Business Understanding

Preparing meals is often a challenge due to individual preferences, dietary needs, and ingredient availability. This project aims to develop a Personalized Recipe Recommendation System that uses machine learning and NLP to suggest relevant recipes tailored to each user. The system is designed to enhance convenience, promote healthier eating habits, and reduce food waste. It has potential applications in health tech, food delivery platforms, and smart kitchen systems.

Problem Statement

To develop a Personalized Recipe Recommendation System that leverages machine learning and NLP

Objectives

1. To develop a content-based model using NLP to recommend recipes based on ingredients and instructions.
2. To build a collaborative filtering model using user ratings and interactions.
3. To combine both approaches into a hybrid recommendation system.
4. To evaluate model performance

In [1]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
pip install isodate
```

```
Requirement already satisfied: isodate in /usr/local/lib/python3.11/dist-packages
(0.7.2)
```

In [3]:
```
%load_ext tensorboard
```

In [4]:
```
import pandas as pd
import numpy as np
import ast
import matplotlib.pyplot as plt
import seaborn as sns
from isodate import parse_duration
import warnings
warnings.filterwarnings("ignore")
```

In [5]:
```
file_path = "recipes.parquet"

df_recipes = kagglehub.load_dataset(
    KaggleDatasetAdapter.PANDAS,
    "irkaal/foodcom-recipes-and-reviews",
    file_path,
)
```

In [6]:
```
file_path2 = "reviews.parquet"

df_reviews = kagglehub.load_dataset(
    KaggleDatasetAdapter.PANDAS,
    "irkaal/foodcom-recipes-and-reviews",
    file_path2,
)
```

Data Understanding

In [7]:
```
print("Recipes:", df_recipes.shape)
print("Reviews:", df_reviews.shape)
```

```
Recipes: (522517, 28)
Reviews: (1401982, 8)
```

In [8]:
```
df_recipes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 522517 entries, 0 to 522516
Data columns (total 28 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   RecipeId                    522517 non-null  float64
 1   Name                        522517 non-null  object
 2   AuthorId                    522517 non-null  int32
 3   AuthorName                  522517 non-null  object
 4   CookTime                    439972 non-null  object
 5   PrepTime                    522517 non-null  object
 6   TotalTime                   522517 non-null  object
 7   DatePublished               522517 non-null  datetime64[us, UTC]
 8   Description                 522512 non-null  object
 9   Images                      522516 non-null  object
 10  RecipeCategory              521766 non-null  object
 11  Keywords                    522517 non-null  object
 12  RecipeIngredientQuantities  522517 non-null  object
 13  RecipeIngredientParts       522517 non-null  object
 14  AggregatedRating            269294 non-null  float64
 15  ReviewCount                 275028 non-null  float64
 16  Calories                    522517 non-null  float64
 17  FatContent                  522517 non-null  float64
 18  SaturatedFatContent         522517 non-null  float64
 19  CholesterolContent          522517 non-null  float64
 20  SodiumContent               522517 non-null  float64
 21  CarbohydrateContent         522517 non-null  float64
 22  FiberContent                522517 non-null  float64
 23  SugarContent                522517 non-null  float64
 24  ProteinContent              522517 non-null  float64
 25  RecipeServings              339606 non-null  float64
 26  RecipeYield                 174446 non-null  object
 27  RecipeInstructions          522517 non-null  object
dtypes: datetime64[us, UTC](1), float64(13), int32(1), object(13)
memory usage: 109.6+ MB
```

In [9]:
```
df_reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1401982 entries, 0 to 1401981
Data columns (total 8 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   ReviewId       1401982 non-null  int32
 1   RecipeId       1401982 non-null  int32
 2   AuthorId       1401982 non-null  int32
 3   AuthorName     1401982 non-null  object
 4   Rating         1401982 non-null  int32
 5   Review         1401982 non-null  object
 6   DateSubmitted  1401982 non-null  datetime64[us, UTC]
 7   DateModified   1401982 non-null  datetime64[us, UTC]
dtypes: datetime64[us, UTC](2), int32(4), object(2)
memory usage: 64.2+ MB
```

In [10]:
```
df_recipes.head()
```

Out[10]:

	RecipeId	Name	AuthorId	AuthorName	CookTime	PrepTime	TotalTime	DatePub
0	38.0	Low-Fat Berry Blue Frozen Dessert	1533	Dancer	PT24H	PT45M	PT24H45M	1999 21:46:00
1	39.0	Biryani	1567	elly9812	PT25M	PT4H	PT4H25M	1999 13:12:00
2	40.0	Best Lemonade	1566	Stephen Little	PT5M	PT30M	PT35M	1999 19:52:00
3	41.0	Carina's Tofu-Vegetable Kebabs	1586	Cyclopz	PT20M	PT24H	PT24H20M	1999 14:54:00
4	42.0	Cabbage Soup	1538	Duckie067	PT30M	PT20M	PT50M	1999 06:19:00

5 rows × 28 columns

In [11]:

```
df_reviews.head()
```

Out[11]:

	ReviewId	RecipeId	AuthorId	AuthorName	Rating	Review	DateSubmitted	DateM
0	2	992	2008	gayg msft	5	better than any you can get at a restaurant!	2000-01-25 21:44:00+00:00	200(21:44:0(
1	7	4384	1634	Bill Hilbrich	4	I cut back on the mayo, and made up the differ...	2001-10-17 16:49:59+00:00	200 16:49:5(
2	9	4523	2046	Gay Gilmore ckpt	2	i think i did something wrong because i could ...	2000-02-25 09:00:00+00:00	200(09:00:0(
3	13	7435	1773	Malarkey Test	5	easily the best i have ever had. juicy flavor...	2000-03-13 21:15:00+00:00	200(21:15:0(
4	14	44	2085	Tony Small	5	An excellent dish.	2000-03-28 12:51:00+00:00	200(12:51:0(

In [12]:

```
df_recipes.isnull().sum()
```

Out[12]:

	0
RecipeId	0
Name	0
AuthorId	0
AuthorName	0
CookTime	82545
PrepTime	0
TotalTime	0
DatePublished	0
Description	5
Images	1
RecipeCategory	751
Keywords	0
RecipeIngredientQuantities	0
RecipeIngredientParts	0
AggregatedRating	253223
ReviewCount	247489
Calories	0
FatContent	0
SaturatedFatContent	0
CholesterolContent	0
SodiumContent	0
CarbohydrateContent	0
FiberContent	0
SugarContent	0
ProteinContent	0
RecipeServings	182911
RecipeYield	348071
RecipeInstructions	0

dtype: int64

In [13]:
```
df_reviews.isnull().sum()
```

Out[13]:

	0
ReviewId	0
RecipeId	0
AuthorId	0
AuthorName	0
Rating	0
Review	0
DateSubmitted	0
DateModified	0

dtype: int64

Data Cleaning

In [14]:
```
#Handling Missing Values
df_recipes['AggregatedRating'] = df_recipes['AggregatedRating'].fillna(0)
df_recipes['ReviewCount'] = df_recipes['ReviewCount'].fillna(0)
df_recipes['RecipeServings'] = df_recipes['RecipeServings'].fillna(df_recipes['Reci
df_recipes['RecipeCategory'] = df_recipes['RecipeCategory'].fillna("Unknown").str.l

df_reviews.dropna(subset=['Review'], inplace=True)
```

In [15]:
```
# Converting the time to minutes
def safe_parse_minutes(x):
    if pd.isnull(x) or not isinstance(x, str) or not x.startswith('P'):
        return 0
    try:
        return parse_duration(x).total_seconds() / 60
    except:
        return 0

df_recipes['CookTimeMinutes'] = df_recipes['CookTime'].apply(safe_parse_minutes)
df_recipes['PrepTimeMinutes'] = df_recipes['PrepTime'].apply(safe_parse_minutes)
df_recipes['TotalTimeMinutes'] = df_recipes['TotalTime'].apply(safe_parse_minutes)
```

In [16]:
```
# Filling missing time with 0
df_recipes[['CookTimeMinutes', 'PrepTimeMinutes', 'TotalTimeMinutes']] = df_recipes
    ['CookTimeMinutes', 'PrepTimeMinutes', 'TotalTimeMinutes']
].fillna(0)
```

In [17]:
```
# Drop rows where total time is less than 0
df_recipes = df_recipes[df_recipes['TotalTimeMinutes'] > 0]
```

In [18]:
```
# Convert numpy arrays to regular lists
df_recipes['Ingredients'] = df_recipes['RecipeIngredientParts'].apply(lambda x: x.t
df_recipes['Quantities'] = df_recipes['RecipeIngredientQuantities'].apply(lambda x:
```

In [19]:
```
#Convert Text to Lowercase & Clean
for text_col in ['Name', 'Description', 'RecipeInstructions','Keywords']:
    df_recipes[text_col] = df_recipes[text_col].astype(str).str.lower().str.replace
```

In [20]:
```
#Tokenize Keywords into List Format
df_recipes['KeywordList'] = df_recipes['Keywords'].apply(lambda x: x.split())
```

In [21]:
```
df_reviews['Rating'] = df_reviews['Rating'].astype(float)
```

In [22]:
```
# Drop duplicate recipes and reviews
df_recipes.drop_duplicates(subset=['RecipeId'], inplace=True)
df_reviews.drop_duplicates(subset=['ReviewId'], inplace=True)
```

In [23]:
```
#Drop Recipes with Few reviews
MIN_REVIEWS = 5
popular_recipes = df_reviews['RecipeId'].value_counts()
popular_recipes = popular_recipes[popular_recipes >= MIN_REVIEWS].index
df_recipes = df_recipes[df_recipes['RecipeId'].isin(popular_recipes)]
df_reviews = df_reviews[df_reviews['RecipeId'].isin(popular_recipes)]
```

In [24]:
```
# Drop unnecesary cols
drop_cols = ['AuthorName', 'TotalTime', 'PrepTime','CookTime','RecipeIngredientPart
df_recipes.drop(columns=drop_cols, inplace=True, errors='ignore')

drop_cols2 = ['AuthorName']
df_reviews.drop(columns=drop_cols2, inplace=True, errors='ignore')
```

In [25]:
```
recipes_clean=df_recipes
reviews_clean=df_reviews
```

In [26]:
```
missing_recipe_ids = reviews_clean[~reviews_clean['RecipeId'].isin(recipes_clean['R
print(f"Number of reviews with RecipeId not in recipes: {len(missing_recipe_ids)}")
```

```
Number of reviews with RecipeId not in recipes: 4079
```

In [27]:
```
missing_author_ids = reviews_clean[~reviews_clean['AuthorId'].isin(recipes_clean['A
print(f"Number of reviews with AuthorId not in recipes: {len(missing_author_ids)}")
```

```
Number of reviews with AuthorId not in recipes: 532306
```

In [28]:
```
# Create a set of valid (RecipeId, AuthorId) pairs from the recipes dataset
valid_pairs = set(zip(recipes_clean['RecipeId'], recipes_clean['AuthorId']))

# Check which rows in reviews don't have a matching pair
invalid_pairs = reviews_clean[~reviews_clean.apply(lambda row: (row['RecipeId'], ro

print(f"Number of reviews with unmatched RecipeId & AuthorId pairs: {len(invalid_pa
```

```
Number of reviews with unmatched RecipeId & AuthorId pairs: 1028061
```

In [29]:
```
# Keep only reviews with RecipeIds that exist in recipes
valid_reviews = reviews_clean[reviews_clean['RecipeId'].isin(recipes_clean['RecipeI

print(f"Remaining reviews after filtering: {len(valid_reviews)}")
```

```
Remaining reviews after filtering: 1024428
```

In [30]:
```
# Merge on RecipeId
merged_df = pd.merge(
    valid_reviews,
    recipes_clean,
    on='RecipeId',
    how='inner',
    suffixes=('_review', '_recipe')
)

print(f"Merged dataset shape: {merged_df.shape}")
print(merged_df[['RecipeId', 'AuthorId_review', 'AuthorId_recipe']].head())
```

