Banana Ripening and Taste Experiment

Authors: Chengyu Cui, Liza Kostina, Ashlan Simpson, Daniel Zou

Date: October 2024

Objective

This project is an experiment designed to investigate the effects of two treatments on banana ripeness and taste:

Treatment 1 : Storing bananas in a brown paper bag.

Treatment 2 : Baking bananas for 5 minutes at 300 degrees Fahrenheit.

The primary goal is to determine whether these treatments impact banana ripeness and taste, and if there is any interaction effect between the two treatments.

Hypothesis 1 The paper bag treatment impacts ripeness.

Hypothesis 2 Baking the bananas for 5 minutes at 300 degrees impacts ripeness.

Hypothesis 3 There is an interaction effect between the 2 treatments.

Hypothesis 4 The paper bag treatment does not negatively impact taste.

Hypothesis 5 Baking bananas does not negatively impact taste.

Introduction

Bananas are a basic fruit in many households, and there are several well-known methods to speed up their ripening process. Given the widespread consumption of bananas and their tendency to spoil quickly if not properly managed, many household methods have emerged with the aim of accelerating the ripening process. Techniques such as placing bananas in a brown paper bag or using heat to accelerate ripeness are widely used. Experience tells that these method may accelerate the ripening process of the fruit, however, their exact effect on the ripeness process of the fruit remains unclear as well as on the taste.

In this experiment, we seek to evaluate the effects of two treatments on both the ripeness and taste, as well as the potential relationship of these two outcomes. While we do not have specialized equipment to measure ripeness, we will use Lugol's iodine test, which provides a reliable chemical measure of starch content to indicate ripeness. Additionally, we will gather taste assessments from participants to gain a comprehensive understanding of how these treatments influence both the ripeness and the flavor of bananas.

The rest of the report will be organized as follows. In Section Experimental Design, we present the design of our experiment, with details on the treatment procedure and adjustment for confounders. In Section Data Collection we

present our procedure of data collection and in Section Data Analysis we conduct statistical analysis of the collected data. Section Conclusion finishes our report.

Experimental Design

The experiment is designed with a total of 36 observational units, comprising 9 bunches of 4 bananas each. The bunches are selected to be as uniform as possible based on visual characteristics such as size, ripeness, and overall condition, minimizing variation across bunches at the start of the experiment.

Treatment

Our experiment evaluates two common household techniques for accelerating banana ripening:

(1) placing bananas in a brown paper bag and

(2) baking bananas for five minutes at 300°F.

The four treatment groups are structured as follows:

- Control group (A): Bananas left untreated, exposed only to ambient conditions.
- Paper bag treatment (B): Bananas placed in a brown paper bag.
- Oven treatment (C): Bananas baked for five minutes at 300°F.
- Combined treatment (D): Bananas subjected to both the paper bag and oven treatments.

By having each treatment group represented by one banana per bunch, the design ensures that every treatment has the same representation across the naturally occurring variability in the bunches, aiding in balanced comparison and generalizability.

Apart from the treatment, we controlled for external factors that could potentially influence the ripening process of all bananas, such as temperature, humidity, light and etc. These factors can have a large impact on variation in ripening speed and texture and progression of ripening. By controlling these factors, we ensured that any observed differences in ripening or taste were attributable solely to the experimental conditions under investigation.

Blocking and Covariate Balance

To account for other confounding factors that could influence the ripening process such as size, shape, or the ripeness of bananas within the same bunch at the beginning of our experiment, blocking is incorporated into the design. Each bunch acts as a block, ensuring that any intrinsic similarities within the bunch are distributed equally across all treatment groups. Here, we assume that within the same bunch, these possible confounding factors are more likely to

be similar. This blocking mechanism helps balance covariates such as initial ripeness and size, allowing the experiment to control for these variables when analyzing the treatment effects. By structuring our design this way, we hope that the experiment increases its power to detect differences attributable to the treatments rather than to random variability among bananas.

Post-Treatment Procedure

After each treatment is applied, the bananas are stored under uniform ambient conditions to mimic typical household storage. The bananas in the control group are kept in the same environment without any interventions. Bananas from the paper bag treatment are left in the bags for the duration of the experiment, while those subjected to the oven treatment are baked and then returned to the same ambient conditions as the other groups. This standardization ensures that the only differences between bananas are due to the treatments themselves, rather than external factors like storage environment or temperature fluctuations.

Evaluation Timeline

Bananas are evaluated daily to track the progress of ripening and changes in taste. Each day, a bunch is selected randomly, and the four bananas from that bunch—representing the four treatment groups—are evaluated for both ripeness and taste. The evaluations follow a standardized procedure: participants first assess the taste of the top half of the banana, then assess the ripeness of the bottom half, including an examination of the peel. Finally, ripeness is chemically measured using Lugol's iodine test, which quantifies starch content as an objective indicator of ripeness.

This daily assessment continues for a set number of days, ensuring that any differences in the rate of ripening or taste deterioration are captured throughout the experiment. By staggering the evaluations across different bunches, the experiment ensures that ripening trends can be observed consistently while avoiding any confounding effects from over- or under-ripening in specific bananas.

Data Collection

Experimental Measures

Data for this experiment was collected daily over the course of the banana ripening process and the raw data are available in the corresponding day folder. Each day, one banana from each treatment group (control, paper bag, baking, paper bag + baking) was selected from the corresponding bunch, and data was recorded for the following measures:

Color The color of bananas is a key visual indicator of ripeness, evolving from green to yellow and eventually developing brown spots. These changes in color can influence participants' perceptions of ripeness and taste. To capture

this effect, participants evaluated the color of the bananas subjectively using a standardized scale 1-10. Their responses were used in further analysis.

Taste Taste was evaluated subjectively by participants, who rated the bananas on sweetness, texture, and overall appeal using a standardized scale 1-10. Since ripeness can significantly affect taste, this measure were used to assess how the different ripening treatments (paper bag, baking, or both) influence the flavor of the bananas compared to the control.

Lugol's Iodine Test Lugol's iodine solution was used to measure the starch content of the bananas, providing an objective assessment of ripeness. As bananas ripen, starch is converted to sugar, resulting in less intense staining when treated with iodine. Bananas that are less ripe must show darker staining, indicating higher starch levels. This test was supposed to give us a precise, chemical measure of ripeness, complementing the subjective evaluations of color and taste.

Data Analysis

The data collected each day was uploaded to the `data/` folder in the project repository. This includes participant responses on ripeness and taste, as well as the results of the Lugol's iodine test for ripeness. These daily uploads should provide a comprehensive record of the banana ripening process and allow for real-time analysis and tracking of patterns in the data.

EDA

Before diving into formal hypothesis testing, we conducted an Exploratory Data Analysis to understand the characteristics of our data collected over six days. This initial exploration was crucial for examining both ripeness and taste measurements across the different treatment groups. Using various visualizations, including side-by-side box plots for ripeness and taste distributions by day and group, as well as trend lines for mean ripeness and taste across groups, we aimed to identify patterns, trends, and any irregularities. This process provided an overview of how ripeness and taste evolved across days, highlighted differences among groups, and helped us verify key assumptions about distribution and variability that could influence further statistical analyses.

Note: It's important to note that the results of Lugol's iodine test were inconclusive. The test produced nearly identical results across different treatments on the same day, suggesting that it may not serve as a reliable, objective measure of ripeness in this context (see Figure 1 below). Consequently, we opted to exclude these results from further analysis. Suggestions for possible modifications to improve the test's reliability are provided in the appendix.

