DATA SCIENCE

RECIPE PROJECT

BY DANIEL SHAPIRA & DANIEL PAPKOV

CONTENT

INTRODUCTION
RESEARCH QUESTIONS

Question 1:

What factors contribute to the popularity of a recipe, and how accurately can we predict a recipe's popularity based on these factors?

Question 2 :

Is it possible to determine if a recipe will receive a high rating?

SCRAPING

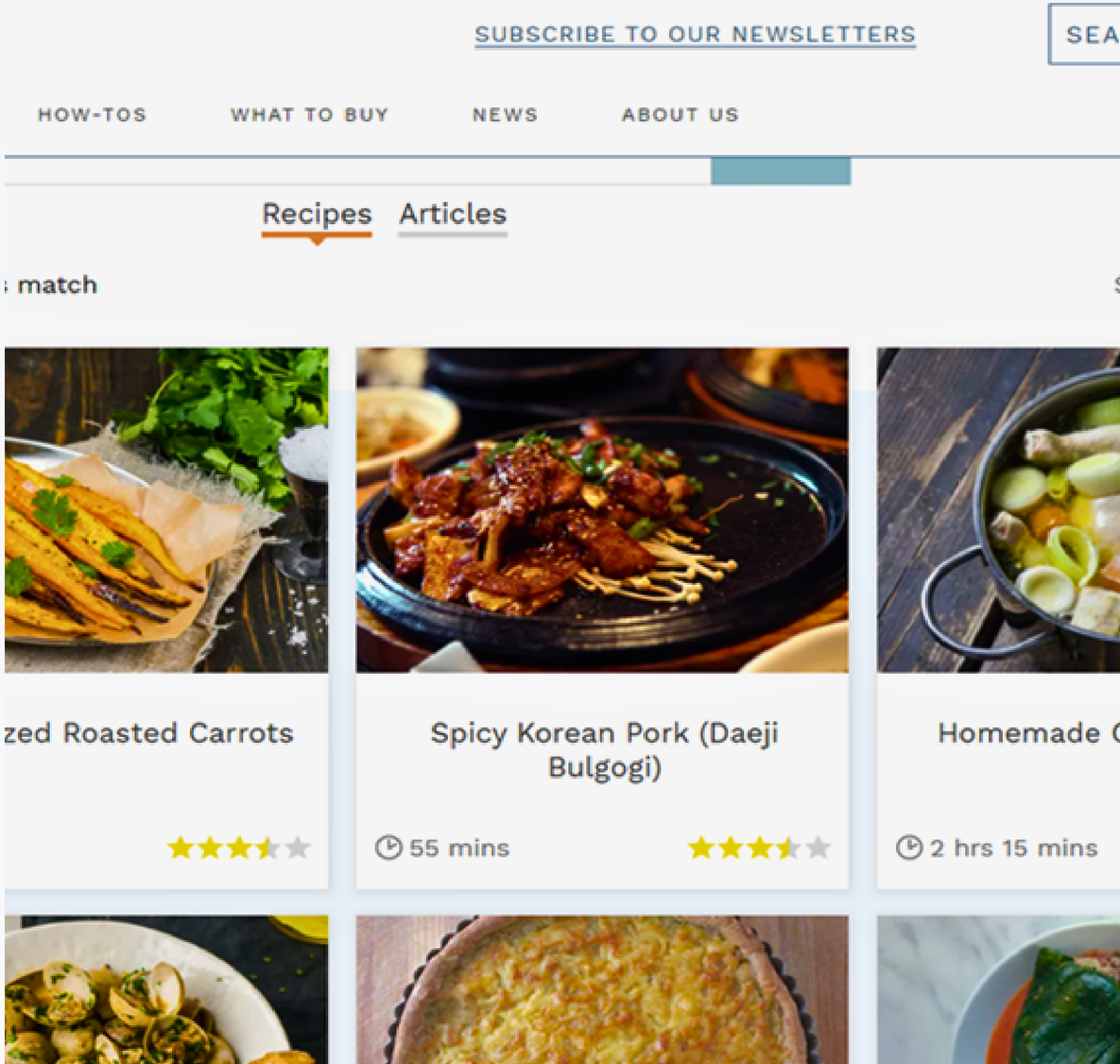
Obtaining necessary data was challenging due to limited access and inconsistent website layout.

Scraping the data into a dataframe required debugging and the use of selenium. Daily site updates posed difficulties in testing specific recipes.

code example

```python
def get_full_page_thespruceeats():#this returns a list of all the links to receipies on the page:
    # URL to scrape
    url = "https://www.thespruceeats.com/search?q=&searchType=recipe"

    # Configure the Selenium webdriver
    options = webdriver.ChromeOptions()
    options.add_argument('--headless')  # Run in headless mode (no GUI)
    driver = webdriver.Chrome(options=options)
    driver.get(url)

    # Wait for the page to load
    wait = WebDriverWait(driver, 10)

    # Get the page source and parse it with BeautifulSoup
    page_source = driver.page_source
    soup = BeautifulSoup(page_source, "html.parser")

    results_div = soup.find("div", attrs={"class": "results-list__container"})
    recipe_names = []
    recipe_links = []

    # Scrape the first page
    for li in results_div.find_all("li", class_="results__item"):
        if li.find("a") is not None:
            link = li.find("a").get("href")
        else:
            link = ''

        if li.find("h4", class_="card__title") is not None:
            name = li.find("h4", class_="card__title").text.strip()
        else:
            name = ''

        recipe_names.append(name)
        recipe_links.append(link)

    # Scrape subsequent pages if the "Next" button exists
    while True:
        try:
            next_button = wait.until(EC.element_to_be_clickable((By.CSS_SELECTOR, ".pagination__item-link--next")))
            next_button.click()
            time.sleep(5)

            # Get the page source and parse it with BeautifulSoup
            page_source = driver.page_source
            soup = BeautifulSoup(page_source, "html.parser")

            results_div = soup.find("div", attrs={"class": "results-list__container"})

            # Scrape recipe names and links
            for li in results_div.find_all("li", class_="results__item"):
                if li.find("a") is not None:
                    link = li.find("a").get("href")
                else:
                    link = ''

                if li.find("h4", class_="card__title") is not None:
                    name = li.find("h4", class_="card__title").text.strip()
                else:
                    name = ''
                print(f"{name} added")
                recipe_names.append(name)
                recipe_links.append(link)

        except:
            break

    # Create a DataFrame with the recipe names and links
    df = pd.DataFrame({"Recipe_name": recipe_names, "Recipe_link": recipe_links})

    # Write DataFrame to a CSV file
    df.to_csv("Recipe_Links_and_Names.csv", index=False)

    # Close the driver
    driver.quit()

    print("Done!")
```

