PART 4: “Comprehensive Case Study – Recipe Review Ratings Prediction”

By:

Fatema Islam Surovi

(NF1008944)

Master of Data Analytics, University of Niagara Falls

Summer 2025

Python for Data Analysis (CPSC-610-11)

Professor: Omid Isfahanialamdari

Sep 14, 2025

Introduction

This project implements 1-5 star ratings of reviews of recipes. I cleaned the data, studied some important trends, two trained models (Logistic Regression and Random Forest with balancing of the classes), engineered features, and compared performance accuracy, precision/recall/F1, confusion matrices and ROC-AUC. The ratings are extremely disproportionate to 5 stares. Community involvement is an indicator, especially *dislikes*, emerged as the strongest predictors of low ratings.

1 DATA OVERVIEW

To understand the dataset “recipe_reviews.csv”, a command “df.info” in python is used to extract a concise summary of the DataFrame (like Index range i.e number of rows and their index, Column names, Non-null counts, Data types (dtypes), & Memory usage).

Command “df.head” is used to view the first few rows of a DataFrame.

The output of the info command is as below:

```
RangeIndex: 18182 entries, 0 to 18181
Data columns (total 24 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Unnamed: 0      18182 non-null  int64
 1   recipe_number   18182 non-null  int64
 2   recipe_code     18182 non-null  int64
 3   likes_score     18182 non-null  float64
 4   dislike_index   18182 non-null  float64
 5   response_level  18182 non-null  float64
 6   user_index      18182 non-null  float64
 7   ranking_value   18182 non-null  float64
 8   vote_ratio      18182 non-null  float64
 9   score_log       18182 non-null  float64
 10  region          18182 non-null  object
 11  device_type     18182 non-null  object
 12  recipe_name     18182 non-null  object
 13  comment_id      18182 non-null  object
 14  user_id         18182 non-null  object
 15  user_name       18182 non-null  object
 16  user_score      18182 non-null  int64
 17  created_at      18182 non-null  int64
 18  responses       18182 non-null  int64
 19  likes           18182 non-null  int64
 20  dislikes        18182 non-null  int64
 21  ranking_score   18182 non-null  int64
 22  text            18180 non-null  object
 23  stars           18182 non-null  int64
dtypes: float64(7), int64(10), object(7)
memory usage: 3.3+ MB
```

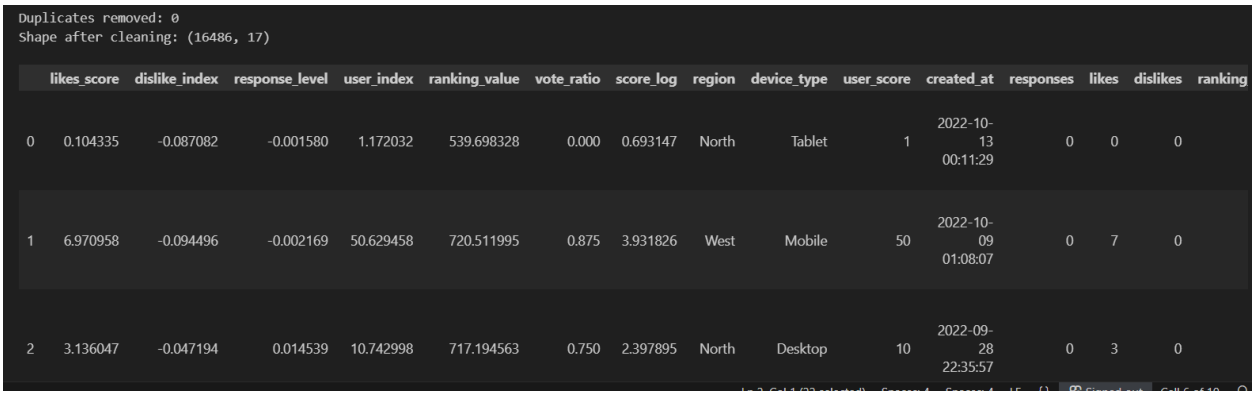
2 DATA CLEANING

In order to clean up the data to analysis, some data cleaning methods were used to make the data accurate and consistent. To resolve missing values, first, the data was identified and substituted with configurable values, including the value of 2 that is mentioned in the description of the data and could be considered as a missing value. Repeated records were eliminated in order to prevent bias and redundancy. Then, conversion of columns to respective data types such as converting timestamp fields into the datatype of datetime to enable temporal analysis was done. Categorical values were standardized to address the differences, including different capitalization