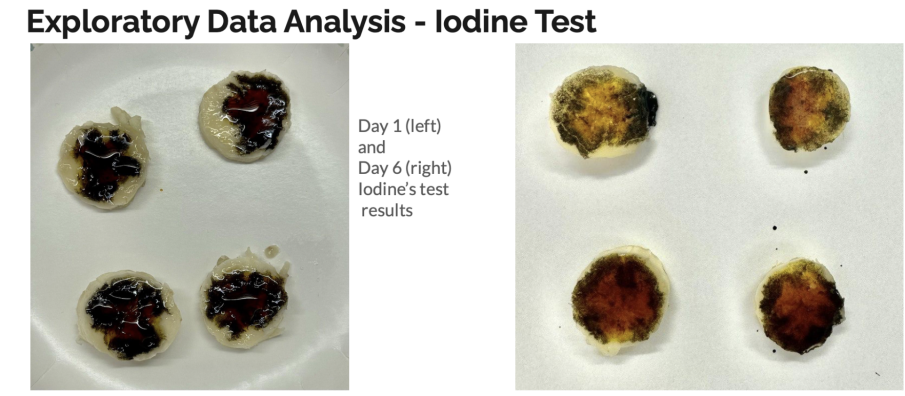

Figure 1: png

Figure 1: Results of Lugol's Iodine test for day 1 and day 6.

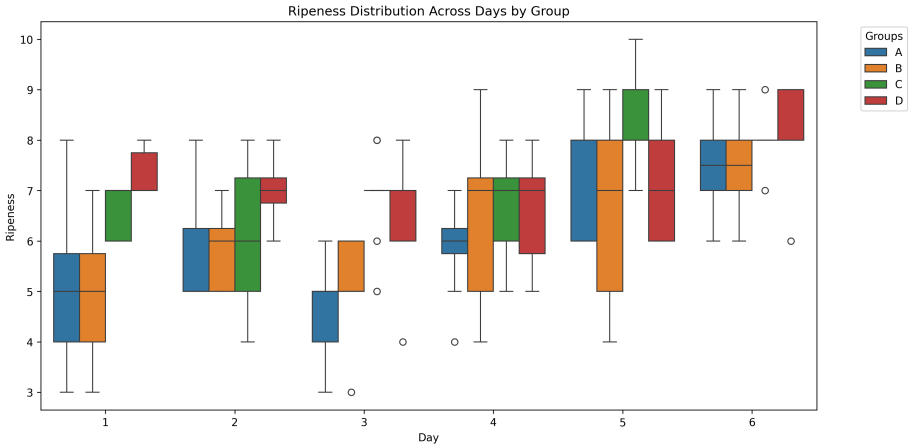

Figure 2: png

Figure 2: Ripeness Distribution across days by group.

Figure 3: Mean Ripeness across days by group.

The plot (see Figure 2) shows the distribution of ripeness ratings for bananas across six days, divided into four groups: **A (Control)**, **B (Paper Bag)**, **C (Baking)**, and **D (Baking + Paper Bag)**. Each box plot represents the ripeness scores as rated by volunteers, with the central line showing the median,

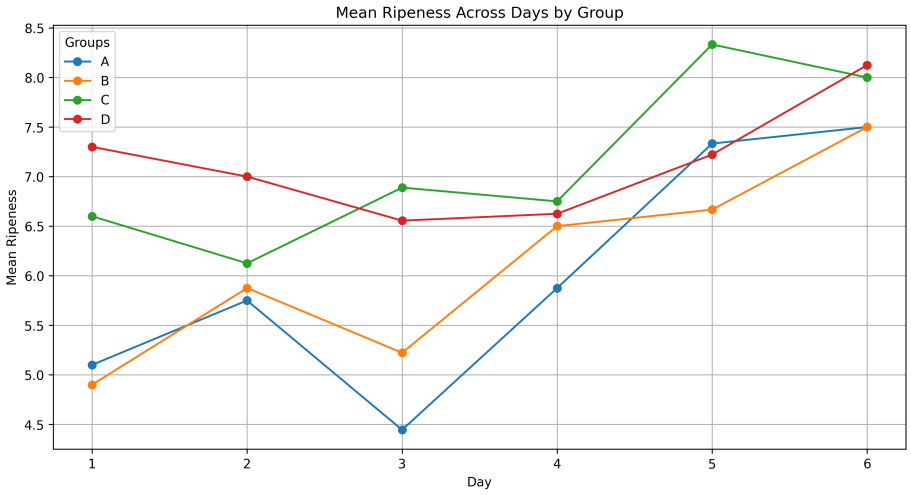

Figure 3: png

the box indicating the interquartile range, and the whiskers covering the range of non-outlier data.

The number of responses per day varied, with some days having as many as 12 responses and others as few as 8. Day 3 had the fewest responses, which could explain the narrower distribution observed for that day compared to others.

The visualizations reveal differences in ripeness levels among the four treatment groups and across the days, which could indicate changes in ripeness influenced by the treatment conditions over time. This representation helps illustrate how each method (control, paper bag, baking, and baking + paper bag) might affect the ripening process across multiple days of observation.

To focus more on the average responses, we created an additional visualization (see Figure 3). In this figure, we don't display the variability within each day for each group; instead, we summarize the responses by computing the mean. This representation shows an overall increase in ripeness over time, as expected. Additionally, the mean values for each group tend to converge over time, with days 5 and 6 showing closer mean ripeness levels compared to day 1.

We continued our analysis by creating two plots (see Figure 4 and Figure 5) that summarize the previous results. In Mean Ripeness Across Days: Group A vs. Mean of Other Groups, we display box plots for the control group (Group A) and the combined responses of all treatment groups. This visualization highlights the overall ripeness trend more clearly, with a noticeable decrease in variability within the combined treatment group.

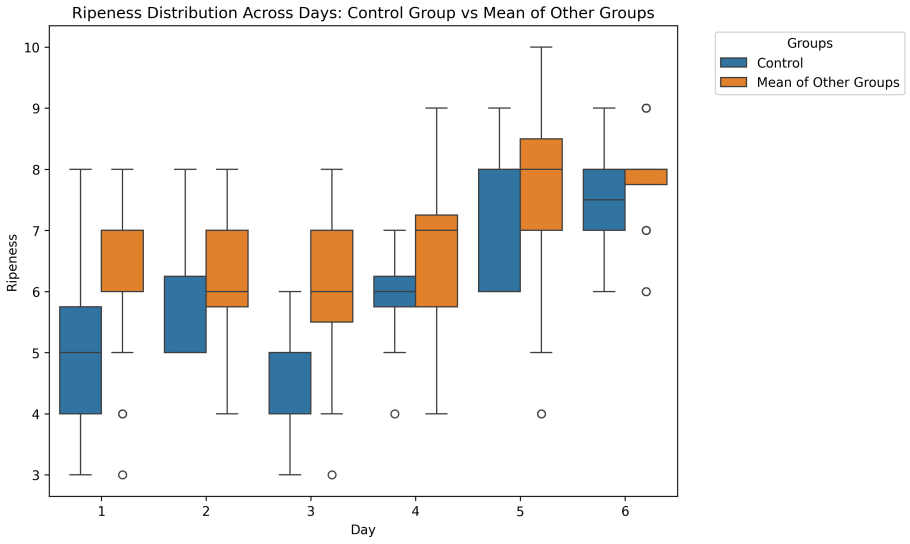

Figure 4: png

Figure 4: Ripeness Distribution Across Days: Control Group vs Mean of Other Groups.

Figure 5: Mean Ripeness Across Days: Control Group vs Mean of Other Groups.

We extended our analysis by creating similar visualizations for taste (see Figure 6- Figure 9). This allowed us to examine trends in taste ratings across days and to compare the control group with the combined treatment groups, providing additional insights into how the different treatments influenced participants' taste perceptions over time.

Figure 6: Mean Taste Across Days by Group.

Figure 7: Mean Taste Across Days by Group (Control and Combined Treatment).

Figure 8: Taste Distribution Across Days by Group.