the main site example



code example

```python
def get_cook_times(soup_obj):
    lines=[]
    results_items = soup_obj
    results_items = soup_obj.find_all(class_='comp article__decision-block mntl-block')
    if(results_items==[]):
        soup = soup_obj
        results_items = soup.find_all(class_='comp project-meta')

    for item in results_items:
        item.find_all(class_='meta-text__data')
        for sub_item in item:
            if bool(sub_item.text.strip()):
                clean_text = sub_item.text.strip().replace('\n', '')
                lines.append(clean_text)

    if(len(lines)>1):
        new_string = lines[0] + lines[1]
        lines[0]= new_string

    #use regex expressions to clean up the line we get it looks something like this
    #['Prep: 15 minsCook: 20 minsTotal: 35 minsServings: 6 servingsYield: 1 cake', 'ratingsAdd a comment']
    cook_time_str = re.findall(r'Cook:\s*(?:(\d+)\s*(?:hrs?|hours?)\s*)?(?:(\d+)\s*mins?)?', lines[0])[0]
    prep_time_str = re.findall(r'Prep:\s*(?:(\d+)\s*(?:hrs?|hours?)\s*)?(?:(\d+)\s*mins?)?', lines[0])[0]
    total_time_str = re.findall(r'Total:\s*(?:(\d+)\s*(?:hrs?|hours?)\s*)?(?:(\d+)\s*mins?)?', lines[0])[0]
    # Convert time to minutes


    hours = int(cook_time_str[0]) if cook_time_str[0] else 0
    minutes = int(cook_time_str[1]) if cook_time_str[1] else 0
    cook_time_minutes = hours * 60 + minutes

    hours = int(prep_time_str[0]) if prep_time_str[0] else 0
    minutes = int(prep_time_str[1]) if prep_time_str[1] else 0
    prep_time_minutes = hours * 60 + minutes


    hours = int(total_time_str[0]) if total_time_str[0] else 0
    minutes = int(total_time_str[1]) if total_time_str[1] else 0
    total_minutes = hours * 60 + minutes
    #sometimes instead of saying servings 6 they say servings 6 to 8 in this case we make it 6
    #example of what text might look like
    #text = "The serving size is 3 servings per container."

    if(lines[0].count('serv')):
        match = re.search(r'serv\w*:\D*(\d+)', lines[0], re.IGNORECASE)
        if match:
            servings=(match.group(1))
    else:
        servings=1
```

```python
def get_stars(soup_obj):
    soup = soup_obj
    results_items = soup.find_all(class_='comp js-feedback-trigger aggregate-star-rating mntl-block')
    #print(results_items.prettify())
    for item in results_items:##result items size is 1
        text=item.prettify()
        full_stars=text.count('class="active"')
        half_stars=text.count('class="half"')
        return(full_stars+0.5*half_stars)
```

```python
f get_nutritional_values(soup_obj):
  soup = soup_obj
  results_items = soup.find_all(class_='nutrition-info__table--row')

  nutritional_vals=[]


  for item in results_items:
      nutritional_vals.append(item.text.strip())
  new_list = []
  for s in nutritional_vals:
      # Split the string by the \n character and add the two parts to a new list
      parts = s.split('\n')
      # Add the new strings to the new list in the desired format
      #[caleories:934,fat:134g,carbs:999]
      new_list.extend([parts[1], parts[0]])
  #[calories,934,far,134g,carbs,1123,]
  df = pd.DataFrame({'nutrient': new_list[::2], 'value': new_list[1::2]})

  # Set 'nutrient' column as index and transpose DataFrame
  df = df.set_index('nutrient').T
  if df.empty: #recepies like cocktails have no calories

      df = pd.DataFrame(columns=['nutrient', 'Calories', 'Fat', 'Carbs', 'Protein'])

      # add a row filled with zeros
      df.loc[0] = ['value', 0, '0g', '0g', '0g']
      df['Calories'] = df['Calories'].astype(np.int32)
      return df

  df['Calories'] = df['Calories'].astype(int)
```

Collard Greens Pasta

By Molly Watson | Updated on 01/18/22

The Spruce / Molly Watson

Prep: 10 mins
Cook: 20 mins
Total: 30 mins
Servings: 4 servings

9 RATINGS

Add a comment

Nutrition Facts (per serving)

337	13g	42g	13g
Calories	Fat	Carbs	Protein

Show Full Nutrition Label

(Nutrition information is calculated using an ingredient database and should be considered an estimate.)

code example

```
def get_ingridients(soup_obj):
    cond=0
    soup = soup_obj
    span_elements = soup.find_all('span', {'data-ingredient-name': 'true'})

    # create an empty list to store the ingredient names
    ingredient_names = []

    # loop over the span elements and extract their text content
    for span in span_elements:
        ingredient_names.append(span.text)
    #print(ingredient_names)
    if(len(ingredient_names)>0):
        return(ingredient_names)
    else:
        cond=0
        soup = soup_obj
        results_items = soup.find_all(class_='structured-ingredients__list text-passage')
                                        #comp ingredient-list simple-list simple-list--bulleted
        #print(results_items)
        if(results_items==[]): #sometimes they like to change the class name
            soup = soup_obj
            results_items = soup.find_all(class_='simple-list__item js-checkbox-trigger ingredient text-passage')
            cond=1
        nutritional_vals=[]
        if(results_items==[]):
            return []

        final_lst=[]

        for item in results_items:
            nutritional_vals.append(item.text.strip())
            #print(item.text.strip())
        if(cond==1):
            return nutritional_vals
        else:
            for i in nutritional_vals:
                my_list = [s.strip() for s in i.split('\n\n\n')]
                final_lst.extend(my_list)
                #print(final_lst)

            return(final_lst)
```

```
def analyze_recipe(ingredients):
    dairy_keywords = ["milk", "cheese", "yogurt", "cream", "butter", "whey", "casein", "curds"]
    meat_keywords = ["beef", "chicken", "pork", "lamb", "turkey", "venison", "duck", "bacon", "sausage",
                     "ham", "prosciutto", "pepperoni", "salami", "chorizo", "bresaola", "pastrami",
                     "corned beef", "veal", "goose", "game", "elk", "bison", "rabbit", "boar", "guinea fowl", "quai

    categories = {'Dairy': 0, 'Meat': 0, 'Fur': 1}

    for ingredient in ingredients:
        ingredient = ingredient.lower()
        if any(keyword in ingredient for keyword in dairy_keywords):
            categories['Dairy'] = 1
            categories['Fur'] = 0
        elif any(keyword in ingredient for keyword in meat_keywords):
            categories['Meat'] = 1
            categories['Fur'] = 0

    df = pd.DataFrame(categories, index=[0])
    return df
```

the main site example

Ingredients

- 1 bunch cooking greens, such as kale, lacinato kale, collard greens, turnip greens, chard
- 1 tablespoon fine sea salt, plus more to taste
- 1 pound spaghetti, or linguini
- 2 anchovy fillets, optional
- 4 cloves garlic
- 2 tablespoons olive oil
- 1/4 to 1/2 teaspoon red pepper flakes, optional
- 3/4 cup freshly shredded grating cheese, such as Parmesan, Asiago, or aged pecorino

ADD TO SHOPPING LIST

Save recipes, create shopping lists, meal plans and more.

