Smart Nutrition: Predicting Personalized Meal Plans Using AI

Tahir Shaikh

BSC.IT, Jai Hind College

INTRODUCTION:

In today's fast-paced world, maintaining a balanced diet tailored to individual needs is crucial for overall health and well-being. This project leverages machine learning to predict the optimal number of meals per day based on a person's demographics and lifestyle habits. By analysing factors such as age, gender, activity level, sleep duration, and diet preferences, the model provides personalized meal recommendations to promote healthier eating habits.

The system utilizes multiple regression techniques to establish relationships between lifestyle factors and meal frequency, enabling users to receive scientifically-backed meal plan suggestions. This predictive approach can help individuals optimize their nutrition, improve energy levels, and maintain a sustainable and healthy lifestyle. The project aims to assist fitness enthusiasts, diet-conscious individuals, and healthcare professionals in making informed dietary decisions using AI-driven insights.

PROBLEM STATEMENT:

1. Predict a User's Activity Level Based on Their Lifestyle factors, such as age, weight, height, sleep duration, drink usage, eating habits, and diet plan Using KNN & Random Forest.
2. Predict Whether a User is Likely to Choose a Diet Focused on Short-Term or Long-Term Health Goals Using Logistic Regression & Random Forest.
3. Predict an individual's ideal daily calorie intake based on their age, weight, height, activity level, and sleep duration Using Multiple Linear Regression.
4. Predict whether a user is trying to lose weight, gain weight, or maintain their current weight based on lifestyle factors like age, activity level, sleep patterns, and dietary habits using Knn.
5. Cluster users into different dietary habit groups based on their food preferences, meal frequency, protein intake, and other lifestyle factors using Kmean Clustering.
6. Predict whether a user will follow a structured diet plan based on their medical conditions, food allergies, age, and activity level using Knn.
7. Predict whether a user has medical conditions based on their weight, height, sleep duration, drinking habits, and meal frequency using Decision Tree.
8. Cluster users into different lifestyle groups based on their activity level, sleep duration, eating-out frequency, and meal intake using K-means.
9. Recommend the best protein source for a user based on their age, weight, activity level, and medical conditions Using Knn.
10. Predict whether an individual prefers eating meals at home or outside based on their activity level, sleep duration, drink usage, medical conditions, and food allergies Using Decision Tree.

DATA DESCRIPTION: (Dataset Preview)

Column Name	Description	Data Type
1. Age	The age of the individual in years.	Integer
2. Gender	The gender of the individual (Male/Female).	String
3. Weight	The weight of the individual in kilograms.	Float
4. Height	The height of the individual in centimeters.	Float
5. Activity	The activity level ranging from sedentary to very active.	Categorical
6. Sleep	The average sleep duration (e.g., 7-8 hours, less than 5 hours).	Categorical
7. Drink Usage	The frequency of consuming water a day.	Categorical
8. Eat Outside	The frequency of eating outside meals per week.	Categorical
9. Medical Conditions	Indicates whether the person has medical conditions (Yes/No).	Categorical
10. Food Allergies	Indicates whether the person has food allergies (Yes/No).	Categorical
11. Meals Per Day	The number of meals consumed daily.	Categorical
12. Source of Protein	The primary protein source (e.g., Dairy, Non-Veg, Plant-based).	Categorical
13. Lose/Gain Weight	Indicates whether the individual is aiming to lose or gain weight (Yes/No).	Categorical
14. Diet Plan	Indicates if the person follows a specific diet plan (Yes/No).	Categorical
15. Focuses short-long-term health	Indicates if the individual focuses on short-term or long-term health (Yes/No).	Categorical
16. Tracking Progress	Indicates if the individual regularly tracks their dietary progress (Yes/No).	Categorical