Figure 9: Taste Distribution Across Days by Group (Control and Combined Treatment).

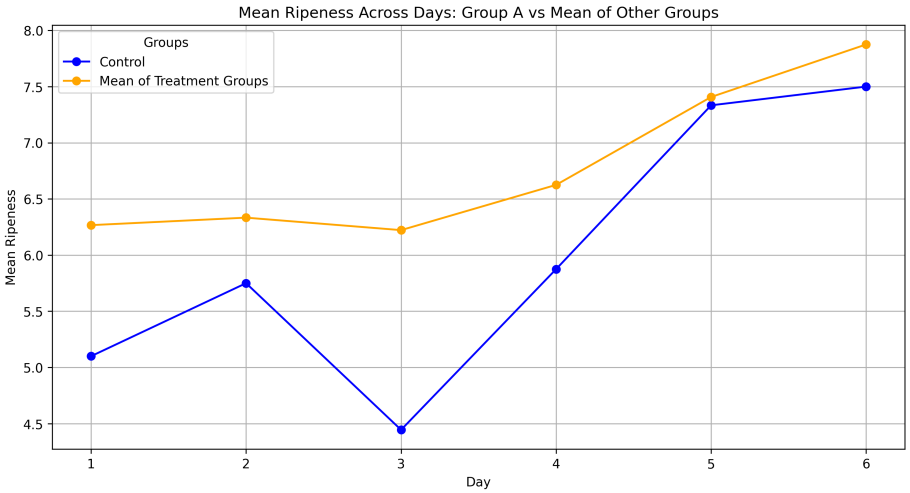

Figure 5: png

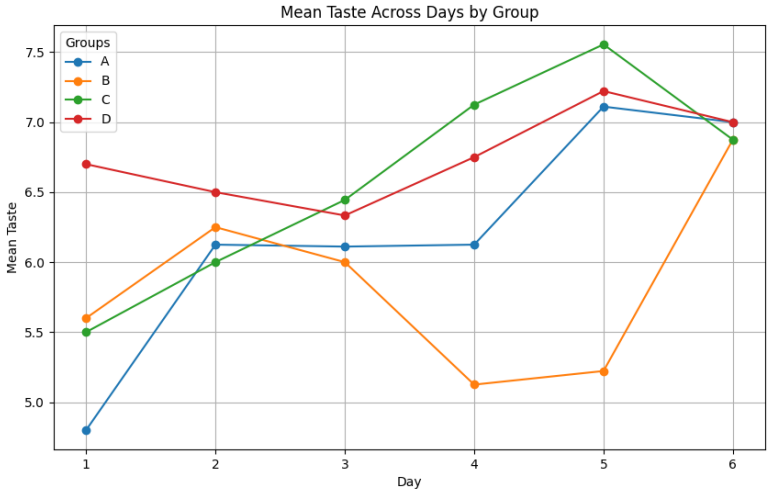

Figure 6: png

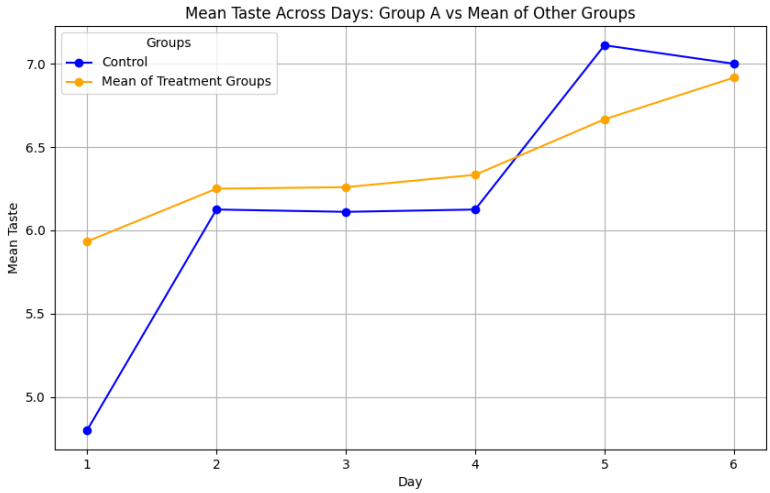

Figure 7: png

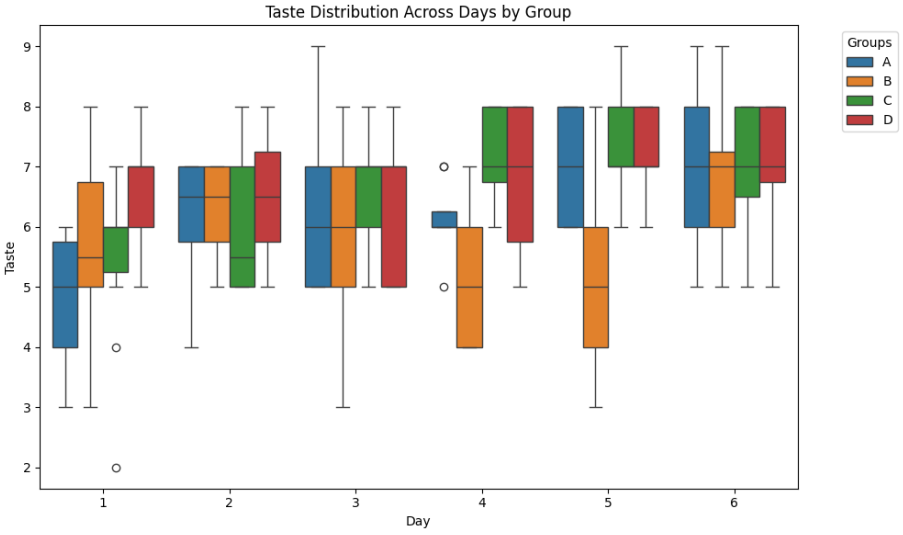

Figure 8: png

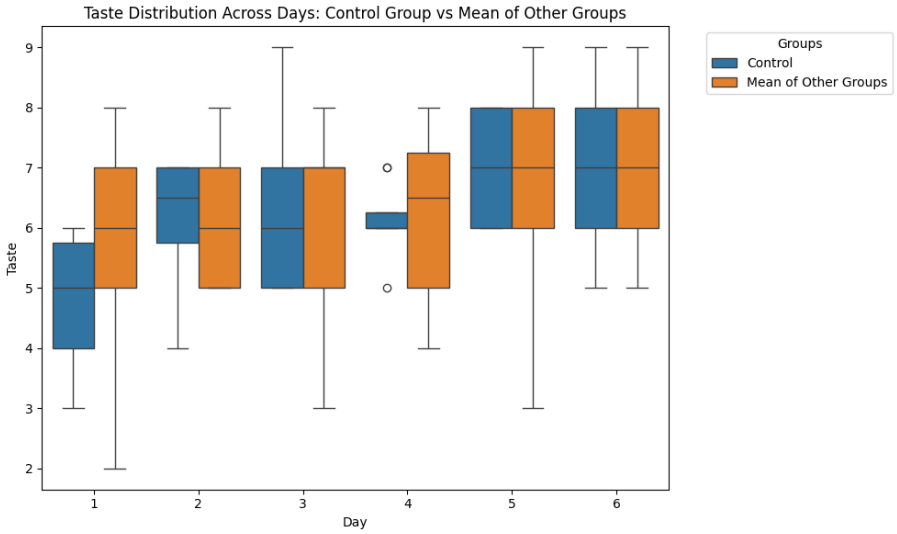

Figure 9: png

We proceeded with our analysis by conducting hypothesis testing. To test the first two hypotheses, we began by checking normality assumptions using the Shapiro-Wilk test. For cases where normality assumptions were met, we applied a t-test, and for those that did not meet the assumptions, we used the Mann-Whitney U test. The results indicated that this approach may not be optimal. For Hypothesis 1, we found no evidence to reject the null hypothesis. For Hypothesis 2, we observed small p-values on only two out of the six days. Based on these findings, we decided to proceed with permutation tests for further analysis.

