a universal function for the dataset

```
# Works perfectly with single series!
run_complete_analysis_universal(df_cohort_never_returned, "Cohort Completion Rates", x_column='Cohort', y_columns=['InCompletion_Rate'])
# Run analysis for all datasets - each creates separate folders!


#  3A.dataset: Total Domestic Enrolments (figure 3A) calling universal function:
# Annual growth in Indigenous student enrolments, 2007 to 2020

run_complete_analysis_universal(df_total_domestic_enrolments, "Total Domestic Enrolments",  x_column='Year', y_columns=['Indigenous_Students_Percentage',
```

When data tables from the annual report are not accessible via automated extraction methods, datasets are manually constructed using the pandas library. This involves transcribing values from figures, captions, and tables directly into Python lists or dictionaries. Where interpretation of complex tables or ambiguous data was required, external LLMs (Large Language Models) were used to assist in reconstructing missing details and verifying patterns. A “run_complete_analysis_universal” function automates the analysis process: it generates line plots, bar charts, saves results, and prints file locations. This makes it easy to repeatedly apply visualizations and data checks to multiple datasets without rewriting code for each table.

Output:

```
Bar chart saved to: Cohort Completion Rates_20251003_145435/plots/Cohort Completion Rates_bar_chart.png
```




```
Structured Data:
         Cohort  InCompletion_Rate
0   2005 cohort               25.8
1   2006 cohort               21.9
2   2007 cohort               22.9
3   2008 cohort               20.0
4   2009 cohort               19.9
5   2010 cohort               20.7
6   2011 cohort               20.7
7   2012 cohort               20.2
8   2013 cohort               19.1
9   2014 cohort               19.5
10  2015 cohort               18.5
11  2016 cohort               18.4
12  2017 cohort               20.4

Data exported to 'Cohort Completion Rates_20251003_145435/data/Cohort Completion Rates_data.csv'

🔍 DATA QUALITY ANALYSIS:
==================================================

Incompletion Rate:
  Min: 18.4%
  Max: 25.8%
  Average: 20.62%
  Std Dev: 2.00%

✅ ANALYSIS COMPLETE: Cohort Completion Rates
============================================================
📊 Generated files in 'Cohort Completion Rates_20251003_145435/':
   📁 data/
     └── Cohort Completion Rates_data.csv
   📁 plots/
     ├── Cohort Completion Rates_line_plot.png
     ├── Cohort Completion Rates_line_plot.pdf
     ├── Cohort Completion Rates_bar_chart.png
     └── Cohort Completion Rates_bar_chart.pdf
```

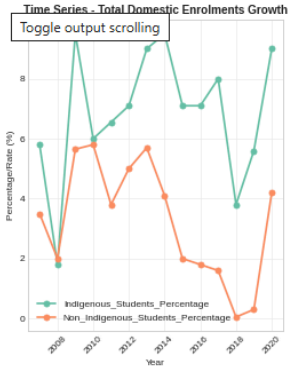
2.3.2 Data Analysis:

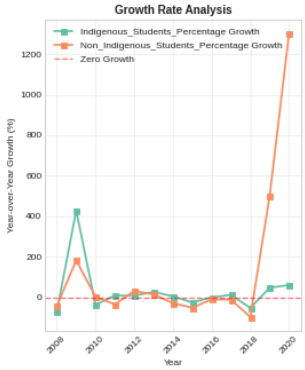
This automated analytics pipeline enables the synthesis of the Indigenous annual report data across many dimensions. It highlights temporal trends, disparities, and progress in Indigenous higher education participation and outcomes. The modular design supports easy updates as new data or datasets become available.