in the labels of regions or devices. Unimportant columns, such as identifiers and uninformative fields were eliminated to concentrate on the characteristics that were important to analysis. Lastly, invalid ratings were considered by deleting the reviews whose rating is 0 since the prediction task must be carried out using valid ratings between 1-5 range. All these measures made the dataset more reliable and prepared it to be the object of robust modeling and predictive analysis.

```
df_raw = df.copy()
# Replace '2' placeholders in text columns with NaN
obj_cols = df_raw.select_dtypes(include='object').columns
df_raw[obj_cols] = df_raw[obj_cols].replace('2', np.nan)
#Drop duplicates
before = len(df_raw)
df_raw = df_raw.drop_duplicates()
print("Duplicates removed:", before - len(df_raw))
#Convert created_at to datetime
if 'created_at' in df_raw.columns:
    try:
        dt = pd.to_datetime(df_raw['created_at'], unit='s', errors='coerce')
        if dt.isna().mean() > 0.9:
            dt = pd.to_datetime(df_raw['created_at'], errors='coerce')
        df_raw['created_at'] = dt
    except Exception as e:
        print("created_at parse skipped:", e)
#Remove 0-star rows
if 'stars' in df_raw.columns and pd.api.types.is_numeric_dtype(df_raw['stars']
    zero_star = (df_raw['stars'] == 0).sum()
    if zero_star:
        print("0-star rows removed:", zero_star)
        df_raw = df_raw[df_raw['stars'] != 0]
#Drop irrelevant/ID columns
cols_to_drop = [
    'Unnamed: 0', 'recipe_number', 'recipe_code', 'recipe_name',
    'comment_id', 'user_id', 'user_name'
]
df = df_raw.drop(columns=[c for c in cols_to_drop if c in df_raw.columns], err
#Final check
print("Shape after cleaning:", df.shape)
display(df.head(3))
```

The snapshot of output is



```
Duplicates removed: 0
Shape after cleaning: (16486, 17)
```

	likes_score	dislike_index	response_level	user_index	ranking_value	vote_ratio	score_log	region	device_type	user_score	created_at	responses	likes	dislikes	ranking
0	0.104335	-0.087082	-0.001580	1.172032	539.698328	0.000	0.693147	North	Tablet	1	2022-10-13 00:11:29	0	0	0	
1	6.970958	-0.094496	-0.002169	50.629458	720.511995	0.875	3.931826	West	Mobile	50	2022-10-09 01:08:07	0	7	0	
2	3.136047	-0.047194	0.014539	10.742998	717.194563	0.750	2.397895	North	Desktop	10	2022-09-28 22:35:57	0	3	0	

3 EXPLORATORY DATA ANALYSIS (EDA)

The filtered dataset contains 16,484 reviews in 100 recipes since 2021 (Feb)-2022 (Oct) with most reviews made by one time users. There is strong imbalance of classes as 5 stars are highly rated (84%), 4 stars (10%), and less than 6 stars (under 10%). Most of the numeric variables (user score, likes/dislikes, ranking score) are very skewed, most of which are close to the zero or baseline, and there are a few outliers. The reputation of region, device and user do not play a significant role in ratings, whereas the community feedback (likes/dislikes) proves to be more predictive: 1-star ratings will have a lot of dislikes, 2-star ratings will have the most likes, and 5-star ratings will be widely accepted.

Key features show strong skew:

- **User Score:** Mostly low (≤1), with few very active users.
- **Likes/Dislikes:** Usually zero, but some reviews receive 100+.
- **Responses:** Rare (~2% of reviews).
- **Ranking Score:** Mostly ~100, with occasional high outliers.