```
Merged dataset shape: (1024428, 32)
   RecipeId  AuthorId_review  AuthorId_recipe
0       992             2008             1545
1      4523             2046             1932
2      7435             1773             1986
3        44             2085             1596
4     13307             2046            20914
```

In [31]:
```
# drop AuthorId_review - we are more interested in the authors of the recipes
drop_cols3 = ['AuthorId_review']
merged_df.drop(columns=drop_cols3, inplace=True, errors='ignore')
```

In [32]:
```
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1024428 entries, 0 to 1024427
Data columns (total 31 columns):
 #   Column              Non-Null Count    Dtype
---  ------              --------------    -----
 0   ReviewId            1024428 non-null  int32
 1   RecipeId            1024428 non-null  int32
 2   Rating              1024428 non-null  float64
 3   Review              1024428 non-null  object
 4   DateSubmitted       1024428 non-null  datetime64[us, UTC]
 5   DateModified        1024428 non-null  datetime64[us, UTC]
 6   Name                1024428 non-null  object
 7   AuthorId_recipe     1024428 non-null  int32
 8   DatePublished       1024428 non-null  datetime64[us, UTC]
 9   Description         1024428 non-null  object
 10  Images              1024428 non-null  object
 11  RecipeCategory      1024428 non-null  object
 12  AggregatedRating    1024428 non-null  float64
 13  ReviewCount         1024428 non-null  float64
 14  Calories            1024428 non-null  float64
 15  FatContent          1024428 non-null  float64
 16  SaturatedFatContent 1024428 non-null  float64
 17  CholesterolContent  1024428 non-null  float64
 18  SodiumContent       1024428 non-null  float64
 19  CarbohydrateContent 1024428 non-null  float64
 20  FiberContent        1024428 non-null  float64
 21  SugarContent        1024428 non-null  float64
 22  ProteinContent      1024428 non-null  float64
 23  RecipeServings      1024428 non-null  float64
 24  RecipeInstructions  1024428 non-null  object
 25  CookTimeMinutes     1024428 non-null  float64
 26  PrepTimeMinutes     1024428 non-null  float64
 27  TotalTimeMinutes    1024428 non-null  float64
 28  Ingredients         1024428 non-null  object
 29  Quantities          1024428 non-null  object
 30  KeywordList         1024428 non-null  object
dtypes: datetime64[us, UTC](3), float64(16), int32(3), object(9)
memory usage: 230.6+ MB
```

In [33]:

```
from sklearn.preprocessing import LabelEncoder

recipe_encoder = LabelEncoder()
author_encoder = LabelEncoder()

merged_df['RecipeId_encoded'] = recipe_encoder.fit_transform(merged_df['RecipeId'])
```

In [34]:

```
#Normalize Nutritional Features
from sklearn.preprocessing import MinMaxScaler

nutritional_cols = [
    'Calories', 'FatContent', 'SaturatedFatContent', 'CholesterolContent',
    'SodiumContent', 'CarbohydrateContent', 'FiberContent',
    'SugarContent', 'ProteinContent'
]
```

```
scaler = MinMaxScaler()
merged_df[nutritional_cols] = scaler.fit_transform(merged_df[nutritional_cols])
```

Feature Engineering

We are going to categorize recipes that share a theme into recipe categories

In [35]:
```
#print unique name and unique RecipeCategory

unique_categories = merged_df['RecipeCategory'].unique()

print("\nUnique Recipe Categories:")
print(unique_categories)
```

```
Unique Recipe Categories:
['vegetable' 'chicken breast' 'meat' 'chicken' 'dessert' 'lamb/sheep'
 'steak' 'pork' 'pie' 'tuna' 'sauces' 'quick breads' 'drop cookies'
 'lunch/snacks' 'winter' 'breads' 'whole chicken' 'bar cookie'
 'high protein' 'candy' 'beans' 'cheesecake' 'meatloaf' 'corn' 'stew'
 'potato' 'german' 'breakfast' 'white rice' '< 60 mins' 'cheese' 'crab'
 'punch beverage' 'cauliflower' 'european' 'ham' 'onions' 'clear soup'
 'beverages' 'lentil' 'low protein' 'savory pies' 'chowders' 'pineapple'
 'free of...' 'yeast breads' '< 15 mins' 'spaghetti' 'poultry' 'spreads'
 'gelatin' 'berries' 'one dish meal' 'fruit' 'mussels' 'frozen desserts'
 'roast beef' 'smoothies' 'long grain rice' 'salad dressings' 'oven'
 'weeknight' 'rice' 'manicotti' 'shakes' 'low cholesterol' 'scones'
 'halibut' 'greens' 'veal' 'potluck' 'vegan' 'very low carbs' 'plums'
 'tex mex' 'collard greens' 'brown rice' 'cajun' 'black beans' 'jellies'
 '< 30 mins' 'mahi mahi' 'penne' 'apple' 'lactose free'
 'southwestern u.s.' 'short grain rice' 'healthy' 'refrigerator' 'summer'
 'whole turkey' 'orange roughy' 'gumbo' 'broil/grill' 'mexican'
 'chicken thigh & leg' 'grains' 'asian' 'swedish' 'curries' 'bass'
 'pasta shells' 'african' 'wild game' 'strawberry' 'yam/sweet potato'
 'peppers' 'greek' 'high in...' 'chutneys' 'bath/beauty' 'canadian'
 'coconut' 'beef organ meats' 'kid friendly' 'thanksgiving' 'lobster'
 'oranges' 'southwest asia (middle east)' 'japanese' 'spinach' 'catfish'
 'tilapia' 'squid' 'stocks' 'australian' 'household cleaner' 'deer'
 'raspberries' 'whole duck' '< 4 hours' 'trout' 'christmas' 'kosher'
 'spicy' 'hungarian' 'crawfish' 'spanish' 'sourdough breads' 'soy/tofu'
 'medium grain rice' 'hawaiian' 'spring' 'caribbean' 'chinese'
 'duck breasts' 'korean' 'high fiber' 'pears' 'thai' 'sweet' 'vietnamese'
 'turkey breasts' 'szechuan' 'polish' 'summer dip' 'pot pie' 'savory'
 'homeopathy/remedies' 'tempeh' 'roast' 'moroccan' 'egyptian' 'perch'
 'polynesian' 'cherries' 'lemon' 'citrus' 'creole' 'russian' 'lime' 'duck'
 'chard' 'tarts' 'toddler friendly' 'tropical fruits' 'melons' 'brunch'
 'octopus' 'lebanese' 'turkish' 'microwave' 'mango' 'brazilian'
 "st. patrick's day" 'rabbit' 'ethiopian' 'venezuelan' 'no cook' 'belgian'
 'whitefish' 'pheasant' 'native american' 'nuts' 'for large groups'
 'no shell fish' 'unknown' 'chicken livers' 'elk' 'costa rican' 'egg free'
 'easy' 'scandinavian' 'dairy free foods' 'portuguese' 'halloween'
 'icelandic' 'stove top' 'south african' 'dutch' 'peanut butter' 'papaya'
 'stir fry' 'georgian' 'danish' 'beginner cook' 'pakistani' 'swiss'
 'finnish' 'norwegian' 'chocolate chip cookies' 'welsh' 'new zealand'
 'mixer' 'pressure cooker' 'scottish' 'south american' 'moose'
 'bread machine' 'chilean' 'cuban' 'ecuadorean' 'camping'
 'small appliance' 'czech' 'cantonese' 'quail' 'palestinian' 'canning'
 'meatballs' 'goose' 'nepalese' 'bread pudding' 'kiwifruit' 'bear'
 'pennsylvania dutch' 'indonesian' 'oatmeal' 'puerto rican'
 'mashed potatoes' 'indian' 'macaroni and cheese' 'inexpensive' 'avocado'
 'artichoke' 'pumpkin' 'ice cream' 'beef liver' 'cambodian']
```

In [36]:

```python
# Define the category mapping (as shown above)
category_mapping = {
    'chicken': ['chicken', 'chicken breast', 'chicken thigh & leg', 'whole chicken'
    'beef': ['steak', 'roast beef', 'beef organ meats', 'wild game', 'goose', 'meat
    'pork': ['pork', 'ham', 'sausage', 'bacon'],
    'lamb': ['lamb/sheep'],
    'fish': ['tuna', 'halibut', 'tilapia', 'trout', 'bass', 'perch', 'salmon', 'cat
    'vegetarian': ['vegetable', 'vegan', 'tofu', 'lentil', 'beans', 'cauliflower',
    'desserts': ['dessert', 'candy', 'cheesecake', 'gelatin', 'frozen desserts', 's
```

```
    'baked goods': ['bread', 'breads', 'quick breads', 'yeast breads', 'sourdough b
    'pasta': ['spaghetti', 'macaroni and cheese', 'penne', 'manicotti', 'pasta shel
    'rice': ['white rice', 'brown rice', 'short grain rice', 'long grain rice', 'wi
    'potatoes': ['potato', 'mashed potatoes', 'sweet potato', 'yam/sweet potato'],
    'salads': ['salad', 'salad dressings', 'coleslaw', 'potato salad', 'fruit salad
    'soups': ['soup', 'clear soup', 'chowders', 'gumbo', 'stew', 'lentil soup', 'ch
    'grains': ['grains', 'quinoa', 'oats', 'barley', 'couscous', 'farro', 'bulgur',
    'meatloaf': ['meatloaf'],
    'sauces': ['sauces', 'gravy', 'chutneys', 'barbecue sauce', 'pasta sauce'],
    'sides': ['corn', 'side dish', 'potluck', 'side salad'],
    'drinks': ['beverages', 'smoothies', 'punch beverage', 'shakes', 'milkshakes',
    'breakfast': ['breakfast', 'brunch', 'oatmeal', 'pancakes', 'eggs', 'waffles',
    'high protein': ['high protein', 'protein shakes', 'protein bars'],
    'low protein': ['low protein'],
    'healthy': ['healthy', 'low fat', 'low carb', 'low sugar', 'low cholesterol', '
    'low carb': ['very low carbs', 'low carb'],
    'low cholesterol': ['low cholesterol'],
    'high fiber': ['high fiber'],
    'gluten free': ['gluten free'],
    'dairy free': ['dairy free foods', 'lactose free'],
    'sugar free': ['sugar free', 'low sugar'],
    'holiday': ['thanksgiving', 'christmas', "st. patrick's day", 'halloween', 'eas
    'international': ['mexican', 'italian', 'chinese', 'indian', 'japanese', 'greek
    'mexican': ['mexican', 'tex mex', 'southwestern u.s.'],
    'italian': ['italian', 'sicilian', 'neapolitan'],
    'asian': ['asian', 'chinese', 'japanese', 'korean', 'vietnamese', 'thai'],
    'american': ['american', 'southern', 'new england', 'midwestern', 'californian'
    'bbq': ['bbq', 'barbecue', 'grilled', 'broil/grill'],
    'cajun': ['cajun', 'creole', 'gumbo'],
    'seafood': ['fish', 'seafood', 'shrimp', 'lobster', 'mussels'],
    'vegan': ['vegan'],
    'low fat': ['low fat'],
    'poultry': ['poultry', 'duck', 'turkey', 'chicken', 'whole turkey', 'duck breas
    'holiday': ['christmas', 'thanksgiving', 'halloween', "st. patrick's day"],
    'fruit': ['fruit', 'berries', 'apples', 'bananas', 'pears', 'pineapple', 'grape
    'potluck': ['potluck', 'party', 'for large groups'],
    'sweets': ['sweets', 'candies', 'chocolates', 'cookies', 'brownies', 'cakes'],
    'comfort food': ['comfort food', 'mac and cheese', 'meatloaf', 'mashed potatoes
    'quick meals': ['quick meals', 'quick', '< 30 mins', '< 15 mins', '< 60 mins'],
    'free of allergens': ['free of...', 'egg free', 'dairy free', 'gluten free', 'l
    'canned': ['canning', 'preserving'],
    'meals in a dish': ['one dish meal'],
    'low calorie': ['low calorie', 'low fat', 'low carb'],
    'sweets and snacks': ['sweets', 'candies', 'cookies', 'chips', 'snacks'],
    'easy': ['easy', 'beginner cook', 'beginner'],
    'other': ['unknown', 'bath/beauty', 'household cleaner', 'microwave', 'mixer',
}

# Flatten the category mapping into a lookup dictionary
lookup = {kw.lower(): group for group, keywords in category_mapping.items() for kw

# Function to map a single category to a broader group
def map_category(category):
    if pd.isna(category):
        return 'others'  # Handling missing values
    category = category.lower()  # Convert category to lowercase
```

```
    for keyword, group in lookup.items():
        if keyword in category:  # Check if keyword is part of category
            return group  # Return the corresponding group if found
    return 'others'  # Default category if no match found

# Apply the map_category function to the RecipeCategory column and add it as a new
merged_df['MappedCategory'] = merged_df['RecipeCategory'].apply(map_category)