Simple hypothesis testing

Hypothesis 1: The paper bag treatment impacts ripeness.

H_0 : The bag treatment does NOT impact ripeness (i.e., any difference in ripeness between groups is due to chance).

H_1 : The bag treatment does impact ripeness (i.e., the difference in ripeness between groups is statistically significant).

See code and results in Appendix.

Hypothesis 2: Baking the bananas for 5 minutes at 300 degrees impacts ripeness.

H_0 : The baking treatment does NOT impact ripeness (i.e., any difference in ripeness between groups is due to chance).

H_1 : The baking treatment does impact ripeness (i.e., the difference in ripeness between groups is statistically significant).

See code and results in Appendix.

Inference on Treatment Effect

In this section we present the inference results on the treatment effect by running a rank permutation test and a propensity score matching. These methods have been commonly used in causal discovery.

Since we employed blocking by testing bananas from the same bunch every day, we applied a blocked permutation test, in which treatments were permuted within the four bananas of a single bunch. This approach allowed us to control for variation within each bunch and assess the treatment effects more reliably. Specifically, we used a rank-based permutation test, a non-parametric method that compares groups by examining differences in the ranks of the data rather than relying on raw values. This method was chosen because it is particularly useful in small sample sizes and does not require assumptions about the underlying distribution of the data, making it well-suited for our experiment. By basing the analysis on ranks, we minimized the influence of outliers and distributional irregularities, ensuring a more robust comparison of treatment effects.

Hypothesis 3: There is an interaction effect between the treatments.

H_0 : There is no interaction effect between the two treatments (i.e., the effect of one treatment does not depend on the presence or absence of the other).

H_1 : There is an interaction effect between the two treatments (i.e., the effect of one treatment depends on whether the other treatment is applied).

See code and results in Appendix.

Figure 10: p-values for Rank Permutation Test for Different Treatments.

The p-values presented in Figure 10 indicate that cooking bananas in the oven produced a significant discrepancy on day 3, suggesting that cooking is an effective method for accelerating the ripening process after a certain amount of time. The trend of smaller p-values on day 3 can be interpreted as follows: on the first day, the bananas show little acceleration in the ripening process due to the initial stages of treatment, while by the last day, all groups—including the control—reach a similar ripeness stage. This suggests that the impact of cooking on ripening is most pronounced around day 3, after which the differences between treatments diminish as the bananas naturally converge to similar ripeness levels.

In addition to the rank permutation test, we employed propensity score matching (PSM) to reduce bias in comparing the different ripening techniques. PSM is a statistical method used to control for confounding variables by matching treated and control groups based on their observed characteristics. This approach allowed

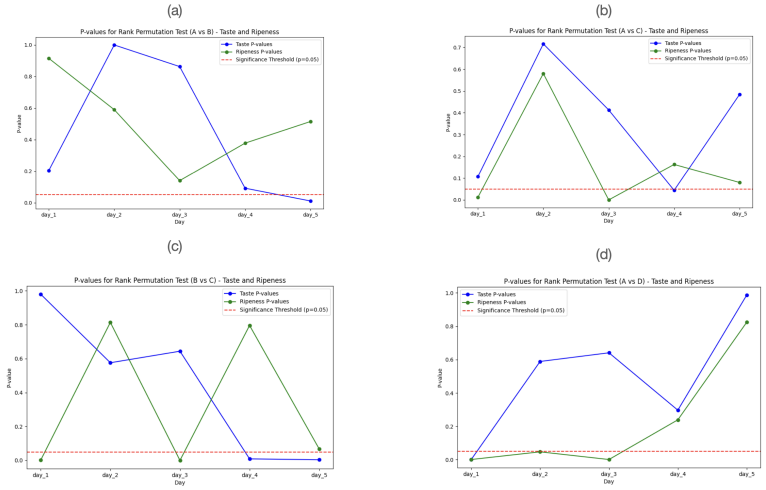

Figure 10: png

us to better isolate the effects of the ripening while testing the effect on the taste and isolate taste while testing the effect on the ripeness. In such way PSM enabled a more balanced comparison, improving the internal validity of our analysis. Also, it allowed for efficient use of the data by maximizing the information extracted from our relatively small sample size.

In our analysis using Propensity Score Matching (PSM), we conducted two paralel studies to evaluate the effects of our treatments on different outcomes: taste and ripeness.

First Study (Taste as the Target Outcome) In this study, we focused on taste as the primary outcome. To ensure that the groups were comparable in terms of ripeness, we matched the bananas based on their ripeness scores. After creating these matched groups, we applied PSM to evaluate the effect of the different ripening techniques on taste, aiming to isolate the impact of the treatments on flavor while controlling for ripeness. We assess the taste of the bananas under different treatment in this trial while controlling that different subjects give similar view on the ripeness.

Ripeness as the Target Outcome) In the second study, we shifted our focus to ripeness as the target outcome. This time, we matched the groups based on their taste scores to create comparable groups in terms of flavor. Similarly, we assess the subjects' view of the bananas' ripeness under different treatment while controlling that the bananas have similar tastes.

The results of this analysis, presented in Figure 11, show a clear pattern: on day 3, cooking in the oven had a significant impact on ripeness. This finding

aligns with previous observations that oven cooking accelerates ripening within a specific time frame, with the most significant effect occurring around day 3. After this point, the differences in ripeness between treatment groups diminished as all bananas approached a similar ripeness stage.

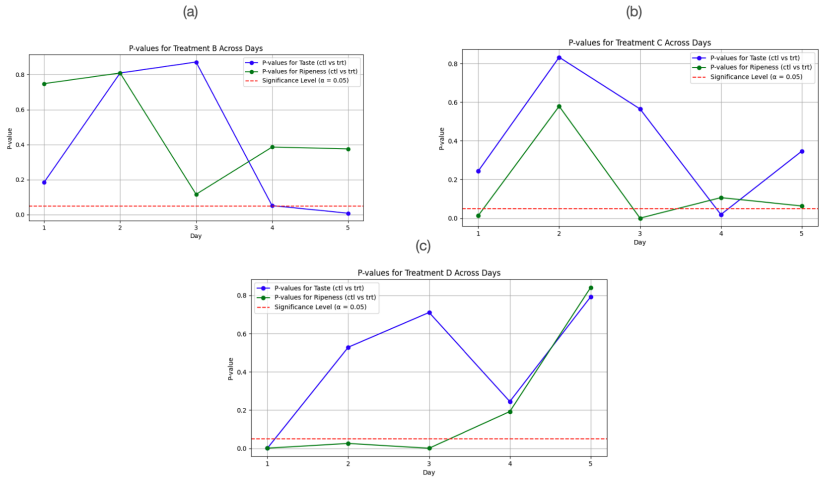

Figure 11: png

Figure 11: p-values for Different Treatments across Days.

Conclusion

The oven treatment demonstrated a clear and significant impact on accelerating the ripening process of bananas. Despite the inherent challenges in maintaining consistency across participants and subjective measurements, the results consistently indicated that cooking in the oven effectively hastens the ripening process compared to other techniques.

A notable challenge in the study was the considerable variability in the participants' evaluations of the bananas. Many participants provide ratings with little effort, with some repeatedly giving the same score across different days and treatments. This lack of differentiation made it difficult to capture variations.

The study also highlighted the inherent subjectivity of taste as a measure. Taste perception can vary significantly from person to person, influenced by individual preferences and sensory sensitivity. This variability made it challenging to draw definitive conclusions based solely on taste ratings

Appendix

Further work

Several challenges arose during our experiment that likely impacted our study's results. First, it is important to note that this experiment was single-blinded. Although we made efforts to conduct it fairly, we cannot fully rule out the possibility of unintentional influence on the outcomes.