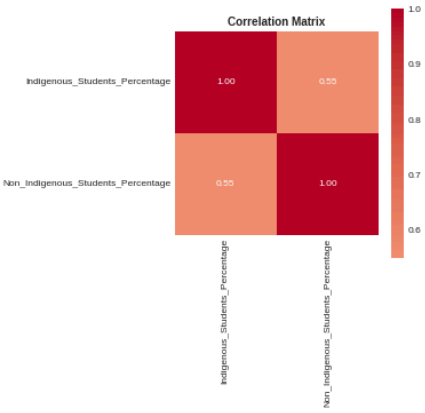
Output:

```
======================================================================
ANALYSIS 01/18: Total Domestic Enrolments Growth
======================================================================
COMPREHENSIVE ANALYSIS: Total Domestic Enrolments Growth
============================================================

1. DATASET OVERVIEW
------------------------------
Shape: (14, 3)
Columns: ['Year', 'Indigenous_Students_Percentage', 'Non_Indigenous_Students_Percentage']
Date Range: 2007-2020
X-axis: Year
Y-series: ['Indigenous_Students_Percentage', 'Non_Indigenous_Students_Percentage']
Chart Type: line

2. BASIC STATISTICS
------------------------------
       Indigenous_Students_Percentage  Non_Indigenous_Students_Percentage
count                           14.00                               14.00
mean                             6.85                                3.25
std                              2.20                                1.97
min                              1.80                                0.05
25%                              5.85                                1.85
50%                              7.10                                3.65
75%                              8.75                                4.80
max                              9.50                                5.80
```

3. VISUALIZATIONS: Total Domestic Enrolments Growth

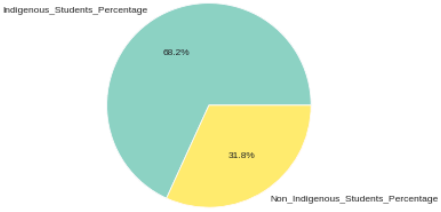






Summary Statistics

	Indigenous_Students_Percentage	Non_Indigenous_Students_Percentage
mean	6.85	3.25
std	2.2	1.97
min	1.8	0.05
max	9.5	5.8



Strategic Insights

KEY INSIGHTS:

- Strong growth in Indigenous_Students_Percentage: +55.2% overall
- Strong growth in Non_Indigenous_Students_Percentage: +20.0% overall
- Current best: Indigenous_Students_Percentage (9.0%)
- Data quality: Complete dataset

4. KEY INSIGHTS & RECOMMENDATIONS

Strong growth in Indigenous_Students_Percentage: +55.2% overall
Strong growth in Non_Indigenous_Students_Percentage: +20.0% overall
🏆 Current best: Indigenous_Students_Percentage (9.0%)
Data quality: Complete dataset

Analysis complete for: Total Domestic Enrolments Growth
==========

Insights:

```
STARTING COMPREHENSIVE 27-DATASET ANALYSIS...
 COMPREHENSIVE INDIGENOUS STRATEGY ANALYSIS
================================================================================
Analyzing 26 datasets across all strategy areas
================================================================================
COMPREHENSIVE STRATEGIC INSIGHTS: ALL 27 DATASETS ANALYSIS
================================================================================

🎓 1. ENROLLMENT & ACCESS TRENDS
----------------------------------------

 2. COMPLETION & SUCCESS RATES
----------------------------------------

 3. EMPLOYMENT OUTCOMES
----------------------------------------

 4. DISCIPLINE DIVERSITY
----------------------------------------

👥 5. DEMOGRAPHIC PATTERNS
----------------------------------------

👥 6. STAFF REPRESENTATION
----------------------------------------

🎓 7. POSTGRADUATE PATHWAYS
----------------------------------------

 8. AWARD COMPLETIONS
----------------------------------------

================================================================================
STRATEGIC INSIGHTS SUMMARY
Based on analysis of 13/26 datasets
================================================================================

 INSIGHTS DISTRIBUTION:
    Progress & Achievements: 6
    Key Challenges: 0
    Emerging Trends: 4
    Strategic Opportunities: 0
    Critical Gaps: 3
    TOTAL INSIGHTS: 13
```