# Preview the new column to make sure it's working
#print(merged_df[['RecipeCategory', 'MappedCategory']].head())
merged_df.head()
```

Out[36]:

	ReviewId	RecipeId	Rating	Review	DateSubmitted	DateModified	Name	A
0	2	992	5.0	better than any you can get at a restaurant!	2000-01-25 21:44:00+00:00	2000-01-25 21:44:00+00:00	jalapeno pepper poppers	
1	9	4523	2.0	i think i did something wrong because i could ...	2000-02-25 09:00:00+00:00	2000-02-25 09:00:00+00:00	chinese imperial palace general tsos chicken	
2	13	7435	5.0	easily the best i have ever had. juicy flavor...	2000-03-13 21:15:00+00:00	2000-03-13 21:15:00+00:00	kevins best corned beef	
3	14	44	5.0	An excellent dish.	2000-03-28 12:51:00+00:00	2000-03-28 12:51:00+00:00	warm chicken a la king	
4	19	13307	5.0	chewy goodness, not crispy at all. i even thre...	2000-05-21 16:59:00+00:00	2000-05-21 16:59:00+00:00	neimanmarcus chocolate chip cookies recipe	

5 rows × 33 columns

EDA

We will do visualizations across themes to get understanding of how the data looks.

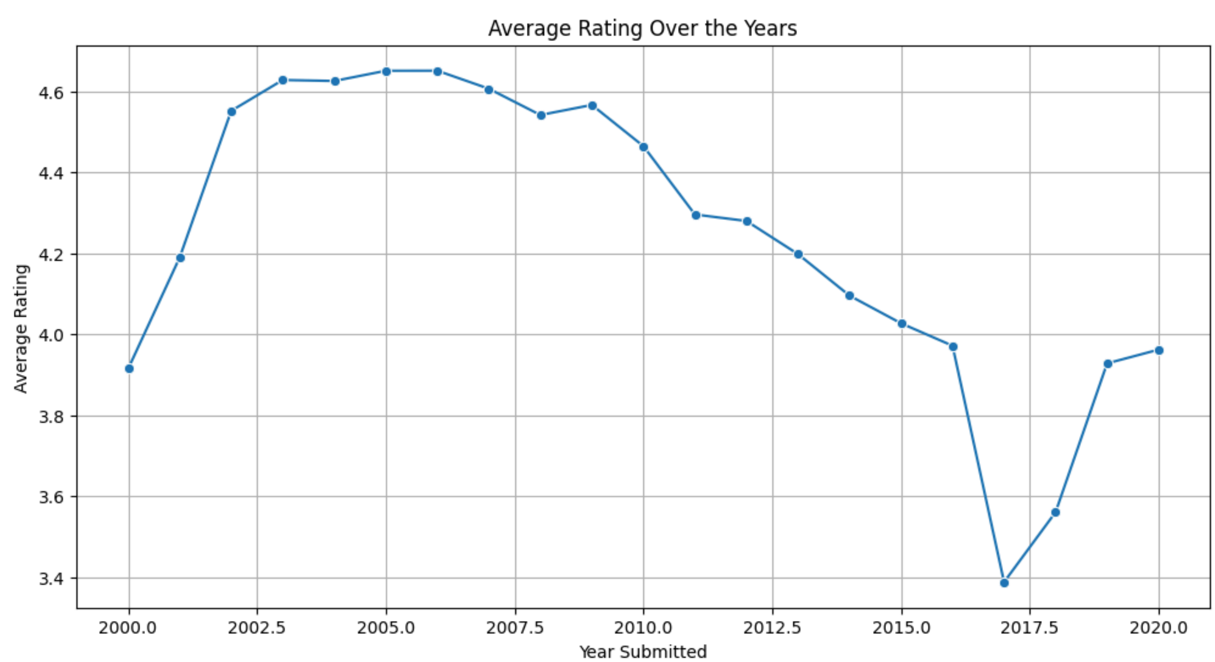
In [37]:
```
#Check unique years under DateSubmitted column
unique_years = merged_df['DateSubmitted'].dt.year.unique()
print(unique_years)
```

```
[2000 2001 2002 2005 2003 2006 2004 2007 2020 2016 2018 2017 2008 2019
 2009 2010 2011 2012 2013 2014 2015]
```

2.We create a Line Plot to check how rating was done over the years.

The results indicate that recipes were highest rated by users in the year 2006. and least rated in the year 2017

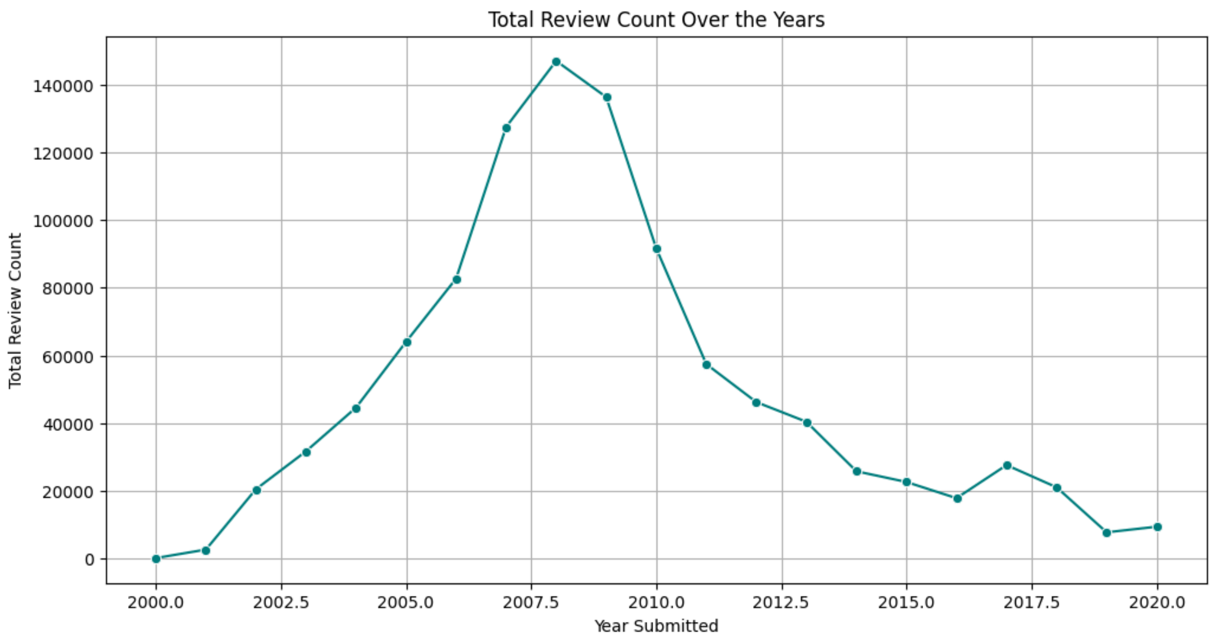
In [38]:
```
# Ensure 'DateSubmitted' is in datetime format
merged_df['DateSubmitted'] = pd.to_datetime(merged_df['DateSubmitted'])

# Extract the year
merged_df['SubmissionYear'] = merged_df['DateSubmitted'].dt.year

# Calculate the average rating per year
average_rating_by_year = merged_df.groupby('SubmissionYear')['Rating'].mean().reset

# Sort by year
average_rating_by_year = average_rating_by_year.sort_values(by='SubmissionYear')

# Create the line plot
plt.figure(figsize=(12, 6))
sns.lineplot(data=average_rating_by_year, x='SubmissionYear', y='Rating', palette='
plt.title('Average Rating Over the Years')
plt.xlabel('Year Submitted')
plt.ylabel('Average Rating')
plt.grid(True)
plt.show()
```



We do a Line Plot to checktotal count of reviews each year.

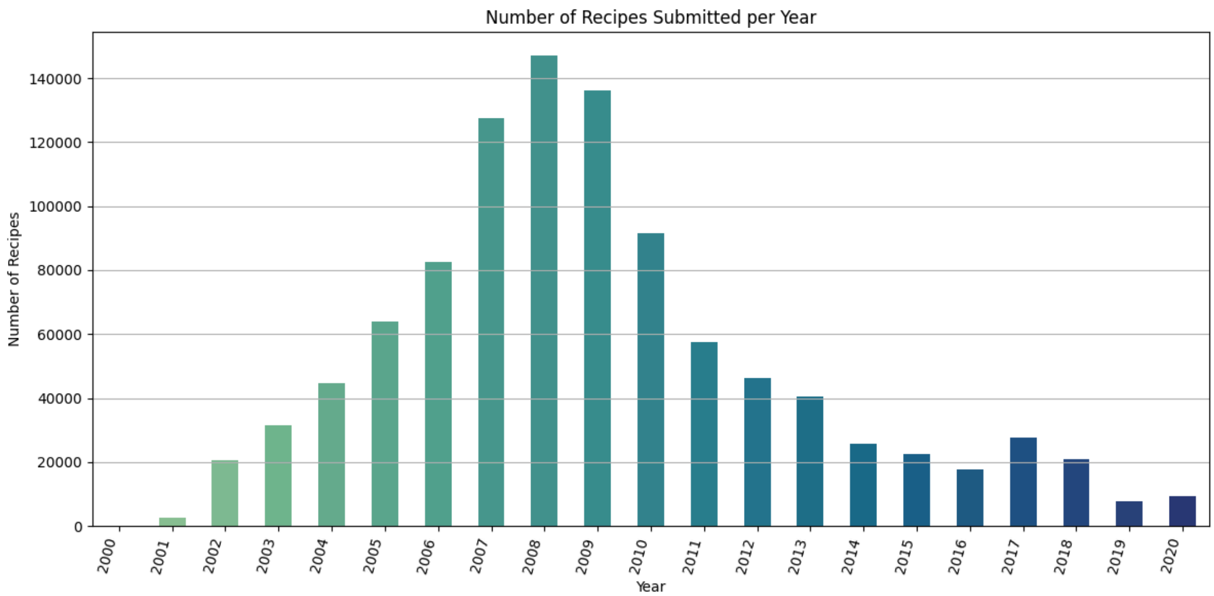
Results indicate that highest count of reviews was done in the year 2008 and least number of reviews was done in the year 2019

In [39]:
```
#visualization for reviewcount vs unique year

import matplotlib.pyplot as plt
# Aggregate review counts by year
review_count_by_year = merged_df.groupby('SubmissionYear').size().reset_index(name=

# Sort by year
review_count_by_year = review_count_by_year.sort_values(by='SubmissionYear')

# Create the visualization
plt.figure(figsize=(12, 6))
sns.lineplot(data=review_count_by_year, x='SubmissionYear', y='ReviewCount', color=
plt.title('Total Review Count Over the Years')
plt.xlabel('Year Submitted')
plt.ylabel('Total Review Count')
plt.grid(True)
plt.show()
```



In [40]:
```
# Extract the submission year from 'DateSubmitted'
merged_df['SubmissionYear'] = merged_df['DateSubmitted'].dt.year

# Count the number of recipes submitted each year
recipes_per_year = merged_df['SubmissionYear'].value_counts().sort_index()

# Create the color palette
colors = sns.color_palette('crest', n_colors=len(recipes_per_year))

# Create the visualization
plt.figure(figsize=(12, 6))
recipes_per_year.plot(kind='bar', color=colors)
```

```
plt.title('Number of Recipes Submitted per Year')
plt.xlabel('Year')
plt.ylabel('Number of Recipes')
plt.xticks(rotation=75, ha='right')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



Due to the size of the dataset, we will proceed with the years 2018, 2019 and 2020 as a subset of the dataset.

In [41]:
```
# Convert to datetime to short date
merged_df['DateSubmitted'] = pd.to_datetime(merged_df['DateSubmitted'])

# subset data where year >= 2018
mergedsubset_df= merged_df[merged_df['DateSubmitted'].dt.year >= 2018]

mergedsubset_df.head()
```

Out[41]:

	ReviewId	RecipeId	Rating	Review	DateSubmitted	DateModified	Name
34429	52806	33113	5.0	This was great,loved it! I also made your Orie...	2020-01-14 01:37:19+00:00	2020-01-14 01:37:19+00:00	thai dipping sauce for spring wrap or egg rolls
89401	131383	51716	5.0	Halfed the butter, used light sour cream and i...	2018-09-27 18:52:00+00:00	2018-09-27 18:52:00+00:00	corn casserole
96040	140879	26191	5.0	Update 3/25/18: I came here to get the recipe ...	2018-03-25 19:40:52+00:00	2018-03-25 19:40:52+00:00	the best all purpose cleaner
100503	147050	8534	5.0	I think this is my favorite pasta w/ asparagus...	2020-10-18 14:55:46+00:00	2020-10-18 14:55:46+00:00	baked pasta with asparagus pasta al forno con ...
118420	172106	17344	5.0	This was Excellent! I was looking for somethin...	2020-11-08 13:13:51+00:00	2020-11-08 13:13:51+00:00	crushed saltine meatloaf