A significant issue was the Lugol's iodine test. Despite our attempts to standardize conditions—using banana slices of similar size, applying consistent amounts of Lugol's iodine, and taking photos under controlled lighting after 30 seconds—the results were not as clear as anticipated. For future research, we suggest using color calibration markers (such as specific RGB values) in each photo to minimize the effect of camera auto-correction. Additionally, employing computer vision techniques to quantify color pixels could improve the reliability of Lugol's test results. Another approach would be to have objective, third-party evaluators (unfamiliar with the experimental design) rate the images, though this would still require consistent image quality to minimize nuisance variability.

Another limitation in this study was the small sample size. We recommend increasing the sample size by including multiple banana bunches per day, which could better account for day-to-day variability and yield more robust results.

Finally, our volunteers had varied preferences, with some favoring greener bananas. This preference variability likely introduced bias, making some responses less informative. We suggest conducting conditional analyses in future studies to account for participants' individual preferences in the evaluation process, which could provide a more nuanced understanding of the treatment effects.

Code

Data collection part

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Define the root folder where the data is stored
root_folder = 'data'

# Initialize a dictionary to store data frames for each day
data_frames = {}

# Loop over the days (folders day_1 to day_6)
for n in range(1, 7):
    day_folder = os.path.join(root_folder, f'day_{n}')
```

```
    # Get the list of .csv files in the current day's folder
    csv_files = [f for f in os.listdir(day_folder) if f.endswith('.csv')]

    # Read each .csv file into a pandas DataFrame
    for file in csv_files:
        # Construct full file path
        file_path = os.path.join(day_folder, file)

        # Create a unique key for each data frame based on day and file name
        df_key = f'{file.split(" ")[0]}'

        # Read the csv and store in the dictionary
        content = pd.read_csv(file_path)
        df_value = content.drop('response_id ', axis=1)
        data_frames[df_key] = df_value

# Initialize an empty dictionary to store the mean values for ripeness DataFrames
ripeness_means = {}

# Loop through the data_frames dictionary
for df_key, df in data_frames.items():
    # Check if the DataFrame name contains the word "ripeness"
    if "ripeness" in df_key:
        # Calculate the mean of each column in the DataFrame
        mean_values = df.mean(numeric_only=True)  # Only calculate for numeric columns

        # Store the mean values in the ripeness_means dictionary with the corresponding Dat
        ripeness_means[df_key] = mean_values

# Now ripeness_means contains the mean values for every DataFrame with "ripeness" in its na

# Initialize an empty dictionary to store the mean values for ripeness DataFrames
ripeness_data = {}

# Loop through the data_frames dictionary
for df_key, df in data_frames.items():
    # Check if the DataFrame name contains the word "ripeness"
    if "ripeness" in df_key:
        # Store the mean values in the ripeness_means dictionary with the corresponding Dat
        ripeness_data[df_key] = df

#### First two graphics

data_for_boxplot = []

# Create a list of days and groups based on the ripeness_means keys
```

```
days = sorted([int(key.split('_')[1]) for key in ripeness_means.keys()])
groups = ripeness_means[list(ripeness_means.keys())[0]].index  # Assuming the same groups a

# Populate the data for each day and group
for day in days:
    for group in groups:
        key = f'day_{day}_ripeness'

        # Assuming ripeness_data[key] contains the actual ripeness values with variation
        values = ripeness_data[key][group]  # Extract the list of ripeness values for this

        # Append each value along with its day and group to the list for the boxplot
        for value in values:
            data_for_boxplot.append({'Day': day, 'Group': group, 'Ripeness': value})

# Convert to a pandas DataFrame
df_boxplot = pd.DataFrame(data_for_boxplot)

# Create the boxplots using seaborn
plt.figure(figsize=(12, 6))

# Boxplot: hue represents different groups
sns.boxplot(x="Day", y="Ripeness", hue="Group", data=df_boxplot)

# Add labels, title, and legend
plt.xlabel('Day')
plt.ylabel('Ripeness')
plt.title('Ripeness Distribution Across Days by Group')
plt.legend(title='Groups', bbox_to_anchor=(1.05, 1), loc='upper left')

# Show the plot
plt.tight_layout()
plt.savefig("ripeness_distribution.png", dpi=300, bbox_inches='tight')
plt.show()

# Create a list of days and groups based on the ripeness_means keys
days = sorted([int(key.split('_')[1]) for key in ripeness_means.keys()])
groups = ripeness_means[list(ripeness_means.keys())[0]].index  # Assuming the same groups a

# Initialize a dictionary to store mean values for each group across days
group_means = {group: [] for group in groups}

# Populate the group_means dictionary with data from ripeness_means
for day in days:
    for group in groups:
        # Find the key for the current day and group
```

```
        key = f'day_{day}_ripeness'

        # Append the mean value for the current group to the group_means dictionary
        group_means[group].append(ripeness_means[key][group])

# Now plot the means for each group across days
plt.figure(figsize=(12, 6))

for group, means in group_means.items():
    plt.plot(days, means, marker='o', label=group)

# Add labels, title, and legend
plt.xlabel('Day')
plt.ylabel('Mean Ripeness')
plt.title('Mean Ripeness Across Days by Group')
plt.legend(title='Groups')
plt.grid(True)

plt.savefig("mean_ripeness.png", dpi=300, bbox_inches='tight')
# Show the plot
plt.show()
```

Combined treatment graphics

```
# Define the groups
days = range(1, 7)  # Assuming days are from 1 to 5
all_groups = ['A', 'B', 'C', 'D']  # The original 4 groups
target_group = 'A'  # The group to isolate

# Initialize a list to collect ripeness values for the boxplot
data_for_boxplot = []

# Loop over each day to prepare the data
for day in days:
    for group in all_groups:
        key = f'day_{day}_ripeness'

        # Get the ripeness values for the current group and day
        values = ripeness_data[key][group]  # Extract the list of ripeness values

        # For Group A, we store values as is
        if group == target_group:
            for value in values:
                data_for_boxplot.append({'Day': day, 'Group': 'Control', 'Ripeness': value})
        else:
            # Collect values from other groups to calculate their mean later
            if group != target_group:
```

```
                # Store these values as part of the "Other Groups" collection for now
                for value in values:
                    data_for_boxplot.append({'Day': day, 'Group': 'Mean of Other Groups', 'R

# Convert to a pandas DataFrame for boxplot visualization
df_boxplot = pd.DataFrame(data_for_boxplot)

# Create the boxplots using seaborn
plt.figure(figsize=(10, 6))

# Use seaborn to create a boxplot where "Day" is on the x-axis and "Ripeness" on the y-axis
sns.boxplot(x="Day", y="Ripeness", hue="Group", data=df_boxplot)

# Add labels, title, and legend
plt.xlabel('Day')
plt.ylabel('Ripeness')
plt.title('Ripeness Distribution Across Days: Control Group vs Mean of Other Groups')
plt.legend(title='Groups', bbox_to_anchor=(1.05, 1), loc='upper left')

# Show the plot
plt.tight_layout()
plt.savefig("ripeness_distribution_2.png", dpi=300, bbox_inches='tight')
plt.show()

# Create a list of days based on the ripeness_means keys
days = sorted([int(key.split('_')[1]) for key in ripeness_means.keys()])

# Define the groups
all_groups = ripeness_means[list(ripeness_means.keys())[0]].index  # Assuming the same group
target_group = 'A'  # The group to isolate