INITIAL DATA ANALYSIS

- Data Cleaning
- Outlier Detection (Histogram Analysis)
- Data Cleaning Method (IQR and Upper Limit Adjustment)
- Handling Strings Attached to Numbers
- Standardizing Units of Measure
- Adjusting Recipe Timings (where possible)

CLEANING

General Cleaning With IQR

Code Example

```
def clean(df,column_name):
    #print the data we had before so we can later compare
    print(df[column_name].describe())
    # Calculate the IQR of the Prep column
    Q1 = df[column_name].quantile(0.25)
    Q3 = df[column_name].quantile(0.75)
    IQR = Q3 - Q1

    # Determine the upper and lower bounds for the Prep column
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 10 * IQR

    # Replace any values outside of the bounds with NaN
    df.loc[(df[column_name] < lower_bound) | (df[column_name] > upper_bound), column_name] = float('NaN')
    df.dropna(inplace=True)
    #print the data after cleaning for comparison
    print(df[column_name].describe())
    return df
```

```
def clean_df(df):
    #print(df)
    numeric_cols = df.select_dtypes(include=['int', 'float']).columns
    for col in numeric_cols:
        clean(df,col)

    return df
```

A Peak at The Data Frame

The Results

	Name	Prep	Cook	Total	Servings	Rating	Rating_Count	Dairy	Meat	Fur	Calories	Fat	Carbs	Protein	Ingredients
0	Kentucky Buck Cocktail Recipe	5.0	0.0	5.0	1.0	3.5	6.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	['1 chopped and muddled strawberry', '1-ounce ...
1	Sparkling Borage Cocktail	3.0	0.0	483.0	1.0	5.0	6.0	0.0	0.0	1.0	194.0	0.0	19.0	0.0	['water', 'granulated sugar', 'borage leaves'...
2	Lunch Box-Worthy Falafel Kebabs	25.0	30.0	70.0	4.0	4.5	12.0	0.0	0.0	1.0	494.0	17.0	73.0	18.0	['For the Falafel:', '1 cup canned chickpeas, ...
3	Chipotle Pumpkin Queso Dip	5.0	4.0	9.0	6.0	5.0	7.0	1.0	0.0	0.0	247.0	17.0	11.0	13.0	['Velveeta cheese', 'diced tomatoes and green ...
4	Flamin' Hot Cheetos Mac and Cheese Bites	45.0	25.0	280.0	10.0	4.0	57.0	1.0	0.0	0.0	327.0	21.0	28.0	6.0	['macaroni and cheese', 'hot sauce', "Flamin' ...

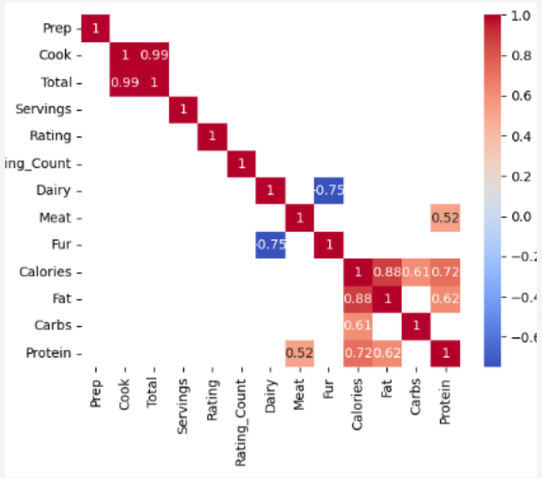
```
Name: Steps, dtype: float64
count    12831.000000
mean         8.279362
std          4.012243
min          1.000000
25%          5.000000
50%          7.000000
75%         10.000000
max         43.000000
Name: Steps, dtype: float64
starting ingridient cleanup this might take a minute
done!
DONE!
```

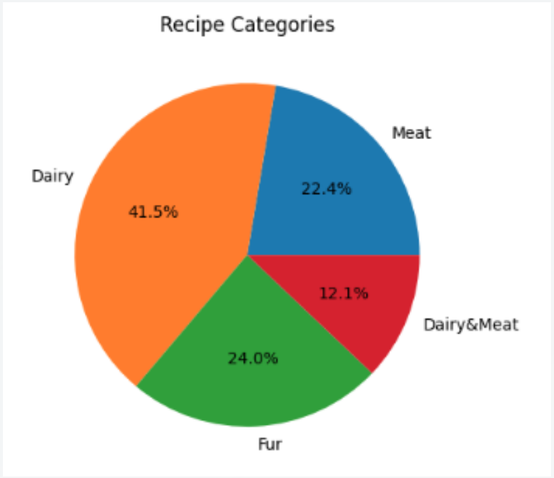
VISUALIZATION AND EDA

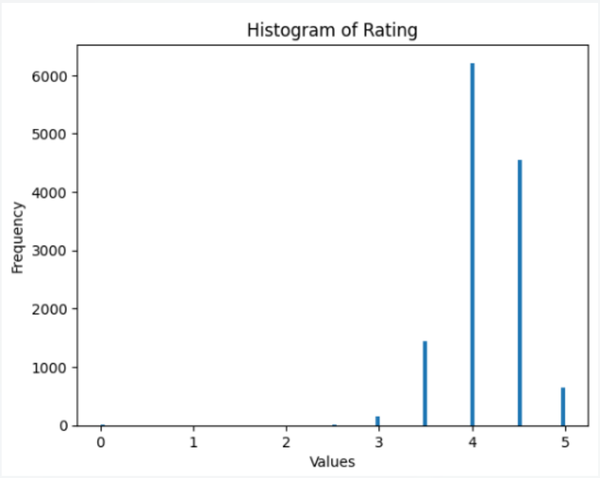
Based on the histogram of calories, we can observe that the most common calorie range for recipes is around 180 calories, which is the average value among approximately 700 recipes Additionally, a significant portion of the recipes falls within the range of 0 to 500 calories.