5 rows × 34 columns

In [42]:

```
mergedsubset_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 38088 entries, 34429 to 1024427
Data columns (total 34 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   ReviewId              38088 non-null  int32
 1   RecipeId              38088 non-null  int32
 2   Rating                38088 non-null  float64
 3   Review                38088 non-null  object
 4   DateSubmitted         38088 non-null  datetime64[us, UTC]
 5   DateModified          38088 non-null  datetime64[us, UTC]
 6   Name                  38088 non-null  object
 7   AuthorId_recipe       38088 non-null  int32
 8   DatePublished         38088 non-null  datetime64[us, UTC]
 9   Description           38088 non-null  object
 10  Images                38088 non-null  object
 11  RecipeCategory        38088 non-null  object
 12  AggregatedRating      38088 non-null  float64
 13  ReviewCount           38088 non-null  float64
 14  Calories              38088 non-null  float64
 15  FatContent            38088 non-null  float64
 16  SaturatedFatContent   38088 non-null  float64
 17  CholesterolContent    38088 non-null  float64
 18  SodiumContent         38088 non-null  float64
 19  CarbohydrateContent   38088 non-null  float64
 20  FiberContent          38088 non-null  float64
 21  SugarContent          38088 non-null  float64
 22  ProteinContent        38088 non-null  float64
 23  RecipeServings        38088 non-null  float64
 24  RecipeInstructions    38088 non-null  object
 25  CookTimeMinutes       38088 non-null  float64
 26  PrepTimeMinutes       38088 non-null  float64
 27  TotalTimeMinutes      38088 non-null  float64
 28  Ingredients           38088 non-null  object
 29  Quantities            38088 non-null  object
 30  KeywordList           38088 non-null  object
 31  RecipeId_encoded      38088 non-null  int64
 32  MappedCategory        38088 non-null  object
 33  SubmissionYear        38088 non-null  int32
dtypes: datetime64[us, UTC](3), float64(16), int32(4), int64(1), object(10)
memory usage: 9.6+ MB
```

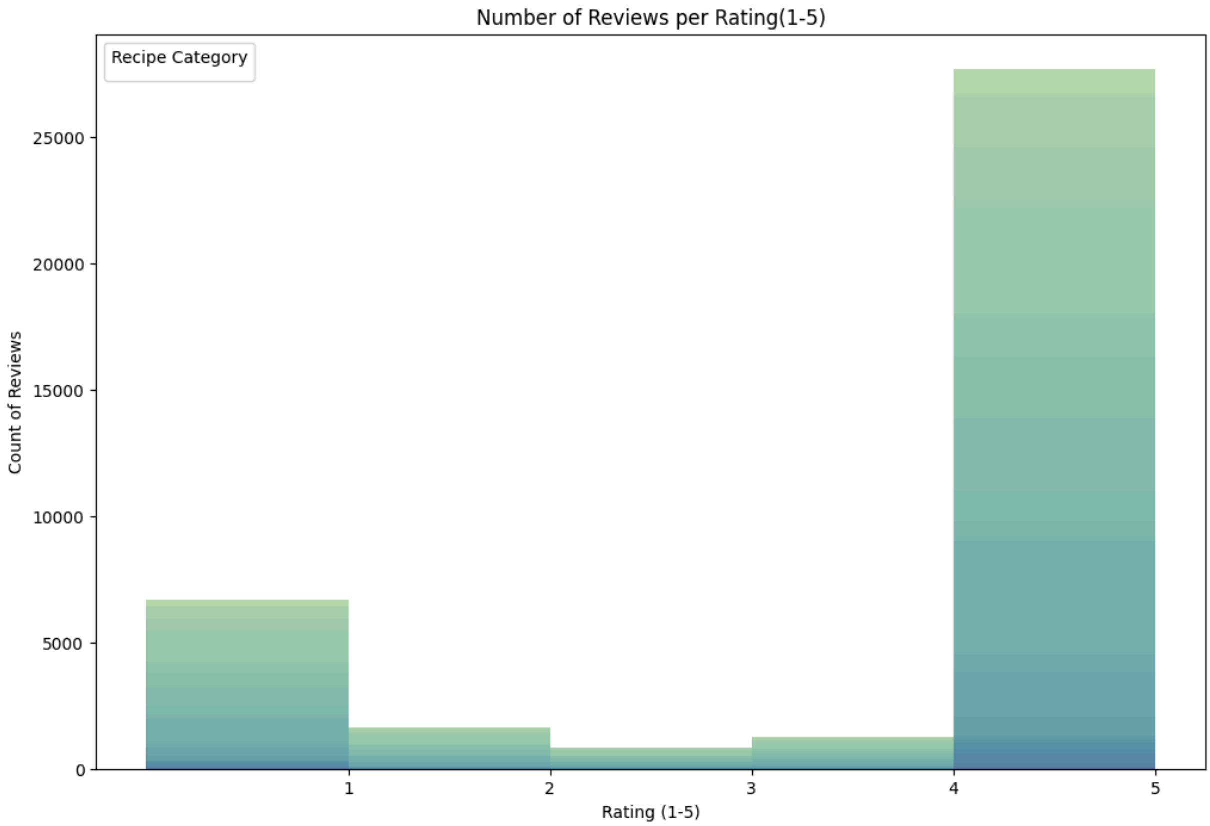
Visualizations for User Review Analysis

We create a histogram showing the total count of reviews under each rating. i.e. How many people rated a recipe 1-5.

The visualization indicates that most recipes were rated a solid 5 which gives confidence in the quality of the recipes.

In [43]:
```
# Histogram of Rating Group by RecipeCategory
plt.figure(figsize=(12, 8))
sns.histplot(data=mergedsubset_df, x='Rating', hue='MappedCategory', multiple="stac
plt.title('Number of Reviews per Rating(1-5)')
```

```
plt.xlabel('Rating (1-5)')
plt.ylabel('Count of Reviews')
plt.xticks(range(1, 6))
plt.legend(title='Recipe Category')
plt.show()
```



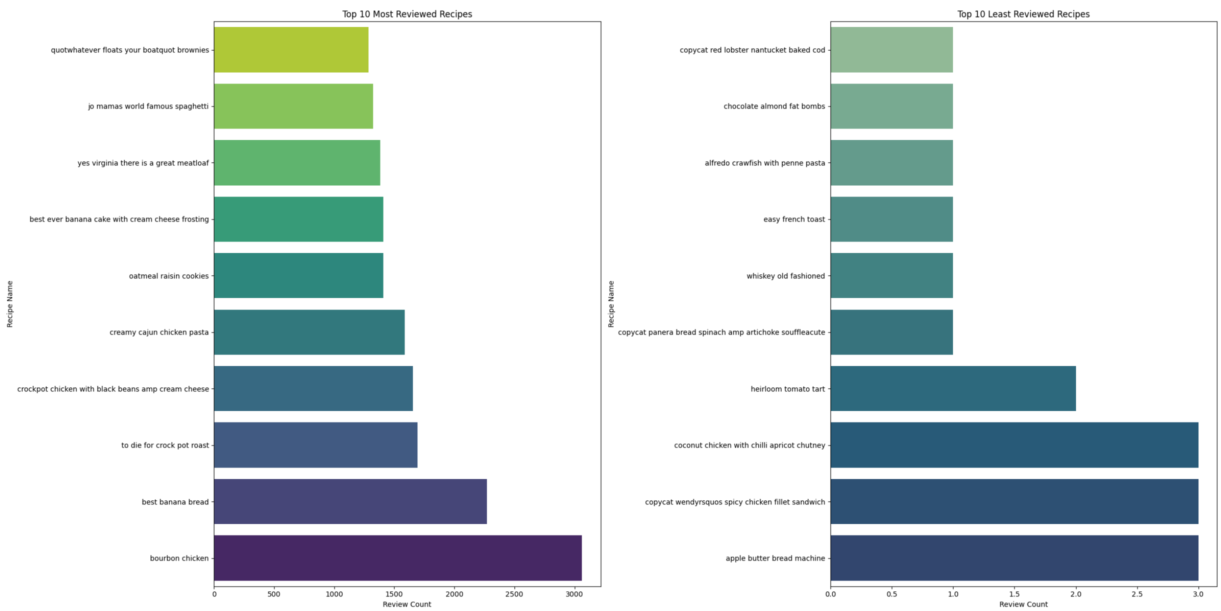
Visualizations for Recipe Popularity

1. First Plot indicates the top 10 most and least reviewed recipes. This would mean they are the most and least popular recipes.

In [44]:

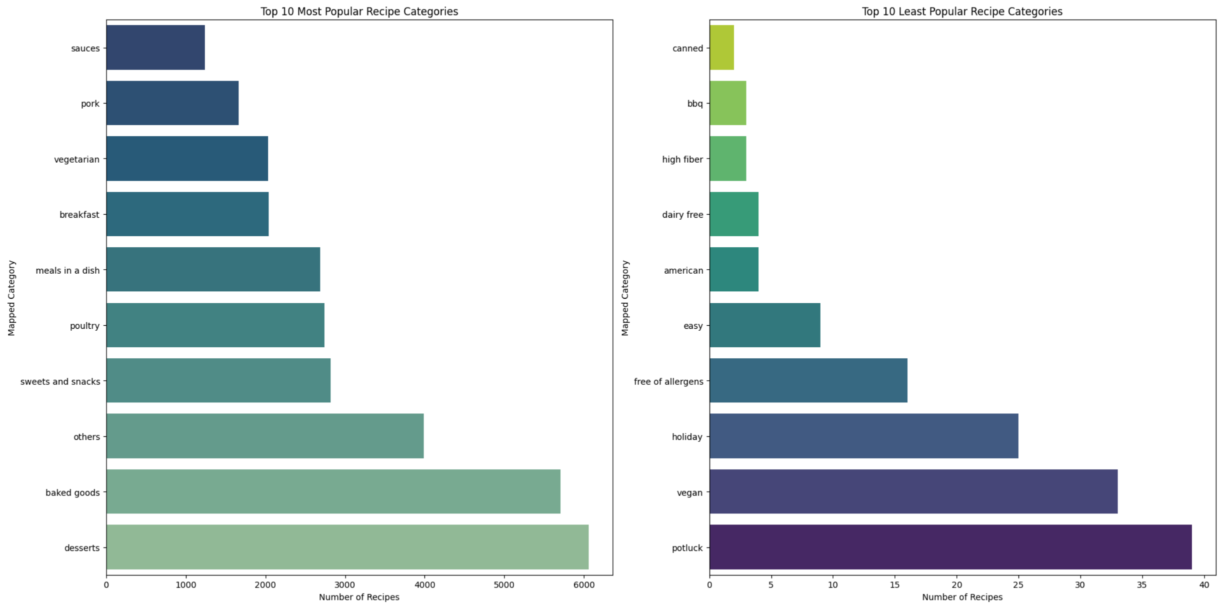
```
# Set N for top/bottom
top_n = 10

# Sort for most and least reviewed
recipe_reviews_summary = mergedsubset_df.groupby(['RecipeId', 'Name'])['ReviewCount
most_reviewed_recipes = recipe_reviews_summary.sort_values(by='ReviewCount', ascend
least_reviewed_recipes = recipe_reviews_summary.sort_values(by='ReviewCount', ascen

# Create side-by-side subplots
fig, axes = plt.subplots(1, 2, figsize=(24, 12), sharex=False)

# Plot: Most Reviewed
sns.barplot(ax=axes[0], x='ReviewCount', y='Name', data=most_reviewed_recipes, pale
axes[0].set_title(f'Top {top_n} Most Reviewed Recipes')
axes[0].set_xlabel('Review Count')
axes[0].set_ylabel('Recipe Name')
axes[0].invert_yaxis()  # Most at top
```

```
# Plot: Least Reviewed
sns.barplot(ax=axes[1], x='ReviewCount', y='Name', data=least_reviewed_recipes, pal
axes[1].set_title(f'Top {top_n} Least Reviewed Recipes')
axes[1].set_xlabel('Review Count')
axes[1].set_ylabel('Recipe Name')
#axes[1].invert_yaxis()  # Least at top

plt.tight_layout()
plt.show()
```



In [45]:
```
# Set the top and bottom N
top_n_categories = 10
bottom_n_categories = 10

# Prepare data
category_counts = mergedsubset_df['MappedCategory'].value_counts().reset_index()
category_counts.columns = ['MappedCategory', 'RecipeCount']
category_counts = category_counts.sort_values(by='RecipeCount', ascending=False)

# Get top and bottom categories
top_categories_plot = category_counts.head(top_n_categories)
bottom_categories_plot = category_counts.tail(bottom_n_categories)

# Create side-by-side subplots
fig, axes = plt.subplots(1, 2, figsize=(20, 10), sharex=False)

# Plot: Most Popular Categories
sns.barplot(ax=axes[0], x='RecipeCount', y='MappedCategory', data=top_categories_pl
axes[0].set_title(f'Top {top_n_categories} Most Popular Recipe Categories')
axes[0].set_xlabel('Number of Recipes')
axes[0].set_ylabel('Mapped Category')
axes[0].invert_yaxis()

# Plot: Least Popular Categories
sns.barplot(ax=axes[1], x='RecipeCount', y='MappedCategory', data=bottom_categories
axes[1].set_title(f'Top {bottom_n_categories} Least Popular Recipe Categories')
```