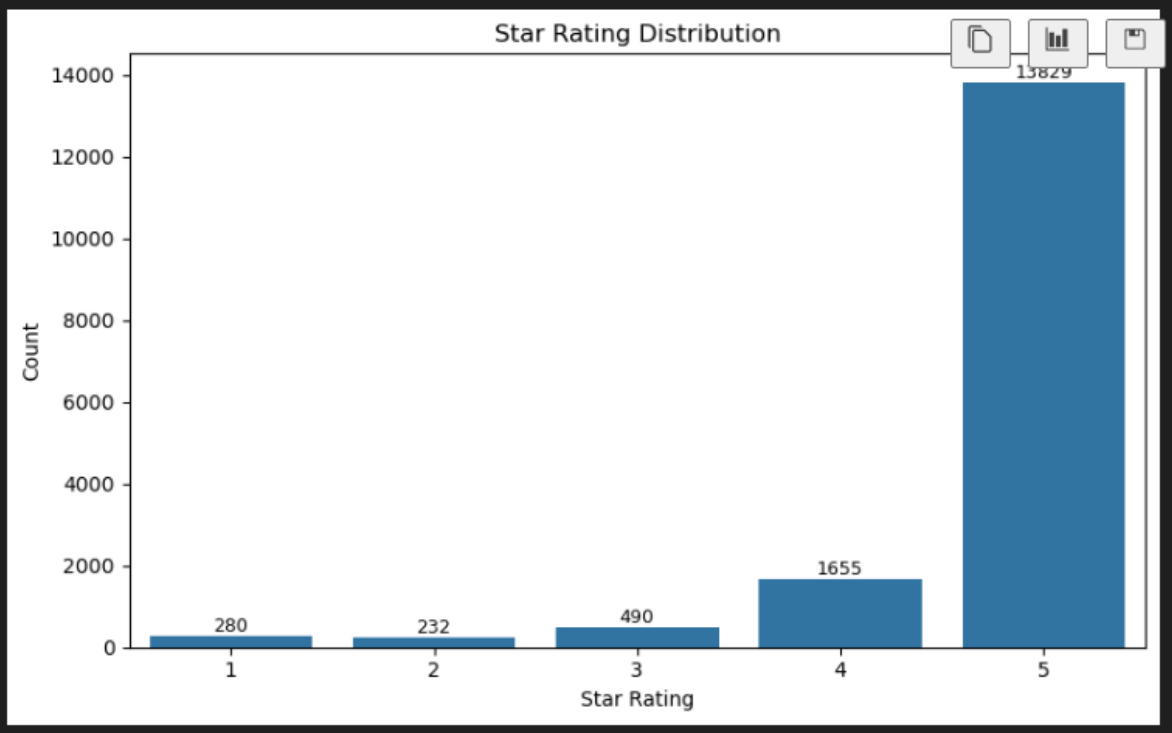
Feature–rating relationships:

- **Region/Device:** No effect on ratings.
- **User Reputation:** Weak link; both new and veteran users mostly give 5 stars.
- **Community Feedback:** Stronger signal—1-star reviews attract many dislikes, while rare 2-star reviews gain the most likes.

```
if 'df' in globals():
    if 'stars' in df.columns:
        rating_counts = df['stars'].value_counts().sort_index()
        print("Star counts:\n", rating_counts)

        plt.figure(figsize=(8,5))
        ax = sns.barplot(x=rating_counts.index.astype(str), y=rating_counts.val
        plt.title("Star Rating Distribution")
        plt.xlabel("Star Rating")
        plt.ylabel("Count")

        # Add value labels
        for i, v in enumerate(rating_counts.values):
            ax.text(i, v, str(v), ha='center', va='bottom', fontsize=9)
        plt.tight_layout()
        plt.show()
    else:
        print("Column `stars` not found.")
else:
    print("Load the data first.")
```

Output:

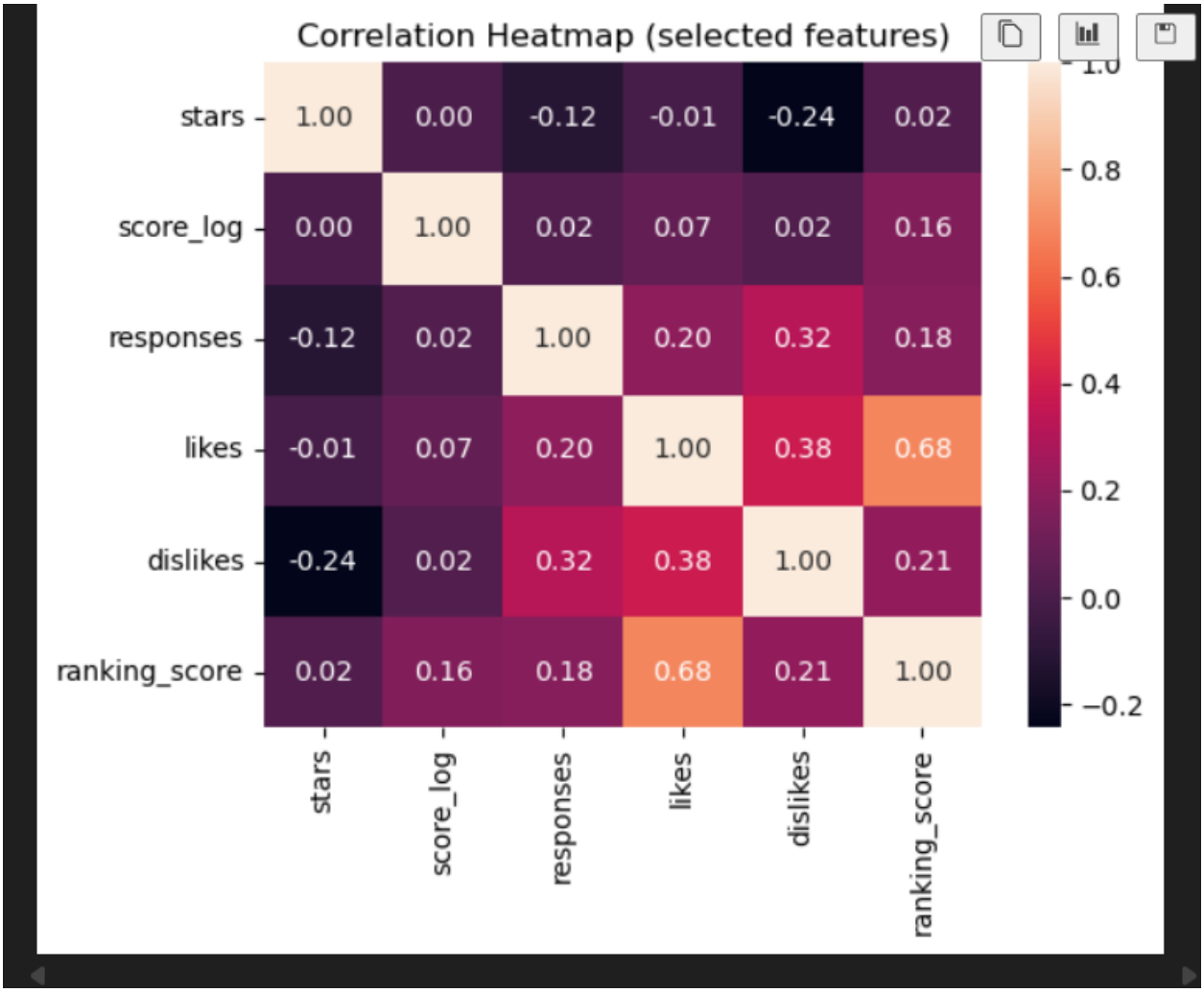
```
Star counts:
 stars
1      280
2      232
3      490
4     1655
5    13829
Name: count, dtype: int64
```



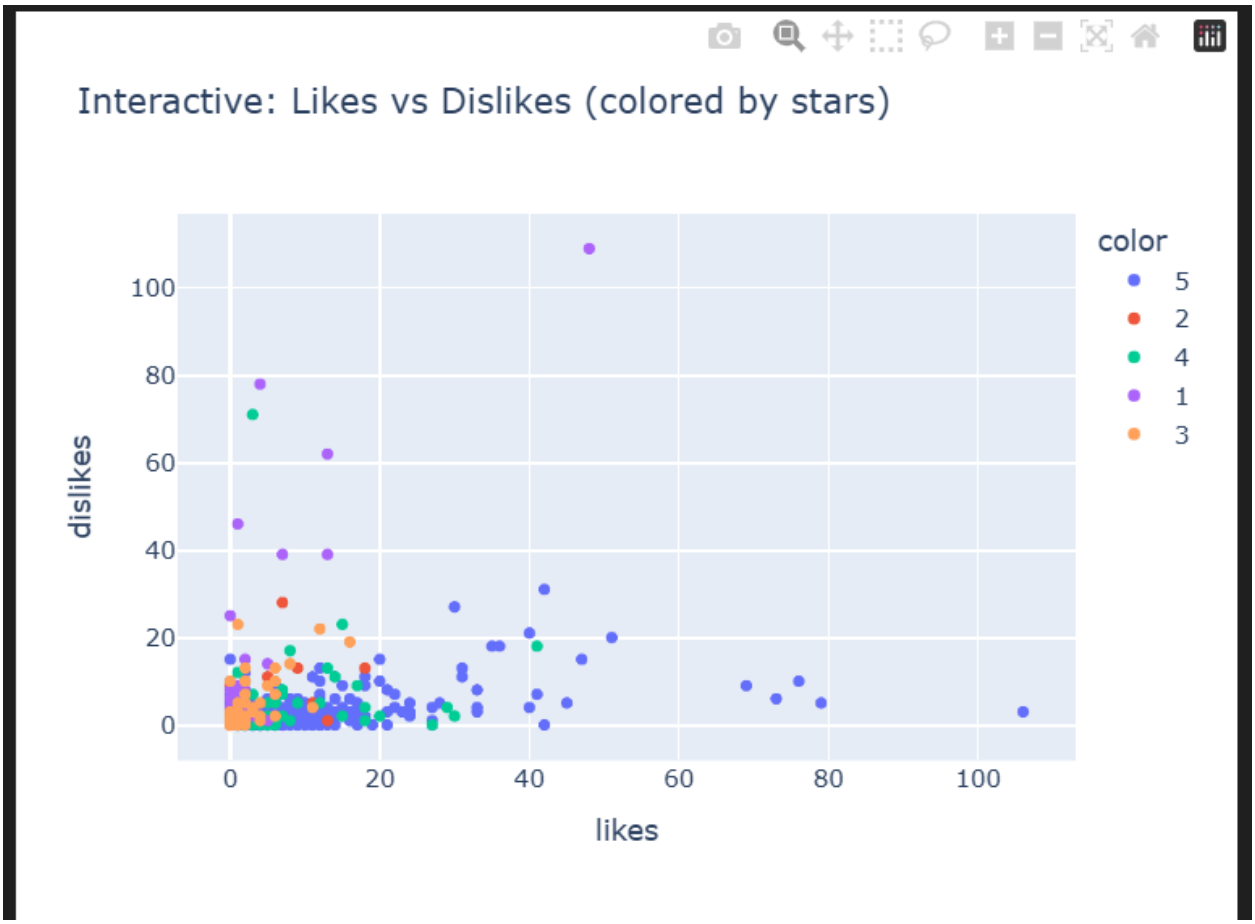
Feature Correlations: To measure some of these correlations, a correlation heatmap of the main numeric features and the star rating is provided below:

Based on the heatmap, I affirm that stars has a moderate negative relationship with dislikes (-0.24) and responses (-0.12). In other words, the lower the star ratings the higher number of dislikes and number of replies (responses) in the discussion. The star rating has little to do with the userscore (scorelog = 0.0) or likes (= -0.01), where a high or a low rating is the one to get likes (depending on context). Interestingly, the ranking_score (internal rank) is also not

correlated with stars (0.017), presumably because it is largely a measure of engagement (a controversial 1-star review could be ranked high with a lot of votes and a 5-star review could be ranking at default).



This scatter plot below shows that most reviews get few likes or dislikes. **5-star reviews** gather likes with almost no dislikes, while **1-star reviews** often receive many dislikes, highlighting clear community disagreement with negative ratings.



4 FEATURE ENGINEERING

I used the feature set to model by removing redundancy and enhancing predictive ability. I eliminated overlapping or strongly connected variables, including likesscore, dislikeindex, and responselevel and used the raw numbers (likes, dislikes, responses). To have user reputation without excessive extreme values, I substituted userscore with its normalized form, scorelog, and retained rankingscore though I eliminated its near-synonym rankingvalue. To maintain the generalisability of the results, I have omitted identifier fields (recipecode, userid, recipename) and put review text aside as sentiment analysis is out of my scope at present. Despite the fact that I looked at such engineered features like text length, temporal splits, and vote ratios, exploratory

checks revealed little incremental value on top of raw likes and dislikes. Lastly, I encoded the nominal variables (region and device-type) using the one-hot method and normalized all numeric variables to similar scales to those models that are sensitive to the magnitude (like logistic regression).

```
if 'df' in globals():
    target = 'stars'
    base_features = []
    # Add columns if they exist
    for c in ['score_log','responses','likes','dislikes','ranking_score','region','device_type']:
        if c in df.columns:
            base_features.append(c)

    if target not in df.columns:
        raise ValueError("Target column `stars` is missing. Please check your CSV.")

    X = df[base_features].copy()
    y = df[target].copy()

    print("Features used:", list(X.columns))
    # Identify numeric/categorical
    numeric_features = [c for c in ['score_log','responses','likes','dislikes','ranking_score'] if c in X.columns]
    categorical_features = [c for c in ['region','device_type'] if c in X.columns]

    display(X.head(3))
```