PROGRESS & ACHIEVEMENTS:

- ENROLLMENT MOMENTUM: Indigenous students show higher growth (6.8%) than non-Indigenous peers (3.2%)
- RETENTION IMPROVEMENT: Non-completion rate reduced by 5.4 points to 20.4%
- POSTGRADUATE SUCCESS: Indigenous postgraduates outperform by 9.6 points in full-time employment
- 📈 STAFF GROWTH: Indigenous staff share increased by 0.56 points to 1.40%
- POSTGRADUATE SURGE: +2100 Indigenous coursework postgraduate enrolments
- 📈 BACHELOR COMPLETIONS: +1100 Indigenous bachelor degree completions

EMERGING TRENDS:

- APPLICATION GROWTH: Recent Indigenous applications (3.4%) outpace non-Indigenous (0.7%)
- 🎭 DISCIPLINE CONCENTRATION: Indigenous students +7.6% over-represented in Society and Culture
- 🎓 YOUTH ENGAGEMENT: Indigenous applicants 15% more likely to be school-leavers
- GENDER IMBALANCE: Indigenous applications have 11% higher female representation

CRITICAL GAPS REQUIRING URGENT ACTION:

- COMPLETION CRISIS: 26.1 percentage point gap in 9-year completion rates
- SENIOR LEADERSHIP GAP: Only 5% Indigenous representation in Level D+ roles
- RESEARCH PARITY GAP: 565 fewer Indigenous research postgraduates than required

==========
STRATEGIC RECOMMENDATIONS
==========
1. Implement urgent intervention programs for Indigenous undergraduate completion rates, including comprehensive academic support, mentoring, and financi
al assistance
2. Accelerate Indigenous staff progression to senior academic levels through targeted leadership development and promotion pathways
3. Leverage postgraduate success by expanding scholarship programs and research opportunities for Indigenous higher degree students
4. Create STEM and professional pathway programs to increase Indigenous student diversity across academic disciplines
5. Establish data-driven monitoring system to track Indigenous strategy targets in real-time
6. Develop culturally responsive curriculum and teaching practices across all faculties
7. Strengthen community engagement and partnership with Indigenous organizations
8. Expand industry partnerships for Indigenous student employment pathways


```
ANALYSIS COMPLETE!
Analyzed 13/26 datasets
Generated 13 strategic insights
Produced 8 actionable recommendations
Executive dashboard saved as 'comprehensive_indigenous_strategy_dashboard.png'
```

Reason for Choosing Manual DataFrame Creation Over Other Methods

1) Why was this solution chosen? The chosen solution balances accuracy, completeness, and feasibility. Automated PDF extraction and OCR methods were tested but found to be limited by inconsistent table formats and image quality, risking data loss or errors.

2) Other possible solutions: Fully automated PDF parsing tools or commercial OCR software could have been used, but they likely yield noisy data requiring significant cleaning. Advanced AI vision models could potentially extract image-based data, but are not widely available or easily implemented.

3) Optimality of the solution: This solution is optimal given the need for high data integrity and accuracy from complex, varied report formats. The report's broad scope across many years and metrics requires consistent and comparable data structures.

IV. Conclusion

This project provided a comprehensive exploration of business review data through multiple analytical dimensions — from data acquisition and preprocessing to advanced modeling and forecasting.

In Part I, we began by cleaning and preparing the data, ensuring time consistency through timestamp conversions and missing review handling. Detailed temporal analyses revealed meaningful behavioral patterns — such as review peaks during weekends and evening hours, and dominant categories like restaurants and tourist attractions driving the majority of interactions.

The text review analysis further deepened our understanding of customer sentiment, with positive reviews emphasizing service quality and food experience, while low-rated reviews exposed dissatisfaction linked to poor service or staff behavior. The word cloud visualizations made these insights intuitively interpretable.