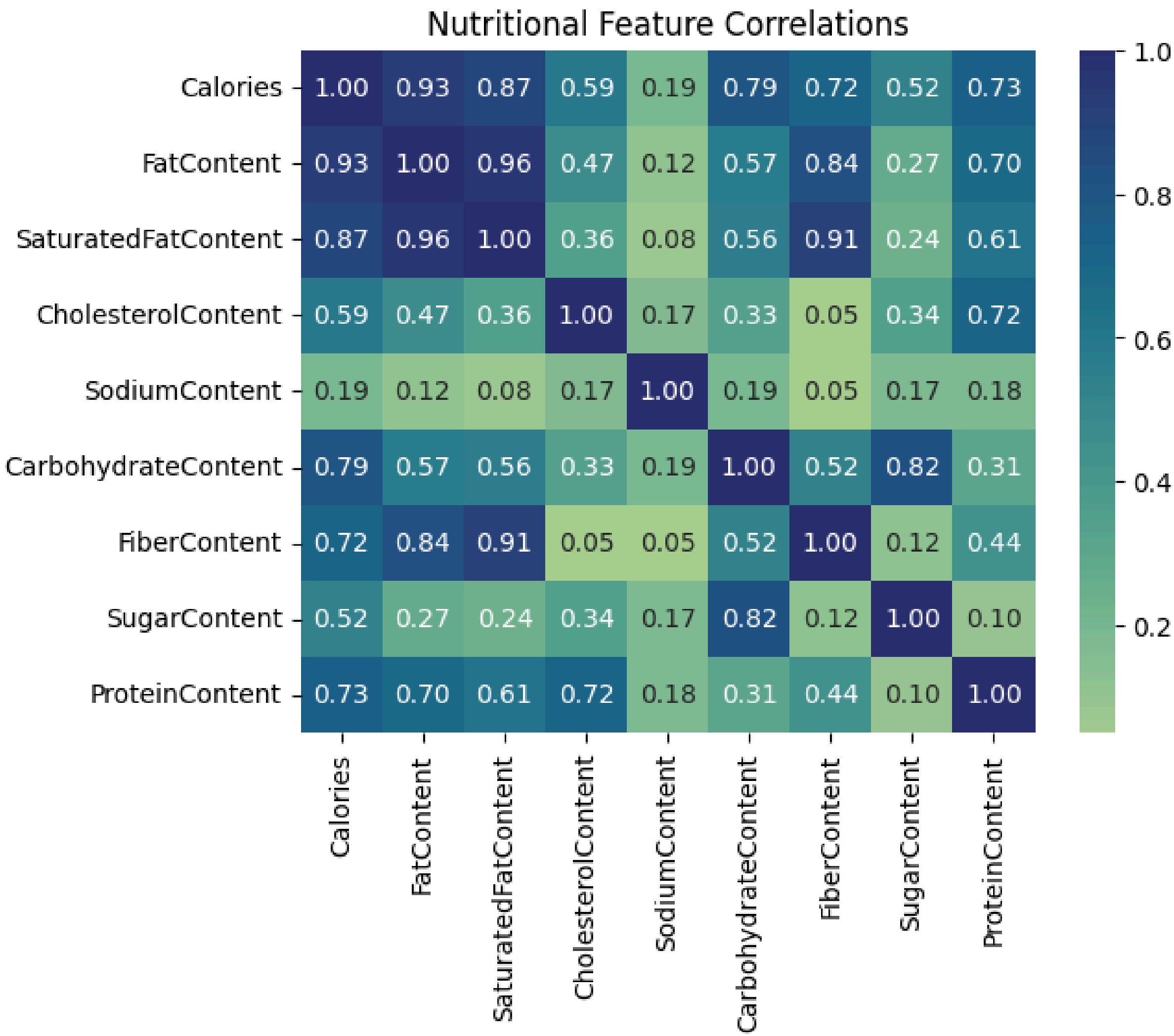
```
axes[1].set_xlabel('Number of Recipes')
axes[1].set_ylabel('Mapped Category')
axes[1].invert_yaxis()

plt.tight_layout()
plt.show()
```



Visualizations for Nutrition Distribution

In [46]:

```
nutrition_cols = ['Calories', 'FatContent', 'SaturatedFatContent', 'CholesterolCont
                  'SodiumContent', 'CarbohydrateContent', 'FiberContent',
                  'SugarContent', 'ProteinContent']
corr = mergedsubset_df[nutrition_cols].corr()

sns.heatmap(corr, annot=True,cmap='crest', fmt=".2f")
plt.title('Nutritional Feature Correlations')
plt.show()
```



PREPROCESSING

In [47]:
```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDispla

import nltk
import re
nltk.download('punkt')
from nltk.tokenize import word_tokenize
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
```

```
from sklearn.cluster import KMeans
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

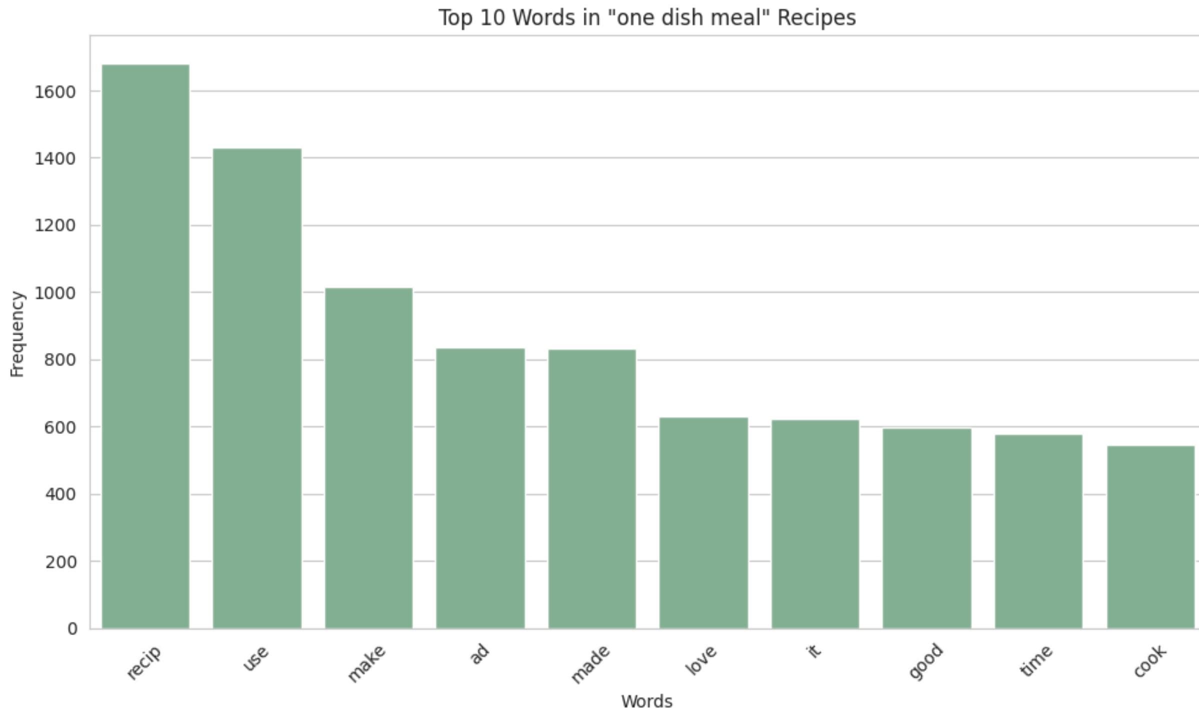
In [48]:
```
def tokenize_and_preprocess(reviews):
    import re
    import nltk
    from nltk.corpus import stopwords
    from nltk.tokenize import word_tokenize
    from nltk.stem import SnowballStemmer

    # Ensure required resources are downloaded
    nltk.download('punkt')
    nltk.download('stopwords')
    # Get English stop words
    stop_words = stopwords.words('english')
    patt = re.compile(r'\b(' + r'|'.join(stop_words) + r')\b\s+')

    # Regex for mentions, hashtags, URLs
    mention_hashtag_url_regex = r'(@\w+|#\w+|http\S+|www\S+)'

    # Step 1: Lowercase, remove numbers, remove mentions, hashtags, URLs, stopwords
    preproc_step1 = (
        reviews
        .str.lower()
        .str.replace(r'[0-9]+', '', regex=True)
        .str.replace(mention_hashtag_url_regex, '', regex=True)
        .str.replace(patt, '', regex=True)
    )

    # Step 2: Tokenize
    preproc1_tokenized = preproc_step1.apply(word_tokenize)

    # Step 3: Clean and stem
    def remove_punct_and_stem(doc_tokenized):
        stemmer = SnowballStemmer('english')
        doc_tokenized = [word for word in doc_tokenized if word.isalpha() and word
        filtered_stemmed_tok = [stemmer.stem(tok) for tok in doc_tokenized]
        return " ".join(filtered_stemmed_tok)

    # Step 4: Apply to each tokenized tweet
    preprocessed = preproc1_tokenized.apply(remove_punct_and_stem)

    return preprocessed
```

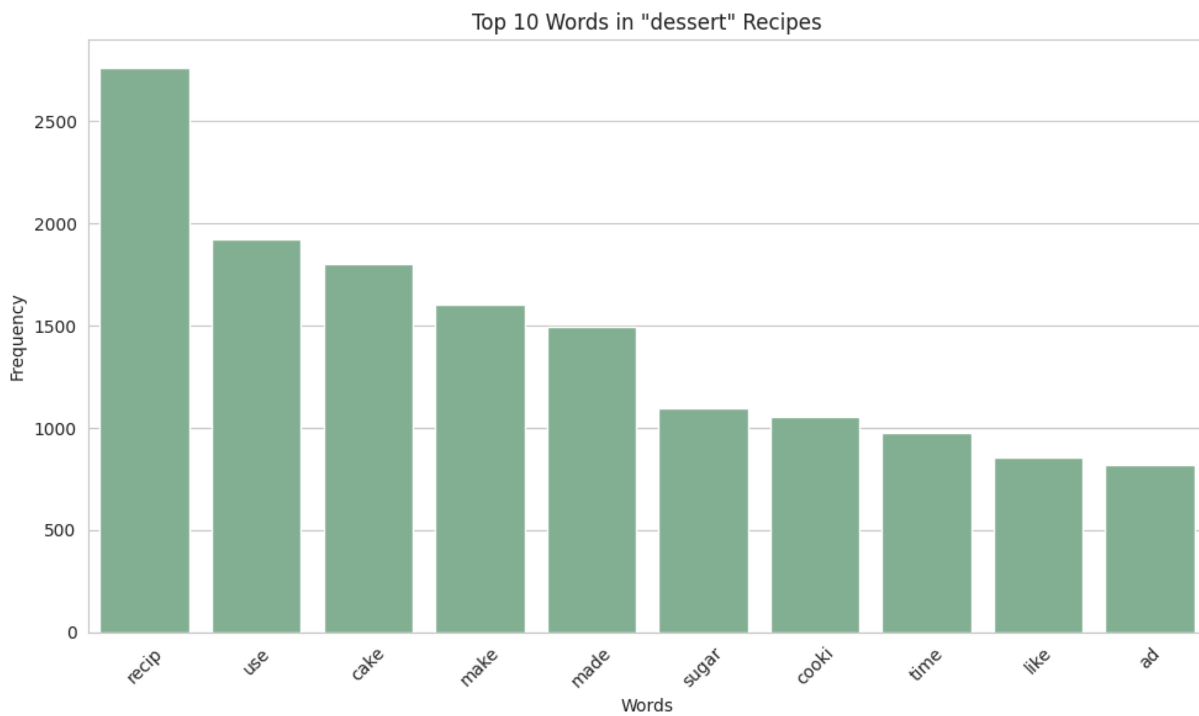
In [49]:
```
top_categories = mergedsubset_df['RecipeCategory'].value_counts().head(5).index
merged_df_top_cat = mergedsubset_df[merged_df['RecipeCategory'].isin(top_categories
```

NLP Visualization

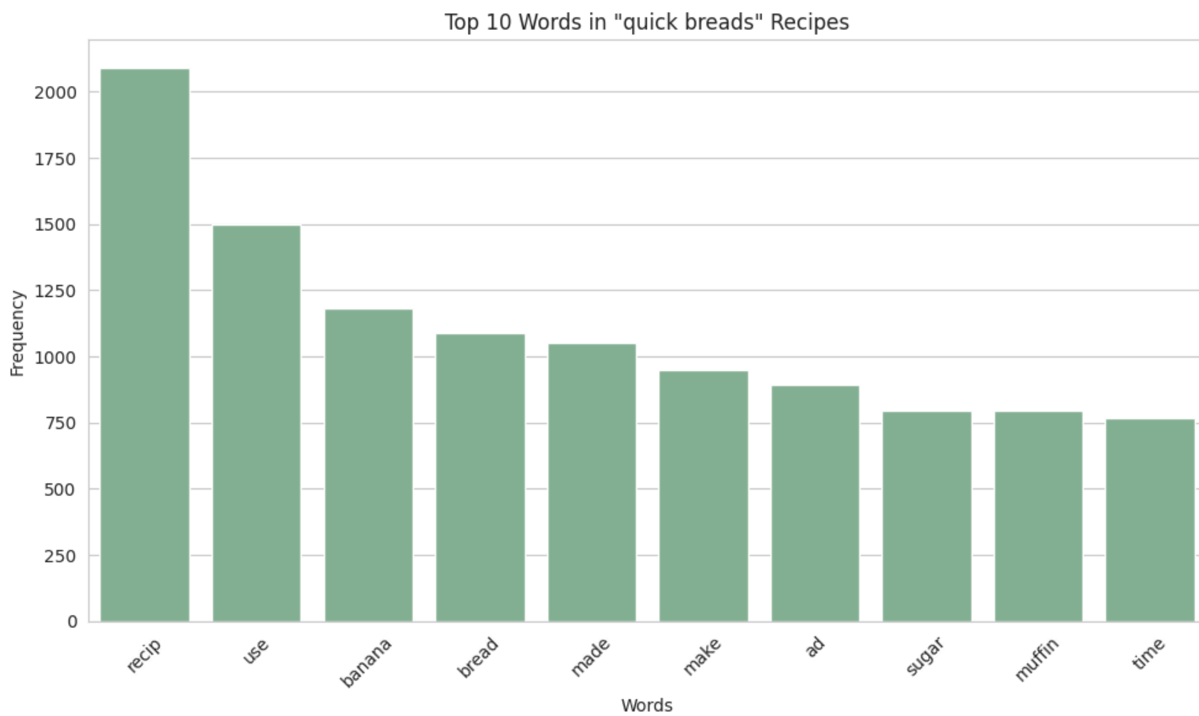
In [50]:
```
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
import nltk