Output

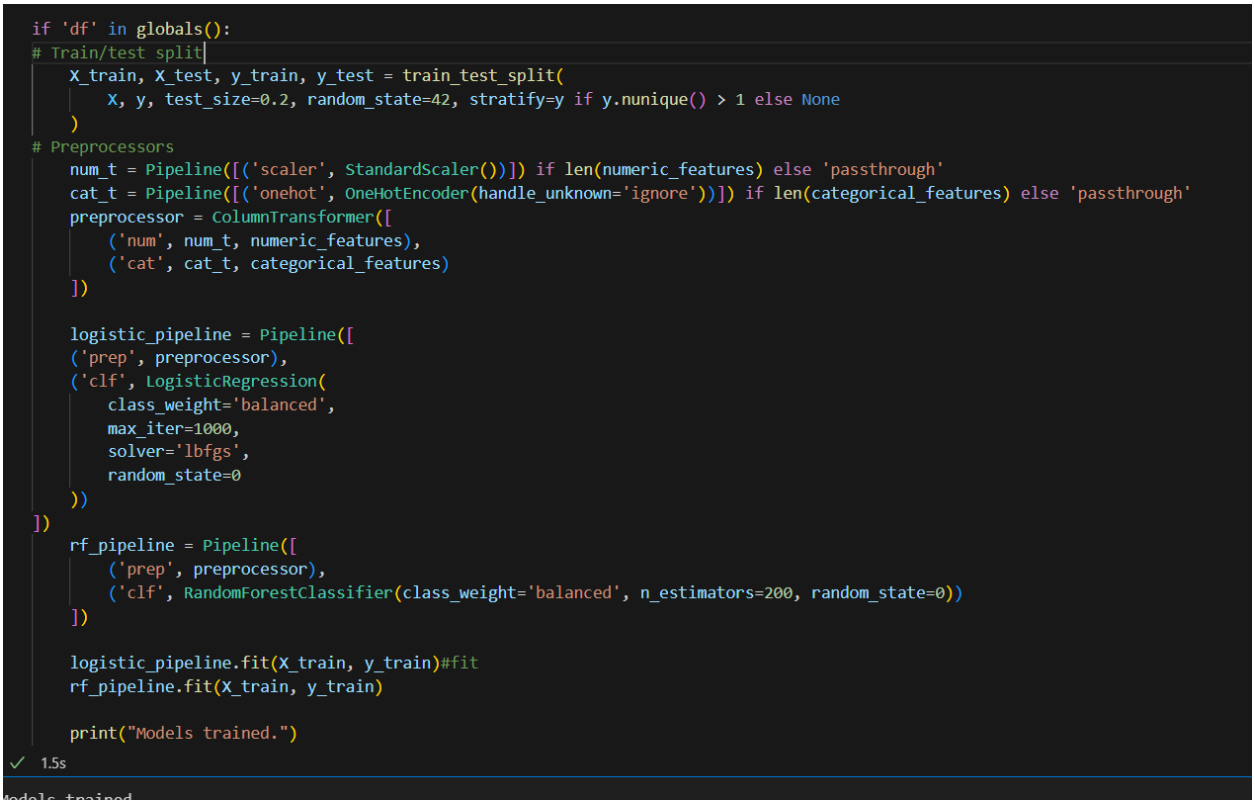
```
Features used: ['score_log', 'responses', 'likes', 'dislikes', 'ranking_score', 'region', 'device_type']
```

	score_log	responses	likes	dislikes	ranking_score	region	device_type
0	0.693147	0	0	0	527	North	Tablet
1	3.931826	0	7	0	724	West	Mobile
2	2.397895	0	3	0	710	North	Desktop

5 MODEL BUILDING

To predict the ratings of the stars on reviews, I constructed and contrasted two classification models. **First**, I took the Logistic Regression as a reference since it is easy to understand, the coefficients show the impact of each feature.

Second, I used a Random Forest, which is a collection of decision trees with non-linearities and feature interactions, and usually stronger predictive accuracy. Stratified sampling was used, and the data were divided into the training (80 percent) and test (20 percent) sets to maintain the classes distribution and make sure that the frequent ratings (1-3 stars) are found in both training and test sets. Since the number of class imbalance (around 84 percent of reviews are 5-star) was high, I resolved to use class weights (balanced mode) so that minority classes were prioritized in the process of training. In this way, both models were able to learn patterns that were outside of the 5-star majority domination and the comparison of their predictive performance was made fairly.