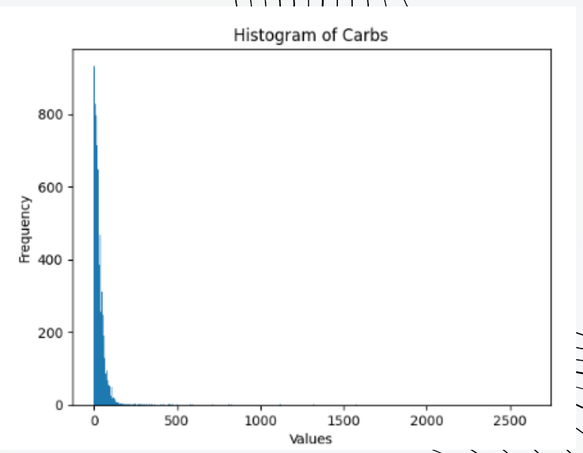
Similarly, for the histogram of carbohydrates, we observe a similar trend. The average recipe tends to contain carbohydrates within the range of 0 to 50 grams.

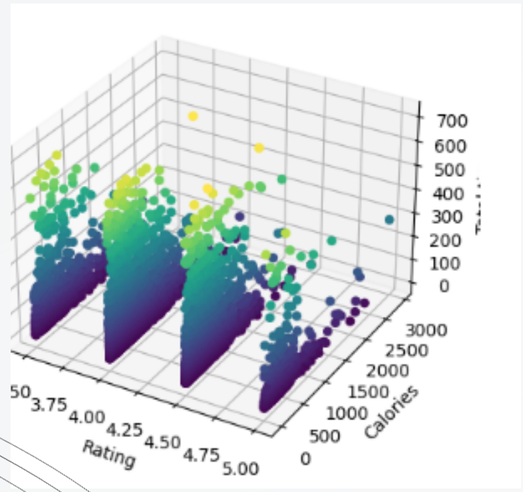
Data After Ccleanning

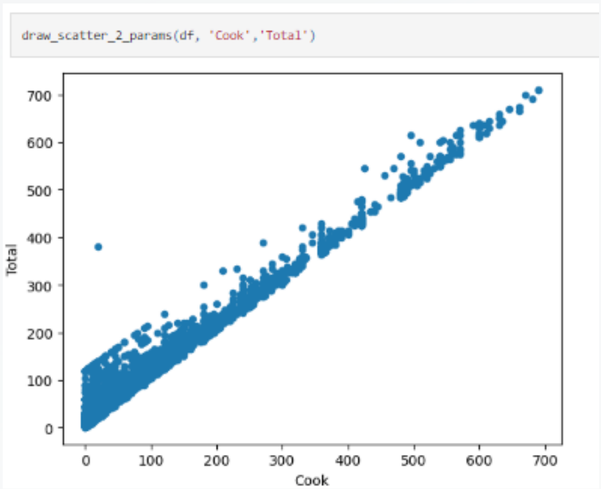


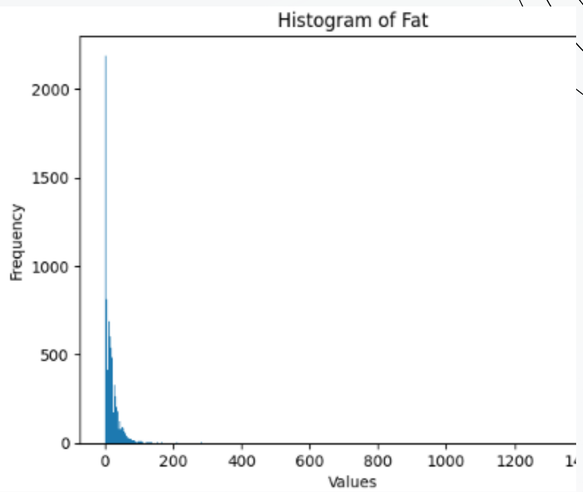












Initial analysis highlighted the need to remove unnecessary information from ingredients and standardize names.

ADVANCED DATA ANALYSYS

To clean the ingredient data, a dictionary was created to track word frequency and remove duplicates and unnecessary information. However, this process had minimal impact on the correlations, with changes observed in less than 0.1% of cases.

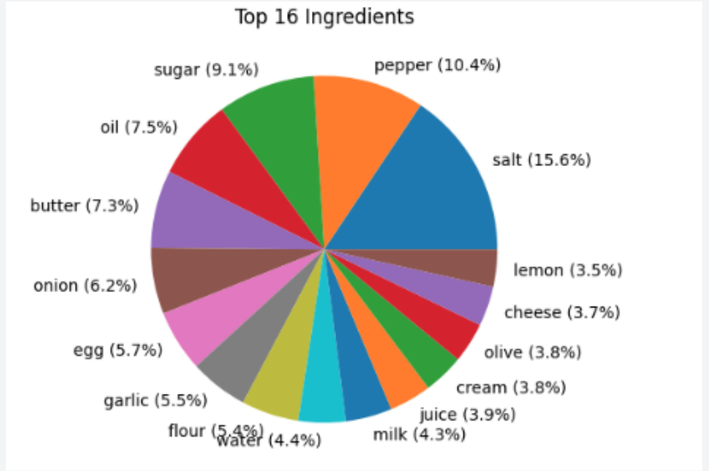
Code Example - Cleaning Ingridients

```python
def remove_teaspoon(df, column):
  df[column] = df[column].str.replace('teaspoon', '')
  df=delete_index_columns(df)
  return df
```

```python
def clean_ingridients(df):
    df2=df.copy()
    pure_ingredients=read_ingridients()
    print("starting ingridient cleanup this might take a minute")
    #df=pd.read_csv(filename)
    for i, ingredient in enumerate(df2['Ingredients']):
        line = ingredient.split(",")
        for j, item in enumerate(line):
            for single_ingredient in pure_ingredients:
                if single_ingredient.lower() in item.lower():
                    #print(f'found {single_ingredient} in {item}')
                    line[j] = single_ingredient.lower()
        df2.loc[i, 'Ingredients'] = ",".join(line)
        #print(line)
        #print('\n\n')
    #print(df['Ingredients'])
    # save the modified dataframe to a new CSV file
    df2=delete_index_columns(df)
    #df2.to_csv('clean_modified.csv', index=False)
    print("done!")

    return df2
```

```python
Sorted_dict = get_ingridients_dict(df)
print("Printing the ingredient Counts:")
for ing, count in Sorted_dict.items():
  print(f'{ing}: {count}')
```