# Initialize lists to store means for Group A and Mean of Other Groups
group_a_means = []
mean_other_groups = []

# Populate the lists with data from ripeness_means
for day in days:
    key = f'day_{day}_ripeness'

    # Append the mean value for Group A
    group_a_means.append(ripeness_means[key][target_group])

    # Calculate the mean of other groups (B, C, D)
    other_groups = [group for group in all_groups if group != target_group]
    mean_of_others = ripeness_means[key][other_groups].mean()
    mean_other_groups.append(mean_of_others)
```

```
# Now plot the means for Group A and Mean of Other Groups
plt.figure(figsize=(12, 6))

# Plot Group A
plt.plot(days, group_a_means, marker='o', label='Control', color='blue')

# Plot Mean of Other Groups
plt.plot(days, mean_other_groups, marker='o', label='Mean of Treatment Groups', color='orang

# Add labels, title, and legend
plt.xlabel('Day')
plt.ylabel('Mean Ripeness')
plt.title('Mean Ripeness Across Days: Group A vs Mean of Other Groups')
plt.legend(title='Groups')
plt.grid(True)

plt.savefig("mean_ripeness_2.png", dpi=300, bbox_inches='tight')
# Show the plot
plt.show()
```

```
#### Hypothesis testing part
```

```
import scipy.stats as ss

alpha = 0.05

# Function to run Shapiro-Wilk normality test
def sh_normality_test(data):
    stat, p_value = ss.shapiro(data)
    return stat, p_value

# Iterate over each day's data
for day_key in ripeness_data.keys():
    print(f"Results for {day_key}:")

    # Extract the ripeness data for the current day
    current_day_data = ripeness_data[day_key]

    # Perform the Shapiro-Wilk test for group A and group B
    stat_A, p_value_A = sh_normality_test(current_day_data['A'])
    stat_B, p_value_B = sh_normality_test(current_day_data['B'])

    # Print the test results
    print(f"Control group - Test Statistic: {stat_A}, P-value: {p_value_A}")
    if p_value_A > alpha:
        print("The data follows a normal distribution.")
```

```
    else:
        print("The data does NOT follow a normal distribution.")
    print(f"Treatment group - Test Statistic: {stat_B}, P-value: {p_value_B}")
    if p_value_B > alpha:
        print("The data follows a normal distribution.")
    else:
        print("The data does NOT follow a normal distribution.")
    # If p-values > 0.05, normality is assumed; otherwise, use non-parametric tests
    if p_value_A > alpha and p_value_B > alpha:
        # Perform t-test
        t_stat, t_p_value = ss.ttest_ind(current_day_data['A'], current_day_data['B'], equal
        print(f"T-test p-value: {t_p_value}")

        if t_p_value <= alpha:
            print(f" We have enough evidance to reject the null hypothesis on 0.05 significa
    else:
        # Perform Mann-Whitney U test
        u_stat, u_p_value = ss.mannwhitneyu(current_day_data['A'], current_day_data['B'])
        print(f"Mann-Whitney U-test p-value: {u_p_value}")

        if u_p_value <= alpha:
            print(f"We have enough evidance to reject the null hypothesis on 0.05 significan

    print('-' * 50)
```

```
Results for day_1_ripeness:
Control group - Test Statistic: 0.929253876209259, P-value: 0.4405936300754547
The data follows a normal distribution.
Treatment group - Test Statistic: 0.9519404172897339, P-value: 0.6914888024330139
The data follows a normal distribution.
T-test p-value: 0.7481272259084623
--------------------------------------------------
Results for day_2_ripeness:
Control group - Test Statistic: 0.7241675853729248, P-value: 0.0042136418633162975
The data does NOT follow a normal distribution.
Treatment group - Test Statistic: 0.8352138996124268, P-value: 0.06723642349243164
The data follows a normal distribution.
Mann-Whitney U-test p-value: 0.6094657688943073
--------------------------------------------------
Results for day_3_ripeness:
Control group - Test Statistic: 0.8917765021324158, P-value: 0.20807792246341705
The data follows a normal distribution.
Treatment group - Test Statistic: 0.751937985420227, P-value: 0.005709535907953978
The data does NOT follow a normal distribution.
Mann-Whitney U-test p-value: 0.09157243311857743
--------------------------------------------------
```

```
Results for day_4_ripeness:
Control group - Test Statistic: 0.8715152740478516, P-value: 0.15595243871212006
The data follows a normal distribution.
Treatment group - Test Statistic: 0.935035765171051, P-value: 0.5629903674125671
The data follows a normal distribution.
T-test p-value: 0.38577462673954366
--------------------------------------------------
Results for day_5_ripeness:
Control group - Test Statistic: 0.8444785475730896, P-value: 0.06475866585969925
The data follows a normal distribution.
Treatment group - Test Statistic: 0.8680493831634521, P-value: 0.1171671599149704
The data follows a normal distribution.
T-test p-value: 0.37542162416923663
--------------------------------------------------
Results for day_6_ripeness:
Control group - Test Statistic: 0.9306910634040833, P-value: 0.5223867297172546
The data follows a normal distribution.
Treatment group - Test Statistic: 0.9306910634040833, P-value: 0.5223867297172546
The data follows a normal distribution.
T-test p-value: 1.0
--------------------------------------------------
```

```python
# Iterate over each day's data
for day_key in ripeness_data.keys():
    print(f"Results for {day_key}:")

    # Extract the ripeness data for the current day
    current_day_data = ripeness_data[day_key]

    # Perform the Shapiro-Wilk test for group A and group B
    stat_A, p_value_A = sh_normality_test(current_day_data['A'])
    stat_C, p_value_C = sh_normality_test(current_day_data['C'])

   # Print the test results
    print(f"Control group - Test Statistic: {stat_A}, P-value: {p_value_A}")
    if p_value_A > alpha:
        print("The data follows a normal distribution.")
    else:
        print("The data does NOT follow a normal distribution.")
    print(f"Treatment group - Test Statistic: {stat_C}, P-value: {p_value_C}")
    if p_value_C > alpha:
        print("The data follows a normal distribution.")
    else:
        print("The data does NOT follow a normal distribution.")
    # If p-values > 0.05, normality is assumed; otherwise, use non-parametric tests
    if p_value_A > alpha and p_value_C > alpha:
```

```
        # Perform t-test
        t_stat, t_p_value = ss.ttest_ind(current_day_data['A'], current_day_data['C'], equal
        print(f"T-test p-value: {t_p_value}")

        if t_p_value <= alpha:
            print(f"We have enough evidance to reject the null hypothesis on 0.05 significan
    else:
        # Perform Mann-Whitney U test
        u_stat, u_p_value = ss.mannwhitneyu(current_day_data['A'], current_day_data['C'])
        print(f"Mann-Whitney U-test p-value: {u_p_value}")

        if u_p_value <= alpha:
            print(f"We have enough evidance to reject the null hypothesis on 0.05 significan

    print('-' * 50)
```

```
Results for day_1_ripeness:
Control group - Test Statistic: 0.929253876209259, P-value: 0.4405936300754547
The data follows a normal distribution.
Treatment group - Test Statistic: 0.640485405921936, P-value: 0.00016868000966496766
The data does NOT follow a normal distribution.
Mann-Whitney U-test p-value: 0.017217395363837748
We have enough evidance to reject the null hypothesis on 0.05 significant level, which sugge
--------------------------------------------------
Results for day_2_ripeness:
Control group - Test Statistic: 0.7241675853729248, P-value: 0.0042136418633162975
The data does NOT follow a normal distribution.
Treatment group - Test Statistic: 0.9299981594085693, P-value: 0.5160583257675171
The data follows a normal distribution.
Mann-Whitney U-test p-value: 0.5808403603117523
--------------------------------------------------
Results for day_3_ripeness:
Control group - Test Statistic: 0.8917765021324158, P-value: 0.20807792246341705
The data follows a normal distribution.
Treatment group - Test Statistic: 0.8463110327720642, P-value: 0.0678541362285614
The data follows a normal distribution.
T-test p-value: 6.869420592701321e-05
We have enough evidance to reject the null hypothesis on 0.05 significant level, which sugge
--------------------------------------------------
Results for day_4_ripeness:
Control group - Test Statistic: 0.8715152740478516, P-value: 0.15595243871212006
The data follows a normal distribution.
Treatment group - Test Statistic: 0.9172791838645935, P-value: 0.40817752480506897
The data follows a normal distribution.
T-test p-value: 0.10619158583153715
--------------------------------------------------
```