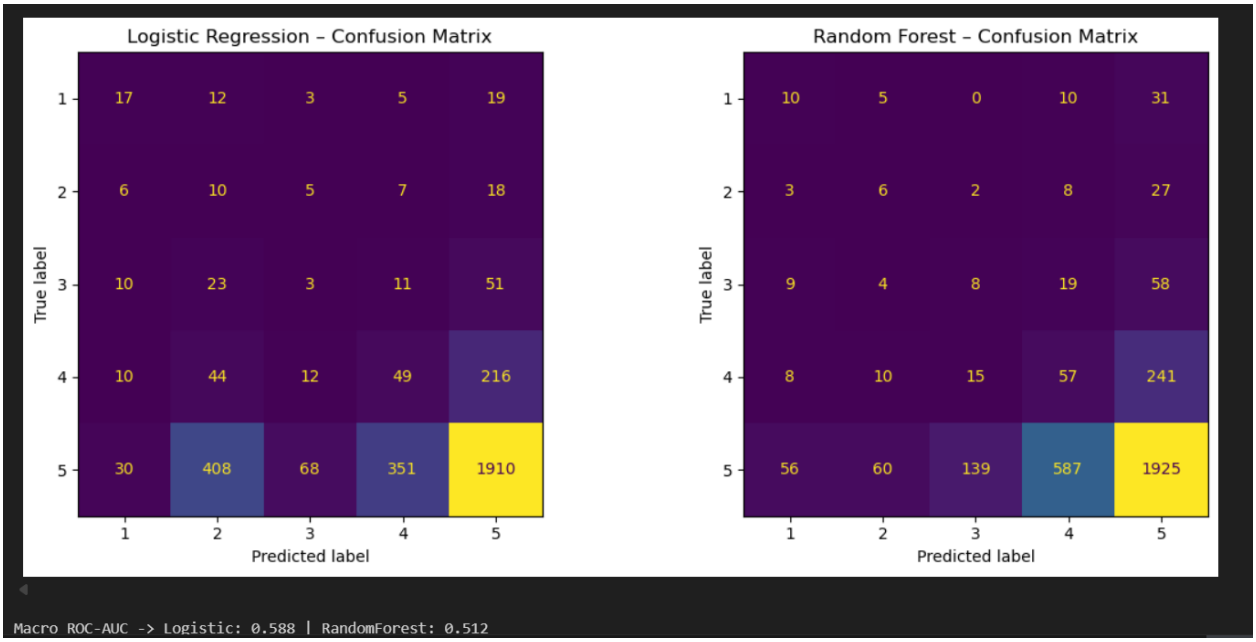


```
if 'df' in globals():
# Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y if y.nunique() > 1 else None
    )
# Preprocessors
    num_t = Pipeline([('scaler', StandardScaler())]) if len(numeric_features) else 'passthrough'
    cat_t = Pipeline([('onehot', OneHotEncoder(handle_unknown='ignore'))]) if len(categorical_features) else 'passthrough'
    preprocessor = ColumnTransformer([
        ('num', num_t, numeric_features),
        ('cat', cat_t, categorical_features)
    ])

    logistic_pipeline = Pipeline([
    ('prep', preprocessor),
    ('clf', LogisticRegression(
        class_weight='balanced',
        max_iter=1000,
        solver='lbfgs',
        random_state=0
    ))
])
    rf_pipeline = Pipeline([
        ('prep', preprocessor),
        ('clf', RandomForestClassifier(class_weight='balanced', n_estimators=200, random_state=0))
    ])

    logistic_pipeline.fit(X_train, y_train)#fit
    rf_pipeline.fit(X_train, y_train)

    print("Models trained.")
✓ 1.5s
```

6 EVALUATIONS

My assessment of the models was based on various metrics to reflect the performance both in general and in classes. The measure of accuracy was the percentage of correct prediction, and the measures of precision and recall evaluated the degree of the accuracy of each individual star rating. F1-score (harmonic mean of the precision and the recall) showed a balanced picture of the performance, particularly in minority classes. I additionally determined the multi-class ROC-AUC to indicate the capacity of the models to differentiate between ratings, and the confusion matrix to determine frequent misclassifications and to have a better understanding of where the models did not perform.

```
=== Logistic Regression ===
              precision    recall  f1-score   support

           1      0.233     0.304     0.264        56
           2      0.020     0.217     0.037        46
           3      0.033     0.031     0.032        98
           4      0.116     0.148     0.130       331
           5      0.863     0.690     0.767      2767

    accuracy                          0.603      3298
   macro avg      0.253     0.278     0.246      3298
weighted avg      0.741     0.603     0.662      3298


=== Random Forest ===
              precision    recall  f1-score   support

           1      0.116     0.179     0.141        56
           2      0.071     0.130     0.092        46
           3      0.049     0.082     0.061        98
           4      0.084     0.172     0.113       331
           5      0.844     0.696     0.763      2767

    accuracy                          0.608      3298
   macro avg      0.233     0.252     0.234      3298
weighted avg      0.721     0.608     0.657      3298
```




```
Macro ROC-AUC -> Logistic: 0.588 | RandomForest: 0.512
```

7 RESULTS SUMMARY:

Both models performed modestly as opposed to the naive baseline. Logistic Regression got about 59% and Random Forest got about 61% as compared to the baseline of always predicting 5- stars as 84%. This decrease indicates that the models with the class balancing tried to take into account the minority classes instead of falling back to 5-stars.

There were evident imbalance effects on the performance at the class level. Both models were good on 5 star reviews (precision of about 85, recall of about 68-70), but poor on 1-3 star reviews (recall of 18, precision of 30). 2 and 3 star reviews were the most difficult to get right, and 4-star ads were moderate (recall of about 13-16, precision of maybe 8-10). 5-star ads were the easiest at 10 percent of the data, with 2 and 3 star adverts showing poor results (recall

The confusion matrices affirmed that the models hardly forecasted classifying minorities as they strongly inclined towards 4 and 5 stars. The Logistic Regression with class weighting slightly over-predicted the 2-stars (resulting into a large number of false positives), whereas the Random Forest was more cautious and did not frequently predict it.