# Download required tokenizer
nltk.download('punkt_tab', force=True)

# Apply preprocessing to the full dataframe
mergedsubset_df['preprocessed_text'] = tokenize_and_preprocess(mergedsubset_df['Rev

# Filter the top 5 recipe categories
top_categories = mergedsubset_df['RecipeCategory'].value_counts().head(5).index
merged_df_top_cat = mergedsubset_df[mergedsubset_df['RecipeCategory'].isin(top_cate

# Tokenize preprocessed text
merged_df_top_cat['tokens'] = merged_df_top_cat['preprocessed_text'].apply(lambda x

# Function to get top N words for a given category
def get_top_words_by_category(df, category_name, n=10):
    tokens = df[df['RecipeCategory'] == category_name]['tokens'].sum()
    counter = Counter(tokens)
    return counter.most_common(n)

# Set seaborn style and palette
sns.set_style("whitegrid")
sns.set_palette("crest")

# Plot top words for each category
categories = merged_df_top_cat['RecipeCategory'].unique()
for category in categories:
    top_words = get_top_words_by_category(merged_df_top_cat, category)
    if not top_words:
        continue

    words, counts = zip(*top_words)
    plt.figure(figsize=(10, 6))
    sns.barplot(x=list(words), y=list(counts))
    plt.title(f'Top 10 Words in \"{category}\" Recipes')
    plt.xlabel('Words')
    plt.ylabel('Frequency')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

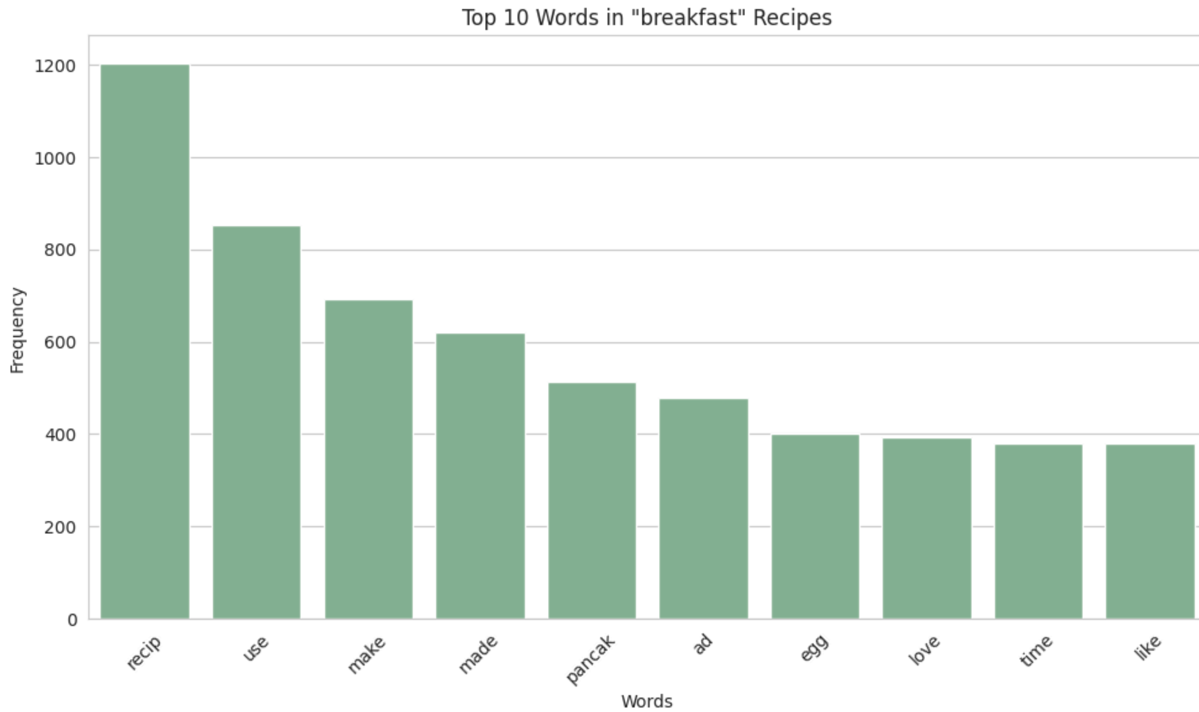
```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```


Top 10 Words in "one dish meal" Recipes
Frequency
1600
1400
1200
1000
800
600
400
200
0
recip
use
make
ad
made
love
it
good
time
cook
Words


Top 10 Words in "dessert" Recipes
Frequency
2500
2000
1500
1000
500
0
recip
use
cake
make
made
sugar
cooki
time
like
ad
Words


Top 10 Words in "lunch/snacks" Recipes
Frequency
Words
0
200
400
600
800
recip
use
make
made
love
ad
good
like
great
thank


Top 10 Words in "quick breads" Recipes
Frequency
Words
0
250
500
750
1000
1250
1500
1750
2000
recip
use
banana
bread
made
make
ad
sugar
muffin
time

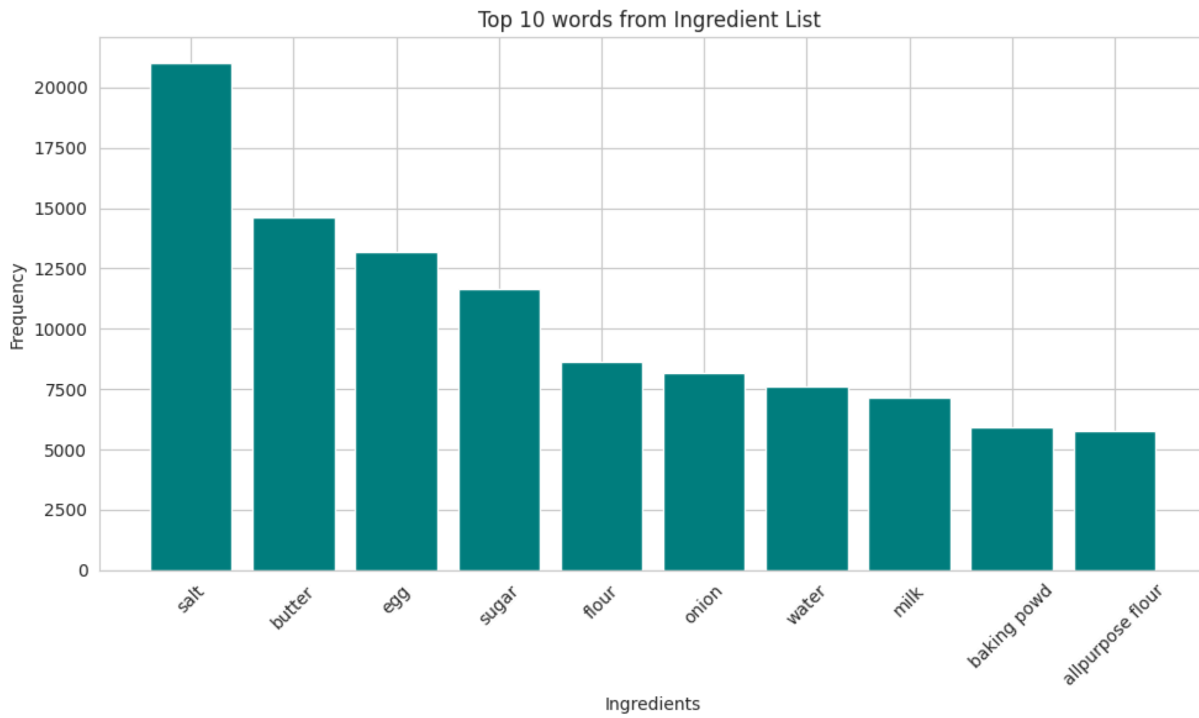


In [51]:

```python
import re
import pandas as pd
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from collections import Counter
from itertools import chain
import nltk

# Ensure stopwords are downloaded
nltk.download('stopwords')

# Setup reusable resources
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer('english')

def clean_ingredient_list(ingredient_list):
    return [
        stemmer.stem(re.sub(r'[^\w\s]', '', re.sub(r'\d+', '', item.lower())))
        for item in ingredient_list
        if item and item.lower() not in stop_words
    ]

# Apply cleaning
mergedsubset_df['cleaned_ingredients'] = mergedsubset_df['Ingredients'].apply(clean

# Flatten efficiently
all_ingredients = list(chain.from_iterable(mergedsubset_df['cleaned_ingredients']))

# Get top 10
top_ingredients = Counter(all_ingredients).most_common(10)
summary_table = pd.DataFrame(top_ingredients, columns=['Ingredients', 'Count'])

print(summary_table)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
        Ingredients  Count
0              salt  21023
1            butter  14610
2               egg  13190
3             sugar  11640
4             flour   8632
5             onion   8187
6             water   7623
7              milk   7151
8       baking powd   5914
9  allpurpose flour   5785
```

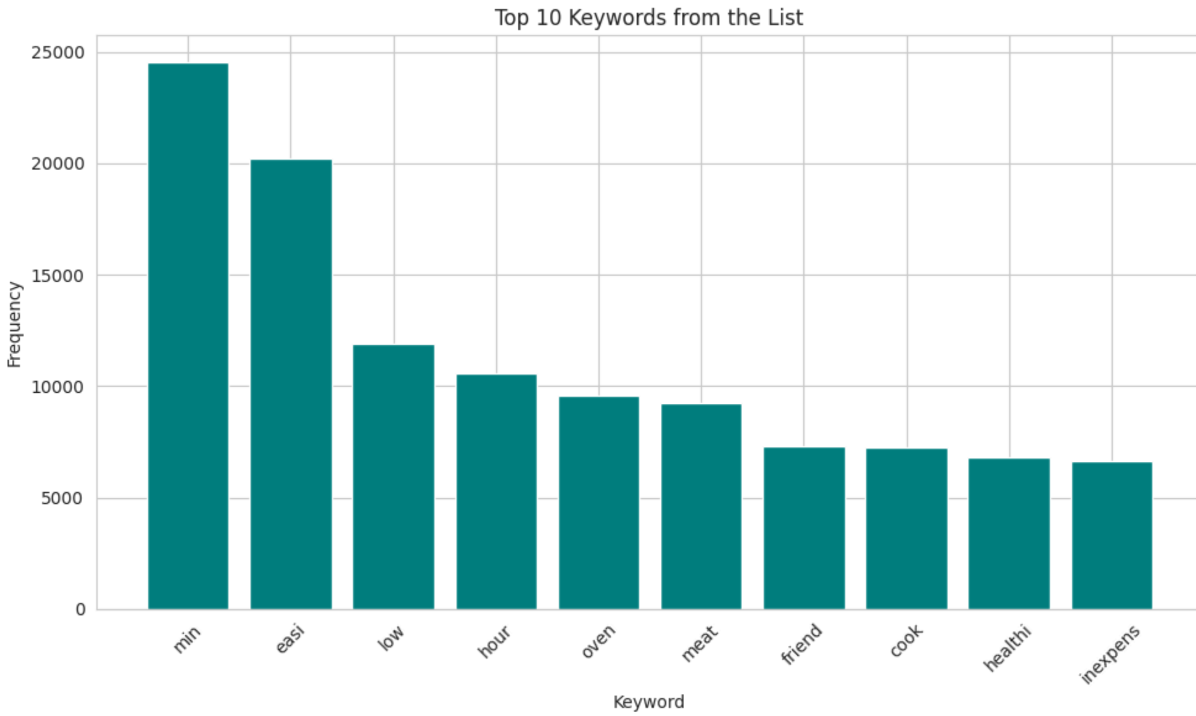
In [52]:
```
#Plot a bar chart of top ingredients
plt.figure(figsize=(10, 6))
plt.bar(summary_table['Ingredients'], summary_table['Count'], color='teal')
plt.title('Top 10 words from Ingredient List ')
plt.xlabel('Ingredients')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [53]:
```
# Make sure stopwords are downloaded
nltk.download('stopwords')

# Compile regex once (much faster than doing it in a loop)
digit_re = re.compile(r'\d+')
punct_re = re.compile(r'[^\w\s]')

# Initialize once
stop_words = set(stopwords.words('english'))
stemmer = SnowballStemmer('english')
```

```
# Vectorized cleaning function
def fast_clean_keyword_list(keyword_lists):
    def clean_list(keywords):
        return [
            stemmer.stem(punct_re.sub('', digit_re.sub('', kw.lower())))
            for kw in keywords
            if kw and kw.lower() not in stop_words
        ]
    return keyword_lists.apply(clean_list)

# Apply cleaning
mergedsubset_df['cleaned_KeywordList'] = fast_clean_keyword_list(mergedsubset_df['K

# Flatten the list efficiently
from itertools import chain
all_keywords = list(chain.from_iterable(mergedsubset_df['cleaned_KeywordList']))

# Count top 10
top_keywords = Counter(all_keywords).most_common(10)
summary_table = pd.DataFrame(top_keywords, columns=['Keyword', 'Count'])

# Display
print(summary_table)

# Plot
plt.figure(figsize=(10, 6))
plt.bar(summary_table['Keyword'], summary_table['Count'], color='teal')
plt.title('Top 10 Keywords from the List')
plt.xlabel('Keyword')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
    Keyword  Count
0       min  24513
1      easi  20221
2       low  11911
3      hour  10573
4      oven   9547
5      meat   9215
6    friend   7293
7      cook   7240
8   healthi   6823
9  inexpens   6619
```