```
Results for day_5_ripeness:
Control group - Test Statistic: 0.8444785475730896, P-value: 0.06475866585969925
The data follows a normal distribution.
Treatment group - Test Statistic: 0.9165481328964233, P-value: 0.3644281327724457
The data follows a normal distribution.
T-test p-value: 0.06298904449383765
--------------------------------------------------
Results for day_6_ripeness:
Control group - Test Statistic: 0.9306910634040833, P-value: 0.5223867297172546
The data follows a normal distribution.
Treatment group - Test Statistic: 0.7321971654891968, P-value: 0.005180804990231991
The data does NOT follow a normal distribution.
Mann-Whitney U-test p-value: 0.21972787984439068
--------------------------------------------------
```

```python
# Permutation Test

from scipy.stats import rankdata

def rank_permutation_test(data, group1, group2, num_permutations=1000):
    # Extract the values for the two groups based on their labels
    values_group1 = data[group1].values.flatten()
    values_group2 = data[group2].values.flatten()

    # Rank the combined data
    combined_data = np.concatenate([values_group1, values_group2])
    ranks = rankdata(combined_data)

    # Compute the observed test statistic (difference in rank sums)
    observed_rank_sum_1 = np.sum(ranks[:len(values_group1)])
    observed_rank_sum_2 = np.sum(ranks[len(values_group1):])
    observed_diff = observed_rank_sum_1 - observed_rank_sum_2

    # Perform permutations
    perm_diffs = []
    for _ in range(num_permutations):
        np.random.shuffle(ranks)
        perm_rank_sum_1 = np.sum(ranks[:len(values_group1)])
        perm_rank_sum_2 = np.sum(ranks[len(values_group1):])
        perm_diff = perm_rank_sum_1 - perm_rank_sum_2
        perm_diffs.append(perm_diff)

    # Compute the p-value
    p_value = np.mean(np.abs(perm_diffs) >= np.abs(observed_diff))

    return observed_diff, p_value
```

```
taste_p_values = []
ripeness_p_values = []

# Perform permutation tests for both taste and ripeness for each day
for day in range(1, 6):
    # For taste
    day_taste_key = f'day_{day}_taste'
    if day_taste_key in data_frames:
        observed_diff_taste, p_value_taste = rank_permutation_test(data_frames[day_taste_key
        taste_p_values.append(p_value_taste)

    # For ripeness
    day_ripeness_key = f'day_{day}_ripeness'
    if day_ripeness_key in data_frames:
        observed_diff_ripeness, p_value_ripeness = rank_permutation_test(data_frames[day_rip
        ripeness_p_values.append(p_value_ripeness)
```

```
# Propensity Score Matching

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import NearestNeighbors
from scipy.stats import ttest_ind


trt = 'B'
#trt = 'C'
#trt = 'D'
ctl = 'A'

# Initialize a list to store p-values for each day
p_values = []

# Step 1: Loop through each day
for day in range(1, 6):
    # Combine Data for the current day
    day_taste_data = data_frames[f'day_{day}_taste'][[ctl, trt]]
    day_ripeness_data = data_frames[f'day_{day}_ripeness']

    # Step 2: Combine A and B into a single DataFrame and label them
    taste_data = pd.melt(day_taste_data, var_name='treatment', value_name='taste_score')

    # Step 3: Create Treatment Indicator
    # Note: 'ctl' and 'trt' as treatment groups are defined here
    taste_data['treatment'] = taste_data['treatment'].map({ctl: ctl, trt: trt})

    # Step 4: Estimate Propensity Scores using Logistic Regression
```

```
    model = LogisticRegression()
    model.fit(taste_data[['taste_score']], taste_data['treatment'])

    # Predict propensity scores
    taste_data['propensity_score'] = model.predict_proba(taste_data[['taste_score']])[:, 1]

    # Step 5: Match Treatment Groups using Nearest Neighbor
    treatment_ctl = taste_data[taste_data['treatment'] == ctl]
    treatment_trt = taste_data[taste_data['treatment'] == trt]

    # Fit NearestNeighbors model on propensity scores
    nn = NearestNeighbors(n_neighbors=1)
    nn.fit(treatment_ctl[['propensity_score']])

    # Find matches for treatment group trt
    distances, indices = nn.kneighbors(treatment_trt[['propensity_score']])

    # Retrieve matched data
    matched_ctl = treatment_ctl.iloc[indices.flatten()]
    matched_trt = treatment_trt.reset_index(drop=True)

    # Combine matched datasets
    matched_data = pd.concat([matched_ctl.reset_index(drop=True), matched_trt], axis=0)

    # Step 6: Compare Treatment Effects in Ripeness
    matched_ripeness_ctl = day_ripeness_data.loc[matched_data.index[matched_data['treatment'
    matched_ripeness_trt = day_ripeness_data.loc[matched_data.index[matched_data['treatment'

    # Conduct a t-test to compare ripeness between matched groups
    t_stat, p_value = ttest_ind(matched_ripeness_ctl, matched_ripeness_trt, equal_var=False)
    p_values.append(p_value)


# Step 7: Aggregate p-values (for visualization, you may use multiple methods)
# Here we'll use the average of p-values
aggregated_p_value_ripeness = np.mean(p_values)
p_values_ripeness = p_values


# Initialize a list to store p-values for each day
p_values = []

# Step 1: Loop through each day
for day in range(1, 6):
    # Combine Data for the current day
    day_ripeness_data = data_frames[f'day_{day}_taste']
```

```
        day_taste_data = data_frames[f'day_{day}_ripeness'][[ctl, trt]]

        # Step 2: Combine A and B into a single DataFrame and label them
        taste_data = pd.melt(day_taste_data, var_name='treatment', value_name='taste_score')

        # Step 3: Create Treatment Indicator
        # Note: 'ctl' and 'trt' as treatment groups are defined here
        taste_data['treatment'] = taste_data['treatment'].map({ctl: ctl, trt: trt})

        # Step 4: Estimate Propensity Scores using Logistic Regression
        model = LogisticRegression()
        model.fit(taste_data[['taste_score']], taste_data['treatment'])

        # Predict propensity scores
        taste_data['propensity_score'] = model.predict_proba(taste_data[['taste_score']])[:, 1]

        # Step 5: Match Treatment Groups using Nearest Neighbor
        treatment_ctl = taste_data[taste_data['treatment'] == ctl]
        treatment_trt = taste_data[taste_data['treatment'] == trt]

        # Fit NearestNeighbors model on propensity scores
        nn = NearestNeighbors(n_neighbors=1)
        nn.fit(treatment_ctl[['propensity_score']])

        # Find matches for treatment group trt
        distances, indices = nn.kneighbors(treatment_trt[['propensity_score']])

        # Retrieve matched data
        matched_ctl = treatment_ctl.iloc[indices.flatten()]
        matched_trt = treatment_trt.reset_index(drop=True)

        # Combine matched datasets
        matched_data = pd.concat([matched_ctl.reset_index(drop=True), matched_trt], axis=0)

        # Step 6: Compare Treatment Effects in Ripeness
        matched_ripeness_ctl = day_ripeness_data.loc[matched_data.index[matched_data['treatment'
        matched_ripeness_trt = day_ripeness_data.loc[matched_data.index[matched_data['treatment'