The Result



```
Printing the ingredient Counts:
salt: 10081
pepper: 6712
sugar: 5872
oil: 4878
butter: 4685
onion: 3982
egg: 3709
garlic: 3528
flour: 3467
water: 2852
milk: 2792
juice: 2543
cream: 2443
olive: 2428
cheese: 2405
lemon: 2241
nut: 2007
vanilla: 1989
chicken: 1796
tomato: 1646
vanilla extract: 1560
vegetable: 1558
corn: 1417
parsley: 1178
cinnamon: 1174
vinegar: 1070
beef: 1042
bread: 964
apple: 947
rice: 944
potato: 929
baking powder: 920
ginger: 898
wine: 895
lime: 858
chocolate: 826
broth: 803
soy sauce: 799
mustard: 774
potatoe: 757
celery: 749
carrot: 748
orange: 745
chili: 712
```

To analyze the correlation between ingredients and recipe ratings, we utilized the "ingredient explosion" technique. This involved expanding ingredients into binary columns, where 1 represents ingredient presence and 0 represents absence. This structured format enables us to explore the relationship between individual ingredients and recipe ratings.

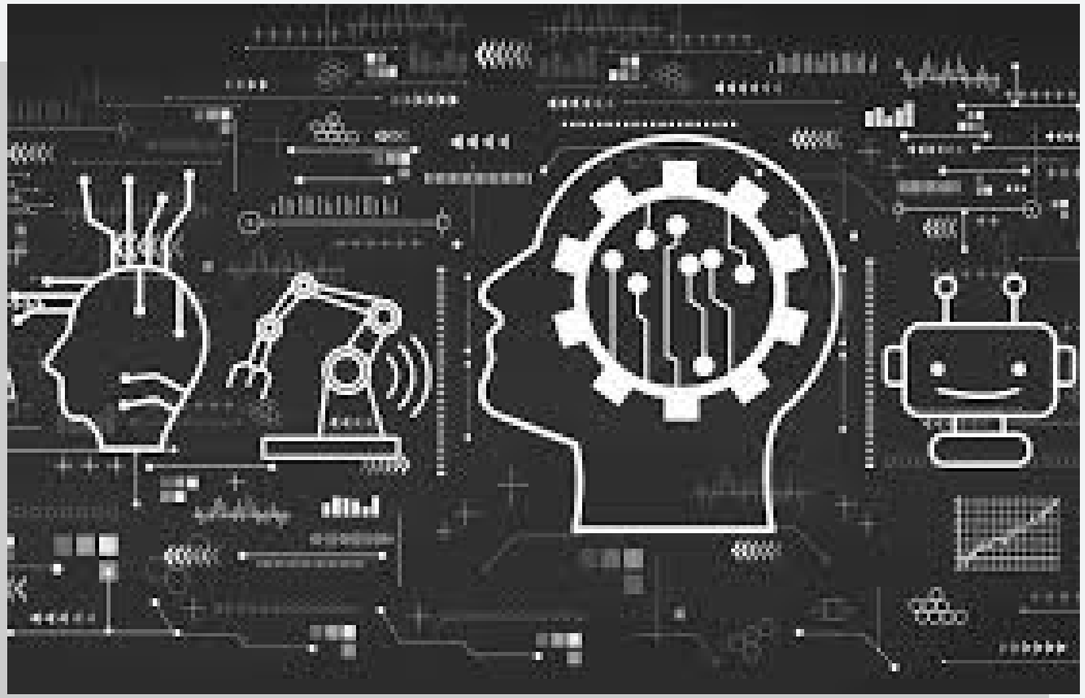
```python
def explode_ingridients_and_get_corr(df):#this function expands the dataframe to each ingridient and tries to find a correl
    #df = pd.read_csv('clean_modified.csv')

    # extract the ingredients column
    df_ing = df['Ingredients']

    # define a list of ingredients you care about
    important_ingredients = []

    with open('ingridients.csv', newline='') as csvfile:
        reader = csv.reader(csvfile)
        for row in reader:
            important_ingredients.extend(row)

    # loop through the ingredients you care about
    for ingredient in important_ingredients:
        # create a new column with a value of 1 if the recipe contains the ingredient and 0 otherwise
        df[ingredient] = df_ing.str.contains(ingredient).astype(int)

    correlations = df[['Rating'] + [ingredient for ingredient in important_ingredients]].corr()

    # print the correlation coefficients for each ingredient
    # print(correlations.loc['Rating'])
    top_20 = correlations.loc['Rating'].sort_values(ascending=False)[1:21]
    print("Top 20 ingredients with the highest correlation with the 'Rating' column:")
    print(top_20)
    return df
```

```python
df= explode_ingridients_and_get_corr(df)
```

```
Top 20 ingredients with the highest correlation with the 'Rating' column:
liqueur       0.062360
juice         0.056769
olive         0.044955
vodka         0.044154
orange        0.039164
apple         0.033292
lemon         0.030181
cheese        0.029343
bacon         0.028237
pepper        0.025770
mayonnaise    0.024706
mayonnais     0.024706
pecan         0.023676
sage          0.023397
pineapple     0.023266
walnut        0.023119
pear          0.023077
sour cream    0.022363
beer          0.021940
beans         0.021857
Name: Rating, dtype: float64
```

MACHINE LEARNING

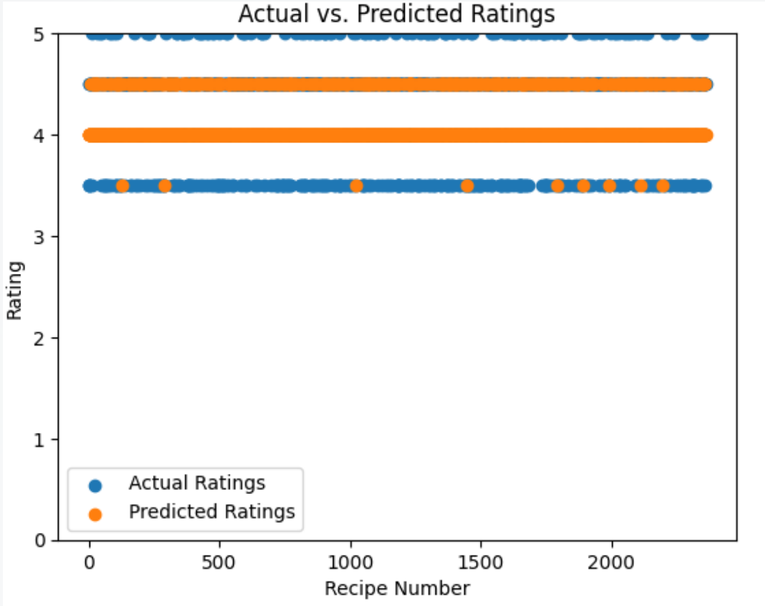
Upon conducting an extensive analysis, we found no significant correlations between individual ingredients and recipe ratings. This suggests that no specific ingredient can directly predict the rating of a recipe. It highlights the intricate nature of determining high-rated recipes, emphasizing the need to consider multiple factors such as preparation methods, cooking techniques, flavor combinations, and presentation when evaluating recipe quality.