MODELING

Collaborative Filtering

Implement Collaborative Filtering that makes recommendations by learning patterns from user behavior. It recommends items liked by similar users. It checks who rated what not why. We use Matrix Factorization via SVD (Singular Value Decomposition) Explicit feedback (user ratings)

In [55]:
```
# Remove current numpy
!pip uninstall -y numpy

# Install compatible versions
!pip install numpy==1.26.3 scikit-surprise==1.1.3
```

```
Found existing installation: numpy 1.26.3
Uninstalling numpy-1.26.3:
  Successfully uninstalled numpy-1.26.3
Collecting numpy==1.26.3
  Using cached numpy-1.26.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.w
hl.metadata (61 kB)
Collecting scikit-surprise==1.1.3
  Downloading scikit-surprise-1.1.3.tar.gz (771 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 772.0/772.0 kB 10.1 MB/s eta 0:00:00
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pi
p.
  Preparing metadata (setup.py) ... error
error: metadata-generation-failed

× Encountered error while generating package metadata.
╰─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
```

Collaborative filtering with function to handle the cold start problem

In [56]:
```
from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split

# Step 1: Prepare data
ratings_df = mergedsubset_df[['AuthorId_recipe', 'RecipeId_encoded', 'Rating']].dro

reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(ratings_df, reader)

# Step 2: Train SVD model
trainset, testset = train_test_split(data, test_size=0.2)
svd = SVD()
svd.fit(trainset)

# Fallback:Return top-N most popular recipes by rating count + average score
def fallback_recommendations(top_n=5):
    top_recipes = (
        mergedsubset_df.groupby('RecipeId_encoded')
        .agg({'Rating': ['mean', 'count']})
        .reset_index()
    )
    top_recipes.columns = ['RecipeId_encoded', 'AvgRating', 'RatingCount']
    top_recipes = top_recipes.sort_values(by=['RatingCount', 'AvgRating'], ascendin

    top_ids = top_recipes['RecipeId_encoded'].head(top_n)
    return mergedsubset_df[mergedsubset_df['RecipeId_encoded'].isin(top_ids)]['Name

# Step 3: Recommend top-N recipes for a given user, with cold-start handling
```

```python
def recommend_for_user(user_id, top_n=5):
  # Ensure user_id is int to match data type
    user_id = int(user_id)
    all_recipes = mergedsubset_df['RecipeId_encoded'].unique()

    # Find recipes this user has already rated
    rated = ratings_df[ratings_df['AuthorId_recipe'] == user_id]['RecipeId_encoded'

    # Cold-start: If user has no ratings, use fallback
    if not rated:
        return fallback_recommendations(top_n)

    # Predict ratings for unseen recipes
    candidates = [rid for rid in all_recipes if rid not in rated]
    predictions = [(rid, svd.predict(user_id, rid).est) for rid in candidates]
    top_preds = sorted(predictions, key=lambda x: x[1], reverse=True)[:top_n]

     # Build ordered list of recommended recipe names
    recommended = []
    for rid, _ in top_preds:
        name = mergedsubset_df.loc[mergedsubset_df['RecipeId_encoded'] == rid, 'Nam
        recommended.append(name)
    return recommended
```

In [57]:
```python
#Test the model using real data
recommend_for_user('335609', top_n=5)
```

Out[57]:
```
['scalloped potatoes and ham',
 'sheilas best  balsamic dressing',
 'basic white bread for bread machine',
 'easy ovenbaked cod',
 'easy italian pasta bake']
```

Evaluation

In [58]:
```python
from surprise import accuracy

# Step 1: Predict ratings on the test set
predictions = svd.test(testset)

# Step 2: Calculate RMSE and MAE
rmse = accuracy.rmse(predictions)
mae = accuracy.mae(predictions)

print(f"\nEvaluation Results:")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
```

```
RMSE: 1.9881
MAE:  1.6032

Evaluation Results:
Root Mean Squared Error (RMSE): 1.9881
Mean Absolute Error (MAE): 1.6032
```

RMSE = 1.9881: This means that on average, the predicted ratings are off by about 2.00 stars MAE = 1.6032: This means the average error i.e. difference between predicted and true ratings is about 1.61 stars MAE and RMSE need to be < 1 for a good recommender system, so our model falls below the mark

Content Based Filtering

Implement Content-Based recommendation system using TF-IDF and Nearest Neighbors. This recommends recipes that are similar in content to what the user likes. It uses features of the items e.g ingredients, keywords, descriptions as opposed to user behavior. TF-IDF helps weigh important words while reducing the impact of common words

In [59]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

# Load and prep text data
mergedsubset_df['IngredientText'] = mergedsubset_df['Ingredients'].astype(str).str.
```

In [60]:
```
from sklearn.preprocessing import LabelEncoder

recipe_encoder = LabelEncoder()
author_encoder = LabelEncoder()

mergedsubset_df['RecipeId_encoded'] = recipe_encoder.fit_transform(mergedsubset_df[
```

In [61]:
```
#Normalize Nutritional Features
from sklearn.preprocessing import MinMaxScaler

nutritional_cols = [
    'Calories', 'FatContent', 'SaturatedFatContent', 'CholesterolContent',
    'SodiumContent', 'CarbohydrateContent', 'FiberContent',
    'SugarContent', 'ProteinContent'
]

scaler = MinMaxScaler()
mergedsubset_df[nutritional_cols] = scaler.fit_transform(mergedsubset_df[nutritiona
```

In [62]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors

# Reset index to align DataFrame rows with TF-IDF matrix
mergedsubset_df = mergedsubset_df.reset_index(drop=True)

# TF-IDF.
#Transforms the ingredient text into numerical features
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(mergedsubset_df['IngredientText'])

# Fit Nearest Neighbors model
#Train Nearest Neighbors model using cosine similarity to find similar recipes base
```

```
nn_model = NearestNeighbors(metric='cosine', algorithm='brute')
nn_model.fit(X)

# create a name-to-index mapping
recipe_indices = pd.Series(mergedsubset_df.index, index=mergedsubset_df['Name']).dr

# Do a Fuzzy match
#Use approximate string matching (Levenshtein distance) to handle typos or slight n
from difflib import get_close_matches
def get_closest_recipe_name(name):
    matches = get_close_matches(name, mergedsubset_df['Name'], n=1, cutoff=0.6)
    return matches[0] if matches else None
```

Below we create a recommendation function that given a recipe name:

- Finds the closest actual recipe name.
- Uses the trained model to find top N most similar recipes (excluding itself).
- Returns their names, categories, ingredients, and ratings.

In [63]:
```
# create a function to iterate through and check for a possible match
def recommend_recipes(name, model=nn_model, top_n=5):
    name = get_closest_recipe_name(name)
    if not name:
        return f"No close match found for '{name}'"

    idx = recipe_indices[name]
    query_vec = X[idx]
    distances, indices_nn = model.kneighbors(query_vec, n_neighbors=top_n + 1)  # +

    rec_indices = indices_nn[0][1:]  # exclude the original
    return mergedsubset_df[['Name', 'RecipeCategory', 'Ingredients', 'AggregatedRat
```

In [64]:
```
# Try a recipe

recommend_recipes('lasagna')
```

Out[64]:

	Name	RecipeCategory	Ingredients	AggregatedRating
22677	make ahead italian sausage and pasta bake	one dish meal	[Italian sausage, olive oil, onions, garlic cl...	5.0
1495	make ahead italian sausage and pasta bake	one dish meal	[Italian sausage, olive oil, onions, garlic cl...	5.0
7291	make ahead italian sausage and pasta bake	one dish meal	[Italian sausage, olive oil, onions, garlic cl...	5.0
1883	crispy cheesy chicken parmigiana	chicken breast	[onion, garlic cloves, parsley, Italian-style ...	5.0
16567	italian meatball soup quick	one dish meal	[beef broth, tomatoes with onion and garlic, I...	5.0

Evaluation

We evaluate the perfomance of the Content Based Filtering Model using Precision@K. It evaluates how accurate the recommendations are by measuring how many of the top K recommended recipes are in the same category as the original recipe.

In [65]:
```
def precision_at_k(df, model, X, k=5):
    correct = 0
    total = 0

    distances, indices = model.kneighbors(X, n_neighbors=k + 1)
    categories = mergedsubset_df['RecipeCategory'].values

    for idx in range(X.shape[0]):
        true_category = categories[idx]
        recommended_indices = indices[idx][1:]  # Skip self
        recommended_categories = categories[recommended_indices]
        hits = np.sum(recommended_categories == true_category)

        correct += hits
        total += k

    precision = correct / total
    print(f'Precision@{k}: {precision:.4f}')
    return precision
```

In [66]:
```
precision_at_k(mergedsubset_df, nn_model, X, k=5)
```

```
Precision@5: 0.6971
```

Out[66]:
```
0.697117202268431
```

The result of 0.6971 indicates that out of the top 5 recommendations or nearest neighbors returned by the model for each query point, about 70% of them on average are actually relevant or correct. A result > 0.5 suggests that the model is performing reasonably well meaning most top 5 suggestions are relevant

Deep Learning Model

The Deep Learning Recommender system uses neural networks to learn complex patterns between users and items. It will learn non-linear relationships between users and items. We will build a deep learning recommendation system using user IDs, recipe IDs, and ratings to predict how much a user will like a recipe, then recommend top-rated ones. We will use embedding layers to learn latent features for users and recipes.