        # Conduct a t-test to compare ripeness between matched groups
        t_stat, p_value = ttest_ind(matched_ripeness_ctl, matched_ripeness_trt, equal_var=False)
        p_values.append(p_value)


# Step 7: Aggregate p-values (for visualization, you may use multiple methods)
# Here we'll use the average of p-values
aggregated_p_value_taste = np.mean(p_values)
```

```
p_values_taste = p_values

# Causal Forest
from econml.dml import LinearDML
from econml.dml import CausalForestDML
from sklearn.model_selection import train_test_split

# Initialize a dictionary to store treatment effects for each day
treatment_effects_by_day = {}

# Loop through each day
for day in range(1, 6):
    # Combine Data for the current day
    day_taste_data = data_frames[f'day_{day}_taste'][['A', 'B']]
    day_ripeness_data = data_frames[f'day_{day}_ripeness']

    # Create a DataFrame with combined treatment and covariate data
    combined_data = pd.DataFrame({
        'ripeness': np.concatenate([day_ripeness_data['A'].values, day_ripeness_data['B'].va
        'treatment': np.concatenate([np.zeros(len(day_taste_data['A'])), np.ones(len(day_tas
        'taste': np.concatenate([day_taste_data['A'].values, day_taste_data['B'].values])  #
    })

    # Prepare outcome, treatment, and covariates
    y = combined_data['ripeness'].values  # Outcome variable (ripeness)
    t = combined_data['treatment'].values  # Treatment indicator
    X = combined_data[['taste']].values  # Covariates (taste)

    # Train-test split
    y_train, y_test, t_train, t_test, X_train, X_test = train_test_split(y, t, X, test_size=

    # Initialize and fit the Causal Forest model
    causal_forest = CausalForestDML(n_estimators=100, min_samples_leaf=5, random_state=42)
    causal_forest.fit(Y=y_train, T=t_train, X=X_train)

    # Estimate treatment effects
    treatment_effects = causal_forest.effect(X=X_test)  # Use the test set for prediction

    # Store treatment effects for the current day
    treatment_effects_by_day[day] = np.mean(treatment_effects)  # Averaging treatment effec

# Convert the dictionary to a DataFrame for easier analysis
treatment_effects_df = pd.DataFrame.from_dict(treatment_effects_by_day, orient='index')
treatment_effects_df.columns = ['Estimated Treatment Effects']

# Display estimated treatment effects for each day
```

```
print(treatment_effects_df)

# Initialize a dictionary to store treatment effects for each day
treatment_effects_by_day = {}

# Loop through each day
for day in range(1, 6):
    # Combine Data for the current day
    day_taste_data = data_frames[f'day_{day}_taste'][['A', 'C']]
    day_ripeness_data = data_frames[f'day_{day}_ripeness']

    # Create a DataFrame with combined treatment and covariate data
    combined_data = pd.DataFrame({
        'ripeness': np.concatenate([day_ripeness_data['A'].values, day_ripeness_data['B'].va
        'treatment': np.concatenate([np.zeros(len(day_taste_data['A'])), np.ones(len(day_tas
        'taste': np.concatenate([day_taste_data['A'].values, day_taste_data['C'].values])  #
    })

    # Prepare outcome, treatment, and covariates
    y = combined_data['ripeness'].values  # Outcome variable (ripeness)
    t = combined_data['treatment'].values  # Treatment indicator
    X = combined_data[['taste']].values  # Covariates (taste)

    # Train-test split
    y_train, y_test, t_train, t_test, X_train, X_test = train_test_split(y, t, X, test_size=

    # Initialize and fit the Causal Forest model
    causal_forest = CausalForestDML(n_estimators=100, min_samples_leaf=5, random_state=42)
    causal_forest.fit(Y=y_train, T=t_train, X=X_train)

    # Estimate treatment effects
    treatment_effects = causal_forest.effect(X=X_test)  # Use the test set for prediction

    # Store treatment effects for the current day
    treatment_effects_by_day[day] = np.mean(treatment_effects)  # Averaging treatment effec

# Convert the dictionary to a DataFrame for easier analysis
treatment_effects_df = pd.DataFrame.from_dict(treatment_effects_by_day, orient='index')
treatment_effects_df.columns = ['Estimated Treatment Effects']

# Display estimated treatment effects for each day
print(treatment_effects_df)

# Initialize a dictionary to store treatment effects for each day
treatment_effects_by_day = {}

# Loop through each day
```

```python
for day in range(1, 6):
    # Combine Data for the current day
    day_taste_data = data_frames[f'day_{day}_taste'][['A', 'D']]
    day_ripeness_data = data_frames[f'day_{day}_ripeness']

    # Create a DataFrame with combined treatment and covariate data
    combined_data = pd.DataFrame({
        'ripeness': np.concatenate([day_ripeness_data['A'].values, day_ripeness_data['D'].va
        'treatment': np.concatenate([np.zeros(len(day_taste_data['A'])), np.ones(len(day_tas
        'taste': np.concatenate([day_taste_data['A'].values, day_taste_data['D'].values])  #
    })

    # Prepare outcome, treatment, and covariates
    y = combined_data['ripeness'].values  # Outcome variable (ripeness)
    t = combined_data['treatment'].values  # Treatment indicator
    X = combined_data[['taste']].values  # Covariates (taste)

    # Train-test split
    y_train, y_test, t_train, t_test, X_train, X_test = train_test_split(y, t, X, test_size=

    # Initialize and fit the Causal Forest model
    causal_forest = CausalForestDML(n_estimators=100, min_samples_leaf=5, random_state=42)
    causal_forest.fit(Y=y_train, T=t_train, X=X_train)

    # Estimate treatment effects
    treatment_effects = causal_forest.effect(X=X_test)  # Use the test set for prediction

    # Store treatment effects for the current day
    treatment_effects_by_day[day] = np.mean(treatment_effects)  # Averaging treatment effec

# Convert the dictionary to a DataFrame for easier analysis
treatment_effects_df = pd.DataFrame.from_dict(treatment_effects_by_day, orient='index')
treatment_effects_df.columns = ['Estimated Treatment Effects']

# Display estimated treatment effects for each day
print(treatment_effects_df)
```

```python
from IPython.display import display, Image, Markdown


display(Image(filename="lugol.png"))
display(Markdown("**Figure 1:** Results of Lugol's Iodine test for day 1 and day 6."))


display(Image(filename="ripeness_distribution.png"))
display(Markdown("**Figure 2:** Ripeness Distribution across days by group."))
```

```
display(Image(filename="mean_ripeness.png"))
display(Markdown("**Figure 3:** Mean Ripeness across days by group."))

display(Image(filename="ripeness_distribution_2.png"))
display(Markdown("**Figure 4:** Ripeness Distribution Across Days: Control Group vs Mean of

display(Image(filename="mean_ripeness_2.png"))
display(Markdown("**Figure 5:** Mean Ripeness Across Days: Control Group vs Mean of Other Gr

display(Image(filename="mean_taste_across_days.png"))
display(Markdown("**Figure 6:** Mean Taste Across Days by Group."))

display(Image(filename="mean_taste_across_days_2.png"))
display(Markdown("**Figure 7:** Mean Taste Across Days by Group (Control and Combined Treatm

display(Image(filename="taste_distribution_across_days.png"))
display(Markdown("**Figure 8:** Taste Distribution Across Days by Group."))

display(Image(filename="taste_distribution_across_days_2.png"))
display(Markdown("**Figure 9:** Taste Distribution Across Days by Group (Control and Combine

display(Image(filename="permutation_test.png"))
display(Markdown("**Figure 10:** p-values for Rank Permutation Test for Different Treatments

display(Image(filename="psm.png"))
display(Markdown("**Figure 11:** p-values for Different Treatments across Days."))
```