To further explore the prediction of ratings, the next step is to employ machine learning. By utilizing machine learning techniques, we can attempt to predict recipe ratings and analyze the weights assigned to various features to draw meaningful conclusions.

The Predictions

Initially, linear regression was employed to predict the star rating, yielding a low accuracy of 0.48. Despite using different and changing the model twice, prediction accuracy remained unchanged.




```
(LinearRegression(),
 4668,
 4783,
 49.39159877261665,
 1155,
 1208,
 48.87854422344478)
```

```
Number of recipes in the dataset: 11814
Number of recipes in the training set: 9451
Number of recipes in the testing set: 2363
Number of correctly predicted recipes in the testing set: 1155
Number of incorrectly predicted recipes in the testing set: 1208
Percent of correctly predicted recipes in the testing set: 48.87854422344478%
```

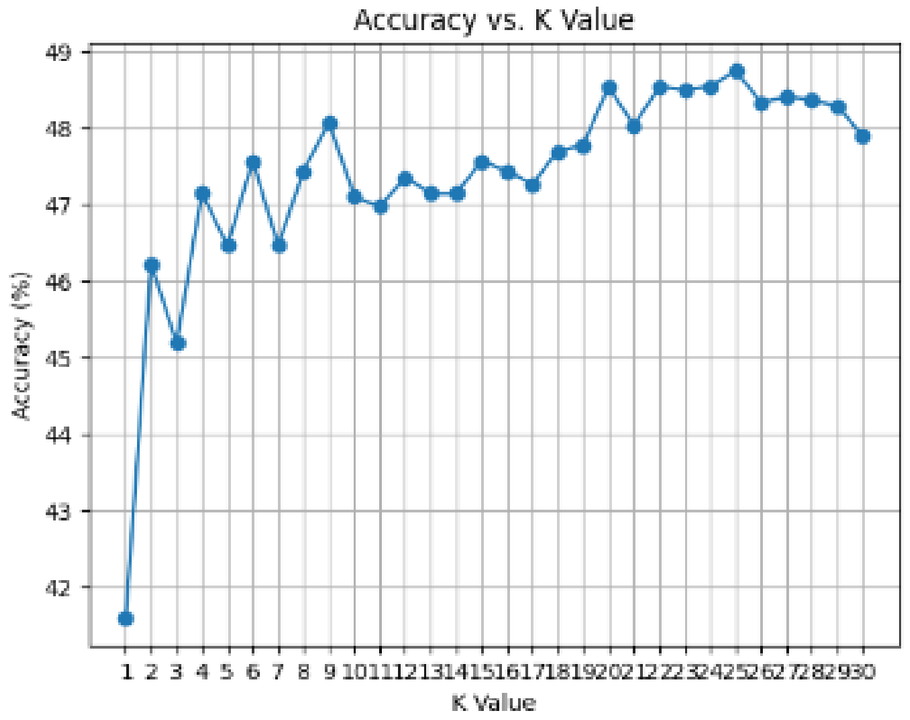
The main code

```python
def predict_rating_linear(df):
    # Prepare X and y
    X = df.drop(['Ingredients', 'Name', 'Rating'], axis=1)
    y = df['Rating']

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Create and fit the model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Round predicted ratings to nearest half-integer value
    def round_half_up(x):
        # Clamp ratings to 0-5 range
        x = np.clip(x, 0, 5)
        # Round to nearest half-integer value
        return np.floor(x * 2 + 0.5) / 2

    # Make predictions on the training set
    y_train_pred = model.predict(X_train)
    y_train_pred_rounded = round_half_up(y_train_pred)

    # Count correct and incorrect predictions in the training set
    num_train_correct = np.sum(np.abs(y_train_pred_rounded - y_train) <= 0.25)
    num_train_incorrect = len(y_train) - num_train_correct
    percent_train_correct = num_train_correct / len(y_train) * 100

    # Make predictions on the testing set
    y_test_pred = model.predict(X_test)
    y_test_pred_rounded = round_half_up(y_test_pred)

    # Count correct and incorrect predictions in the testing set
    num_test_correct = np.sum(np.abs(y_test_pred_rounded - y_test) <= 0.25)
    num_test_incorrect = len(y_test) - num_test_correct
    percent_test_correct = num_test_correct / len(y_test) * 100

    # Print results
    print(f"Number of recipes in the dataset: {len(df)}")
    print(f"Number of recipes in the training set: {len(X_train)}")
    print(f"Number of recipes in the testing set: {len(X_test)}")
    print(f"Number of correctly predicted recipes in the testing set: {num_test_correct}")
    print(f"Number of incorrectly predicted recipes in the testing set: {num_test_incorrect}")
    print(f"Percent of correctly predicted recipes in the testing set: {percent_test_correct}%")

    test_results = np.concatenate((y_test.to_numpy().reshape(-1, 1), y_test_pred_rounded.reshape(-1, 1)), axis=1)
    plt.scatter(range(len(test_results)), test_results[:, 0], label='Actual Ratings')
    plt.scatter(range(len(test_results)), test_results[:, 1], label='Predicted Ratings')
    plt.xlabel('Recipe Number')
    plt.ylabel('Rating')
    plt.ylim(0, 5)  # Set y-axis limits
    plt.title('Actual vs. Predicted Ratings')
    plt.legend()
    plt.show()

    return model, num_train_correct, num_train_incorrect, percent_train_correct, num_test_correct, num_test_incorrect, perc
```

The Predictions

We will try to use different machine-learning algorithms in order to improve our prediction. The next algorithms we will try to use is KNN.

```
Best k: 25
Best percent of correctly predicted recipes in the testing set: 48.751586965721536%
```




```
Final results:
Number of recipes in the dataset: 11814
Number of recipes in the training set: 9451
```

As we can see we got a correct prediction percentage of 48.75% with a K of 25 as best.