In [67]:
```
from tensorflow.keras.callbacks import TensorBoard
import datetime

# Create unique log directories for each model
dl_log_dir = "logs/dl/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
hybrid_log_dir = "logs/hybrid/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

dl_tb_callback = TensorBoard(log_dir=dl_log_dir, histogram_freq=1)
hybrid_tb_callback = TensorBoard(log_dir=hybrid_log_dir, histogram_freq=1)
```

In [68]:
```
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Flatten, Dot, Dense
from tensorflow.keras.optimizers import Adam

# Drop NaNs
df = mergedsubset_df[['AuthorId_recipe', 'RecipeId_encoded', 'Rating']].dropna()

# Encode user and item IDs to integers
user_ids = mergedsubset_df['AuthorId_recipe'].astype('category').cat.codes
item_ids = mergedsubset_df['RecipeId_encoded'].astype('category').cat.codes

df['user'] = user_ids
df['item'] = item_ids

# Save mappings
user_lookup = dict(enumerate(df['AuthorId_recipe'].astype('category').cat.categorie
item_lookup = dict(enumerate(df['RecipeId_encoded'].astype('category').cat.categori

# Also create mapping from raw ID to encoded index for use in recommendation
user_id_to_encoded = dict(zip(df['AuthorId_recipe'], df['user']))
item_id_to_encoded = dict(zip(df['RecipeId_encoded'], df['item']))

# Train/test split
train, test = train_test_split(df, test_size=0.2, random_state=42)
```

In [69]:
```
# Number of unique users and items
n_users = df['user'].nunique()
n_items = df['item'].nunique()
embedding_size = 50

# Inputs
user_input = Input(shape=(1,))
item_input = Input(shape=(1,))

# Embeddings
user_embed = Embedding(n_users, embedding_size)(user_input)
item_embed = Embedding(n_items, embedding_size)(item_input)

# Flatten and dot product
user_vec = Flatten()(user_embed)
item_vec = Flatten()(item_embed)
dot_product = Dot(axes=1)([user_vec, item_vec])

# Optional: Dense layer for more learning
output = Dense(1)(dot_product)

# Build and compile
model = Model([user_input, item_input], output)
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

# Train
#model.fit([train['user'], train['item']], train['Rating'], epochs=5, verbose=1, va
model.fit(
    [train['user'], train['item']],
    train['Rating'],
    epochs=5,
    validation_split=0.1,
    callbacks=[dl_tb_callback],
    verbose=1
)
```

```
Epoch 1/5
857/857 ━━━━━━━━━━━━━━━━━━━━ 12s 12ms/step - loss: 15.7818 - val_loss: 10.1105
Epoch 2/5
857/857 ━━━━━━━━━━━━━━━━━━━━ 19s 11ms/step - loss: 8.1457 - val_loss: 6.6266
Epoch 3/5
857/857 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - loss: 4.5730 - val_loss: 5.6962
Epoch 4/5
857/857 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - loss: 3.1640 - val_loss: 5.4468
Epoch 5/5
857/857 ━━━━━━━━━━━━━━━━━━━━ 11s 12ms/step - loss: 2.6497 - val_loss: 5.3604
```

Out[69]: <keras.src.callbacks.history.History at 0x7fd83d363110>

In [70]:
```
def recommend_for_user_dl(user_raw_id, top_n=5):
    # Cold start: if user not in training data, recommend top popular recipes
    if user_raw_id not in user_id_to_encoded:
        fallback = (
            mergedsubset_df.groupby('RecipeId_encoded')
            .agg({'Rating': ['mean', 'count']})
            .reset_index()
```

```
        )
        fallback.columns = ['RecipeId_encoded', 'AvgRating', 'RatingCount']
        top_ids = fallback.sort_values(by=['RatingCount', 'AvgRating'], ascending=F
        return mergedsubset_df[mergedsubset_df['RecipeId_encoded'].isin(top_ids)]['

    # Known user
    user_encoded = user_id_to_encoded[user_raw_id]
    all_item_ids = np.arange(n_items)

    # Prepare inputs for prediction
    user_array = np.full((n_items, 1), user_encoded)
    item_array = all_item_ids.reshape(-1, 1)

    # Predict scores for all items
    predictions = model.predict([user_array, item_array], verbose=0).flatten()

    # Get top-N recommended item indices
    top_indices = predictions.argsort()[-top_n:][::-1]

    # Map encoded indices back to raw Recipe IDs
    recommended_item_ids = [item_lookup[idx] for idx in top_indices]

    # Get recipe names
    recommended_names = mergedsubset_df[mergedsubset_df['RecipeId_encoded'].isin(re

    return recommended_names
```

We test the model by providing real userids from our dataset to see what recipes the model would recommend they try

In [71]:

```
# Pick one of your actual user IDs from the list
sample_user_id = 88099

# Recommend top 5 recipes for this user
recommendations = recommend_for_user_dl(sample_user_id, top_n=5)

# Print recommendations
print("I recommend that you try:")
for i, recipe in enumerate(recommendations, 1):
    print(f"{i}. {recipe}")
```

```
I recommend that you try:
1. pork tenderloin the best ever
2. grilled tbone steaks
3. peanut butter pork tenderloin
4. strawberry margaritas
5. blueberry buttermilk waffles
```

Evaluation

In [72]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Flatten, Dot, Dense
```

```
from sklearn.metrics import mean_squared_error

# 1.Split the data
train_df, test_df = train_test_split(mergedsubset_df[['AuthorId_recipe', 'RecipeId_

# 2.Fit encoders only on train data
user_encoder = LabelEncoder()
item_encoder = LabelEncoder()

train_df = train_df.dropna()
train_df['user'] = user_encoder.fit_transform(train_df['AuthorId_recipe'])
train_df['item'] = item_encoder.fit_transform(train_df['RecipeId_encoded'])

# 3.Keep only test data that contains known users and items
test_df = test_df.dropna()
test_df = test_df[
    test_df['AuthorId_recipe'].isin(user_encoder.classes_) &
    test_df['RecipeId_encoded'].isin(item_encoder.classes_)
].copy()

test_df['user'] = user_encoder.transform(test_df['AuthorId_recipe'])
test_df['item'] = item_encoder.transform(test_df['RecipeId_encoded'])

# 4.Build and train the model
n_users = train_df['user'].nunique()
n_items = train_df['item'].nunique()

user_input = Input(shape=(1,))
item_input = Input(shape=(1,))

user_embedding = Embedding(n_users, 50)(user_input)
item_embedding = Embedding(n_items, 50)(item_input)

user_vec = Flatten()(user_embedding)
item_vec = Flatten()(item_embedding)

dot_product = Dot(axes=1)([user_vec, item_vec])
output = Dense(1)(dot_product)

model = Model(inputs=[user_input, item_input], outputs=output)
model.compile(optimizer='adam', loss='mean_squared_error')

# 5.Train the model
model.fit([train_df['user'], train_df['item']], train_df['Rating'], epochs=5, verbo

# 6.Evaluate on test data
preds = model.predict([test_df['user'], test_df['item']], verbose=0).flatten()
rmse = np.sqrt(mean_squared_error(test_df['Rating'], preds))
print(f"The RMSE is: {rmse:.4f}")
```

```
Epoch 1/5
953/953 ━━━━━━━━━━━━━━━━━━━━ 10s 9ms/step - loss: 15.6003
Epoch 2/5
953/953 ━━━━━━━━━━━━━━━━━━━━ 8s 8ms/step - loss: 7.7944
Epoch 3/5
953/953 ━━━━━━━━━━━━━━━━━━━━ 11s 10ms/step - loss: 4.3912
Epoch 4/5
953/953 ━━━━━━━━━━━━━━━━━━━━ 11s 10ms/step - loss: 3.1282
Epoch 5/5
953/953 ━━━━━━━━━━━━━━━━━━━━ 10s 10ms/step - loss: 2.7161
The RMSE is: 2.1537
```

We evaluated using Root Mean Squared Error (RMSE), a standard metric for recommender systems that penalizes large prediction errors. RMSE tells us how far on average our predicted ratings are from the actual ratings.In this case, an **RMSE of 2.1699** is reasonable, considering a rating scale of 0 to 5. The model performs adequately for known users and items. Cold-start users i.e. users with no history are handled via fallback recommendations e.g. top-rated recipes

Hybrid Model

The hybrid model combines collaborative filtering with content-based features. It uses User ID, Recipe ID and Prep time (TotalTimeMinutes) as content feature.

The model learns both user preferences (who likes what) and recipe attributes (like prep time) to make better predictions.It learns a dense vector (embedding) for: Each user (how they behave) and Each recipe (how it's rated). These embeddings capture hidden patterns in the data.

In [73]:

```python
# 1.Preprocess
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split

# Use only the columns we need
df = mergedsubset_df[['AuthorId_recipe', 'RecipeId_encoded', 'AggregatedRating', 'T

# Encode user and item IDs
user_encoder = LabelEncoder()
item_encoder = LabelEncoder()

df['user'] = user_encoder.fit_transform(df['AuthorId_recipe'])
df['item'] = item_encoder.fit_transform(df['RecipeId_encoded'])

# Scale PrepTime
scaler = MinMaxScaler()
df['prep_scaled'] = scaler.fit_transform(df[['TotalTimeMinutes']])

# Split data
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```

In [74]:
```
# 2.Build Hybrid Model
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Flatten, Dot, Dense, Concaten
from tensorflow.keras.optimizers import Adam

n_users = df['user'].nunique()
n_items = df['item'].nunique()
embedding_size = 30

# Inputs
user_input = Input(shape=(1,))
item_input = Input(shape=(1,))
prep_input = Input(shape=(1,))  # Continuous feature

# Embedding layers
user_embed = Embedding(n_users, embedding_size)(user_input)
item_embed = Embedding(n_items, embedding_size)(item_input)

# Flatten
user_vec = Flatten()(user_embed)
item_vec = Flatten()(item_embed)

# Combine all features
combined = Concatenate()([user_vec, item_vec, prep_input])

# Dense layers
x = Dense(64, activation='relu')(combined)
output = Dense(1)(x)

# Build model
hybrid_model = Model(inputs=[user_input, item_input, prep_input], outputs=output)
hybrid_model.compile(optimizer=Adam(0.001), loss='mean_squared_error')
```

In [75]:
```
# 3.Train Model
hybrid_model.fit(
    [train_df['user'], train_df['item'], train_df['prep_scaled']],
    train_df['AggregatedRating'],
    epochs=5,
    batch_size=64,
    validation_split=0.1,
    callbacks=[hybrid_tb_callback],
    verbose=1
)
```

```
Epoch 1/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 11.9308 - val_loss: 0.2683
Epoch 2/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - loss: 0.0807 - val_loss: 0.1484
Epoch 3/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - loss: 0.0172 - val_loss: 0.1423
Epoch 4/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - loss: 0.0065 - val_loss: 0.1368
Epoch 5/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - loss: 0.0060 - val_loss: 0.1319
```

Out[75]: <keras.src.callbacks.history.History at 0x7fd83d337d10>

Evaluation

In [76]:
```
# 4.Evaluate Model
from sklearn.metrics import mean_squared_error
import numpy as np

preds = hybrid_model.predict([
    test_df['user'],
    test_df['item'],
    test_df['prep_scaled']
], verbose=0).flatten()

rmse = np.sqrt(mean_squared_error(test_df['AggregatedRating'], preds))
print(f"Simplified Hybrid RMSE: {rmse:.4f}")
```

```
Simplified Hybrid RMSE: 0.3440
```

RMSE measures the average difference between the actual ratings and the model's predicted ratings. The value 0.3757 means that on average, the model's predictions are off by about 0.38 rating points on a 1–5 scale. This means the model is accurately predicting user preferences and generalizing well to unseen data

In [78]:
```
!pip install numpy --upgrade --force-reinstall
```

```
Collecting numpy
  Using cached numpy-2.2.6-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.wh
l.metadata (62 kB)
Using cached numpy-2.2.6-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(16.8 MB)
Installing collected packages: numpy
ERROR: pip's dependency resolver does not currently take into account all the packag
es that are installed. This behaviour is the source of the following dependency conf
licts.
tensorflow 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy 2.2.6 which is i
ncompatible.
numba 0.60.0 requires numpy<2.1,>=1.22, but you have numpy 2.2.6 which is incompatib
le.
Successfully installed numpy-2.2.6
```

TensorBoard

In [79]:
```
%tensorboard --logdir logs
```

Based on the TensorBoard visualizations:

1. Loss and Validation Loss (Scalars Tab)

Both models show steady loss reduction over epochs. The hybrid model converged faster and achieved a lower validation loss, indicating better learning and generalization. The deep learning model had a higher starting loss, and although it improved, it remained less optimal compared to the hybrid. Conclusion: The hybrid model is more accurate and stable on unseen data.

2. Bias Histogram

- Deep Learning Model:

Biases are positively skewed, many around 1.9, up to 4. Indicates strong internal preferences (possibly overfitting).

- Hybrid Model:

Bias values are clustered near 0.15, with a narrower spread. Suggests a more balanced model, leveraging both embeddings and additional features (like prep time). Conclusion: The hybrid model has more controlled and interpretable bias adjustments, likely leading to better generalization.

CONCLUSION

Final Report Summary

Four recommendation models were implemented and evaluated: collaborative filtering (SVD), content-based filtering, a deep learning model, and a hybrid model. Performance was measured using RMSE, MAE, and Precision@5, along with TensorBoard visualizations for training behavior.

Collaborative Filtering (SVD) achieved an RMSE of 1.9881 and MAE of 1.6032, indicating moderate prediction error and suggesting room for improvement through tuning or incorporating side features.

Content-Based Filtering showed good precision with a Precision@5 of 0.6971, meaning it was effective at recommending relevant items within the top 5 results, though limited by a lack of collaborative signals.

The Deep Learning Model (matrix factorization-based) performed less effectively with an RMSE of 2.1843, likely due to overfitting or insufficient feature diversity.

The Hybrid Model, combining user-item embeddings with content features (like preparation time), significantly outperformed all others with an RMSE of 0.3757. TensorBoard confirmed better generalization, smooth loss convergence, and a balanced parameter distribution.

Conclusion: **The Hybrid Model** proved to be the most robust and accurate, leveraging the strengths of both collaborative and content-based methods for optimal recommendation performance.