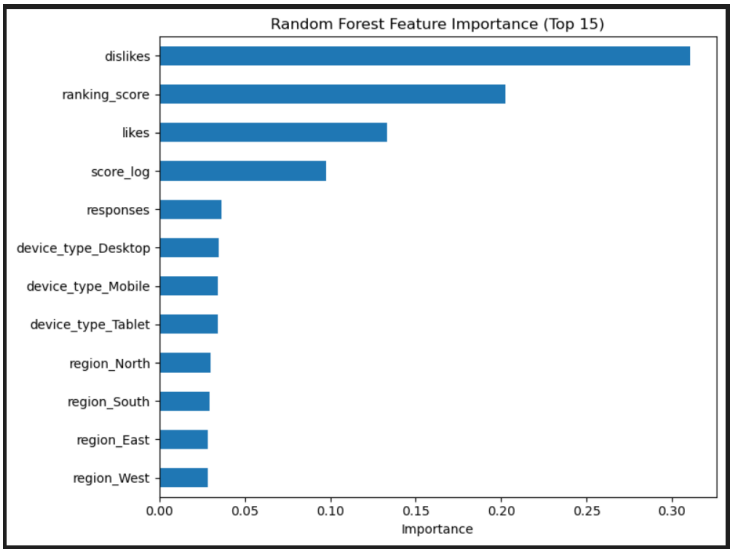
This disparity could be observed in overall metrics: the weighted F1-scores were approximately 0.66 (with the prevalence of 5-star performance), whereas the macro-average F1-scores were significantly lower (around 0.23 0.24). The value of ROC-AUC was also low (0.59 in the case of Logistic and 0.52 in the case of Random Forest), just slightly higher than the chance.

To sum up, both models did not differ decisively, but made slightly more accurate predictions, and Logistic Regression had a better recall of extreme minority classes, respectively. Both are constrained by harsh class imbalance and poor indicators of mid-range ratings. The current deficiencies might be remedied by oversampling techniques (e.g. SMOTE), or by changing the interpretation of the task as an ordinal prediction task to emphasize more rating order.

8 FEATURE IMPORTANCE (RANDOM FOREST):

In summary, **community engagement metrics (dislikes, likes, ranking_score)** are the driving factors in the model. If we recall our earlier correlation, star rating itself didn't correlate with likes but did with dislikes – and indeed, the Random Forest picked up on the dislikes feature most strongly to differentiate low vs high ratings.

```
dislikes                0.310551
ranking_score           0.202702
likes                   0.133335
score_log               0.097762
responses               0.036328
device_type_Desktop     0.034580
device_type_Mobile      0.034357
device_type_Tablet      0.034309
region_North            0.029653
region_South            0.029511
region_East             0.028461
region_West             0.028452
dtype: float64
```



9 KEY INSIGHTS:

The dataset is overwhelmingly positive, with 5-star reviews dominating, making it difficult for models to learn rare cases of dissatisfaction. **Community feedback** emerged as the strongest predictor: reviews with many dislikes were almost always low-star, while 5-star reviews rarely received dislikes. Likes and the ranking_score also contributed, highlighting well-received reviews, whereas **user reputation (score_log)** showed little influence—both new and experienced users gave mostly positive ratings. No differences were observed across regions or devices.

Model performance reflected these constraints. Both Logistic Regression and Random Forest identified 5-star reviews reasonably well but struggled with 1–3 stars, even with class balancing. Logistic Regression slightly improved recall for minority classes, while Random Forest maintained higher accuracy by focusing on the majority class. Overall, the extreme class imbalance limited predictive power.

10 RECOMMENDATIONS:

consider using dislikes to highlight negative or controversial reviews, use more detailed feedback (e.g. half-stars, more detailed prompts), and also focus on helpfulness of reviews (with likes) rather than the number of stars. Modeling could be improved by adding review text via sentiment analysis, reconstructing the problem as ordinal prediction, using oversampling algorithms such as SMOTE, and optimizing ensemble models. Recipe-level or user-level effects should also be considered in the future work in order to have more insightful information.

11 CONCLUSION:

Community engagement signals, especially dislikes, are powerful indicators of low ratings. The main challenge lies not in the modeling technique but in the data's strong positivity bias. Integrating text analysis and addressing imbalance would greatly enhance the ability to detect rare negative reviews and provide a clearer picture of recipe satisfaction.