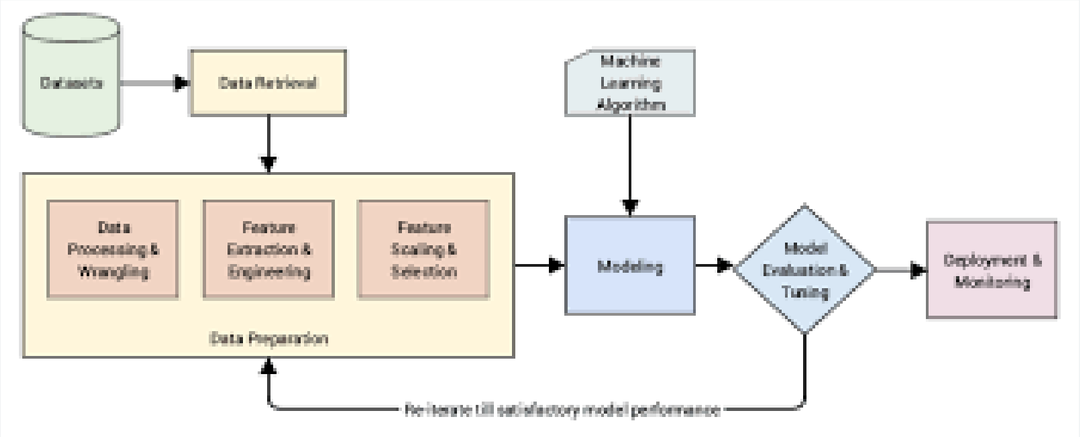
The main code

```python
def predict_rating_knn(df):


    # Prepare X and y
    X = df.drop(['Ingredients', 'Name', 'Rating'], axis=1)
    y = df['Rating']

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Create and fit the model
    best_k = 0
    best_accuracy = 0
    accuracies = []
    for k in range(1, 31):
        model = KNeighborsRegressor(n_neighbors=k)
        model.fit(X_train, y_train)

        # Make predictions on the training set
        y_train_pred = model.predict(X_train)
        y_train_pred_rounded = np.round(np.clip(y_train_pred, 0, 5) * 2) / 2

        # Count correct and incorrect predictions in the training set
        num_train_correct = np.sum(np.abs(y_train_pred_rounded - y_train) <= 0.25)
        percent_train_correct = num_train_correct / len(y_train) * 100

        # Make predictions on the testing set
        y_test_pred = model.predict(X_test)
        y_test_pred_rounded = np.round(np.clip(y_test_pred, 0, 5) * 2) / 2

        # Count correct and incorrect predictions in the testing set
        num_test_correct = np.sum(np.abs(y_test_pred_rounded - y_test) <= 0.25)
        percent_test_correct = num_test_correct / len(y_test) * 100

        # Print results
        print(f"For k={k}:")
        print(f"Number of correctly predicted recipes in the training set: {num_train_correct}")
        print(f"Percent of correctly predicted recipes in the training set: {percent_train_correct}%")
        print(f"Number of correctly predicted recipes in the testing set: {num_test_correct}")
        print(f"Percent of correctly predicted recipes in the testing set: {percent_test_correct}%")

        # Check if the current model is better than the previous best model
        if percent_test_correct > best_accuracy:
            best_k = k
            best_accuracy = percent_test_correct

        accuracies.append(percent_test_correct)

    print(f"\nBest k: {best_k}")
    print(f"Best percent of correctly predicted recipes in the testing set: {best_accuracy}%")

    # Plot the accuracy for each K value
    k_values = range(1, 31)
    plt.plot(k_values, accuracies, marker='o')
    plt.title("Accuracy vs. K Value")
    plt.xlabel("K Value")
    plt.ylabel("Accuracy (%)")
    plt.xticks(k_values)
    plt.grid(True)
    plt.show()

    # Train the best model
    model = KNeighborsRegressor(n_neighbors=best_k)
    model.fit(X_train, y_train)

    # Make predictions on the testing set
    y_test_pred = model.predict(X_test)
    y_test_pred_rounded = np.round(np.clip(y_test_pred, 0, 5) * 2) / 2

    # Count correct and incorrect predictions in the testing set
    num_test_correct = np.sum(np.abs(y_test_pred_rounded - y_test) <= 0.25)
    num_test_incorrect = len(y_test) - num_test_correct
    percent_test_correct = num_test_correct / len(y_test) * 100

    # Print final results
    print("\nFinal results:")
    print(f"Number of recipes in the dataset: {len(df)}")
    print(f"Number of recipes in the training set: {len(X_train)}")
```

FEATURE ENGINEERING

- We encountered challenges with our initial predictions. Firstly, we expanded our data collection efforts by scraping additional information, including the number of ingredients and steps required for each recipe.

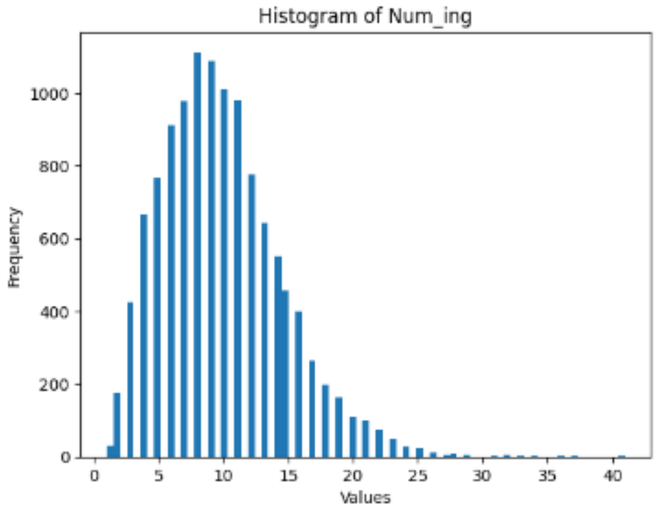
- Furthermore, we performed feature engineering to gain valuable insights. We assigned a popularity score to each ingredient, reflecting its commonality, and developed a metric to evaluate recipe difficulty based on the total number of steps and ingredients involved.