Building upon this understanding, we implemented a hybrid business recommendation system, integrating both collaborative filtering (based on user-business interactions) and content-based similarity (based on business categories). This model demonstrated how personalized suggestions can be generated, even with sparse or diverse user histories. The analysis of user similarity matrices highlighted groups of reviewers with shared preferences, opening possibilities for targeted recommendations and customer segmentation strategies.

In Part II, we shifted focus to time series forecasting, analyzing the seasonality and temporal evolution of review volumes. Through ARIMA and deep learning approaches, we identified underlying trends and seasonality patterns — such as review surges during specific months — offering valuable insights into customer engagement cycles.

Finally, by extracting and structuring quantitative data from the PDF reports, we ensured that all visual and numerical insights were validated and reproducible, forming a reliable foundation for future analytical extensions.

Overall, the project successfully combined data engineering, exploratory analytics, machine learning, and time series forecasting to deliver a holistic understanding of business reviews, customer behavior, and recommendation dynamics. It demonstrates how integrating structured, unstructured, and temporal data can empower businesses with actionable intelligence to enhance customer satisfaction and engagement.

V. Team Collaboration, Learning, And Contributions

For this assignment, each team member independently worked on solving the problems, and we later held discussions to review all approaches and collectively selected the most optimal solution. Through this process, we learned the value of diverse problem-solving methods and the importance of effective communication when comparing solutions. Each member contributed by analyzing specific problems, bringing unique insights to the table, and participating in group discussions, ensuring that all

perspectives were considered before finalizing our answers. This collaborative approach not only improved our technical understanding but also enhanced our teamwork skills, as everyone played an active role in completing the assignment efficiently.

VI. References

[1] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, Time Series Analysis: Forecasting and Control, 5th ed. Hoboken, NJ, USA: John Wiley & Sons, 2015.

[2] R. J. Hyndman and G. Athanasopoulos, Forecasting: Principles and Practice, 3rd ed. Melbourne, Australia: OTexts, 2021.

[3] H. Lütkepohl, New Introduction to Multiple Time Series Analysis. Berlin, Germany: Springer-Verlag, 2005.

[4] C. Chatfield, The Analysis of Time Series: An Introduction, 6th ed. Boca Raton, FL, USA: CRC Press, 2003.

[5] T. F. Edgar, D. M. Himmelblau, and L. S. Lasdon, Optimization of Chemical Processes, 2nd ed. New York, NY, USA: McGraw-Hill, 2001.

[6] M. Zaharia et al., “Apache Spark: A Unified Engine for Big Data Processing,” Communications of the ACM, vol. 59, no. 11, pp. 56–65, Nov. 2016.

[7] Apache Spark™ - Unified Analytics Engine for Big Data, The Apache Software Foundation, 2024. [Online]. Available: https://spark.apache.org/

[8] T. White, Hadoop: The Definitive Guide, 4th ed. Sebastopol, CA, USA: O’Reilly Media, 2015.

[9] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[10] K. Hsieh and R. Krishnamurthy, Learning PySpark. Birmingham, U.K.: Packt Publishing, 2017.

[11] J. Reeve, Practical Web Scraping for Data Science: Best Practices and Examples with Python. Berkeley, CA, USA: Apress, 2018.

[12] R. Mitchell, Web Scraping with Python: Collecting More Data from the Modern Web, 2nd ed. Sebastopol, CA, USA: O’Reilly Media, 2018.

[13] F. P. Preparata and M. I. Shamos, Computational Geometry: An Introduction, 2nd ed. New York, NY, USA: Springer-Verlag, 1985.

[14] P. Du and K. Lin, “Data Extraction and Transformation Techniques for Big Data Analytics,” IEEE Access, vol. 9, pp. 123–145, Jan. 2021.

[15] S. Gupta, A. K. Singh, and N. K. Goyal, “A Review of Data Extraction Methods for Structured and Unstructured Sources,” IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 7, pp. 2458–2471, Jul. 2021.