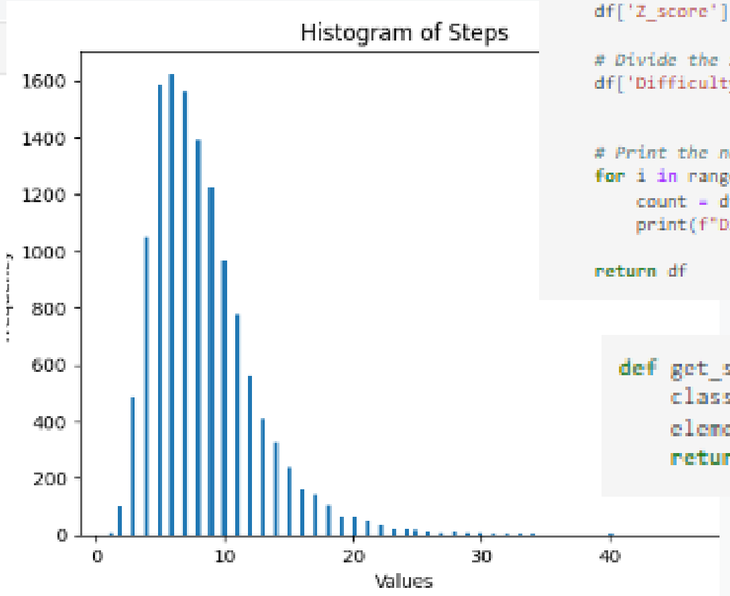


RESCRAPING

We introduced a popularity column based on ingredient frequency to assess recipe uniqueness. Additionally, a difficulty column was created considering steps, cooking time, and ingredient count. These changes necessitate modifying scraping and cleaning functions for the new data.

```
[79]: draw_histo_1_params(Updated_Data,'Num_ing')
```






```python
def add_popularity_score_to_df(df):
    binary_columns = [col for col in df.columns if col not in ['Name', 'Prep', 'Cook', 'Total', 'Servings', 'Rating', 'Rati

    # Calculate the frequencies of each binary value
    ingredient_frequencies = df[binary_columns].sum() / len(df) * 30

    # Calculate the popularity score for each recipe
    popularity_scores = []
    for _, row in df.iterrows():
        score = 0
        for column in binary_columns:
            if row[column] == 1:
                score += ingredient_frequencies[column]
        popularity_scores.append(score)

    # Add the popularity scores as a new column to the DataFrame
    df['Popularity Score'] = popularity_scores

    # Print the updated DataFrame
    #print(df)
    #scraping_functions.draw_histo_1_params(df,'Popularity Score')
    return df
```

```python
def add_difficulty_column(df):
    # Calculate the combined score for each recipe
    df['Combined'] = df['Num_ing'] + df['Steps'] + df['Total']

    # Calculate the mean and standard deviation of the combined score
    mean = df['Combined'].mean()
    std_dev = df['Combined'].std()

    # Calculate the z-score for each recipe
    df['Z_score'] = (df['Combined'] - mean) / std_dev

    # Divide the z-scores into 5 equal-sized groups
    df['Difficulty'] = pd.qcut(df['Z_score'], q=5, labels=[1, 2, 3, 4, 5])


    # Print the number of recipes in each difficulty level
    for i in range(1, 6):
        count = df['Difficulty'][df['Difficulty'] == i].count()
        print(f"Difficulty level {i}: {count}")

    return df
```

```python
def get_steps_to_cook(soup_obj):
    class_name = "comp mntl-sc-block-group--LI mntl-sc-block mntl-sc-block-startgroup"
    elements = soup_obj.find_all(class_=class_name)
    return len(elements)
```

CHANGE OF PLANS

- After thoroughly reevaluating our models with the inclusion of the new data, we were disheartened to find that significant improvements remained elusive. Undeterred, we decided to pivot our approach and made a crucial realization. Recognizing that a good recipe can be considered a 5-star recipe

THE PERCEPTRON MODEL

We shifted our focus toward binary classification. With this in mind, we opted to employ the perceptron model. Leveraging the insights provided by the model's weights, we aimed to shed light on our initial questions.

```
def build_perceptron_model(df):
    #df = pd.read_csv('test123.csv')

    # Save the 'Rating' column separately
    ratings = df['Rating']

    # Drop the 'Rating' column from the dataframe
    df = df.drop(['Rating'], axis=1)
    df = df.drop(['Rating_Count'], axis=1)
    df = df.select_dtypes(include=['int', 'float'])
    # Get the column names of the dataframe
    col_names = df.columns

    # Convert the dataframe to a numpy array
    X = df.select_dtypes(include=['int', 'float']).values

    # Define the target variable
    y = (ratings == 5).astype(int).values

    # Verify that X and y have the same length
    assert len(X) == len(y), "X and y must have the same length"

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)

    # Fit the Perceptron model to the training data
    model = Perceptron()
    model.fit(X_train, y_train)

    # Compute the predictions on the test data
    y_pred = model.predict(X_test)

    # Compute the accuracy of the model
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy:.2f}")

    # Get the weights learned by the perceptron
    weights = model.coef_[0]

    # Create a new dataframe with the column names and weights
    weights_df = pd.DataFrame({'column': col_names, 'weight': weights})

    # Print the weights dataframe
    print(weights_df)

    #save_df(weights_df,'FML.csv')
    return weights_df
```

	column	weight
92	pineapple	129.0
147	almond	129.0
127	vanilla	145.0
110	cinnamon	152.0
207	olive	183.0
224	lime	204.0
152	orange	218.0
204	vodka	218.0
135	liqueur	239.0
200	apple	311.0
197	juice	524.0
7	Fur	596.0
13	Steps	1656.0
12	Num_ing	2380.0
10	Carbs	8242.0

	column	weight
9	Fat	-9089.000000
11	Protein	-8367.000000
0	Unnamed: 0	-8013.000000
241	Popularity Score	-6512.817721
3	Total	-5785.000000
4	Servings	-4315.000000
1	Prep	-3264.000000
2	Cook	-2521.000000
14	Combined	-1749.000000
8	Calories	-1428.000000
206	onion	-410.000000
15	Z_score	-402.488185
209	flour	-320.000000
158	garlic	-232.000000
237	egg	-201.000000

```
weights_data = build_perceptron_model(Updated_Data)
```

```
Accuracy: 0.95
```

CONCLUSIONS

- As we saw by using the perceptron weights as we did, we managed to figure out what are the factors that contribute the most to our recipe which were: the number of ingredients, Steps, and Carbs.

- It possible to determine if a recipe will receive a high rating, with a high degree of accuracy(0.95 in our case) given enough data about the recipe :)

THANK'S FOR WATCHING

Daniel Shapira & Daniel Papkov