PRIMINALARY ANALYSIS (Preprocess Dataset):

```
import streamlit as st
import pandas as pd
import numpy as np
from scipy.stats import skew, kurtosis

def load_and_preprocess_data(file_path):
    st.title("📊 Diet Plan Data Preprocessing & Transformation")

    if file_path is not None:
        df = pd.read_csv(file_path)

        # 🔹 Step 1: Display Raw Data
        st.write("### 📂 Raw Dataset Preview")
        st.dataframe(df.head())

        # Select only numeric columns for analysis
        numeric_df = df.select_dtypes(include=[np.number])

        # 🔹 Step 2: Perform Data Analysis (Before Transformation)
        st.write("## 📊 Exploratory Data Analysis (EDA)")

        # 🔹 Basic Statistics
        st.write("### 🔢 Basic Statistical Metrics")

        # Calculate metrics
        mean_values = numeric_df.mean()
        median_values = numeric_df.median()
        mode_values = numeric_df.mode().iloc[0]
        std_dev = numeric_df.std()
        variance = numeric_df.var()

        # Combine all statistics into a single table
        stats_df = pd.DataFrame({
            "Mean": mean_values,
            "Median": median_values,
            "Mode": mode_values,
            "Standard Deviation": std_dev,
            "Variance": variance
        })

        st.write(stats_df)

        # 🔹 Skewness & Kurtosis
        st.write("### 📉 Skewness & Kurtosis")
        skewness = numeric_df.apply(lambda x: skew(x.dropna()))
        kurtosis_values = numeric_df.apply(lambda x: kurtosis(x.dropna()))
```

```
    st.write(" ◇ **Skewness**")
    st.write(skewness)

    st.write(" ◇ **Kurtosis**")
    st.write(kurtosis_values)

    # ◇ Duplicates Check
    duplicates = numeric_df.duplicated().sum()
    st.write(f"### 🔍 Number of Duplicates: {duplicates}")

    # ◇ Correlation Matrix
    st.write("### 🔗 Correlation Matrix")
    correlation_matrix = numeric_df.corr()
    st.write(correlation_matrix)

    # ◇ Key Correlations
    st.write("### 💧 Top 10 High-Correlation Pairs")
    high_correlation_pairs = correlation_matrix.unstack().sort_values(ascending=False)
    high_correlation_pairs = high_correlation_pairs[high_correlation_pairs < 1]  # Remove
self-correlations
    st.write(high_correlation_pairs.head(10))

    # ◇ Step 3: Identify Trends & Patterns
    st.write("## 📊 Trends and Patterns in the Dataset")

    # Activity Level vs. Meals per Day
    st.write(" ◇ **Activity Level and Meals per Day**")
    activity_meals = df.groupby("Activity")["Meals per day"].mean()
    st.write(activity_meals)

    # Sleep and Hydration Trends
    st.write(" ◇ **Sleep Duration and Water Intake**")
    sleep_hydration = df.groupby("Sleep")["Drink Usage"].mean()
    st.write(sleep_hydration)

    # Weight vs. Preferred Protein Source
    st.write(" ◇ **Weight vs. Preferred Protein Source**")
    weight_protein = df.groupby("Source of protein")["Weight"].mean()
    st.write(weight_protein)

    # Eating Out Frequency and Diet Plan
    st.write(" ◇ **Eating Out Frequency vs. Diet Plan Preference**")
    eat_out_diet = df.groupby("Eat Outside")["Diet Plan"].value_counts(normalize=True)
    st.write(eat_out_diet)

    # Weight Goals Trends
```

```
st.write("🔸 **Weight Goal Trends (Lose/Gain)**")
weight_goals = df["lose/gain weight"].value_counts()
st.write(weight_goals)

# 🔸 Quartiles and Deciles
st.write("### 🔢 Quartiles & Deciles")
st.write("📌 **Quartiles**")
st.write(numeric_df.quantile([0.25, 0.5, 0.75]))

st.write("📌 **Deciles**")
st.write(numeric_df.quantile(np.arange(0.1, 1, 0.1)))

# 🔸 Step 4: Perform Data Transformation (Encoding Categorical Variables)
st.write("## 🔄 Data Transformation (Encoding Categorical Variables)")

column_to_transform = [
    'Medical Conditions',
    'Food Allergies',
    'Diet Plan',
    'Focuses short-long-term health',
    'Track your progress',
    'lose/gain weight'
]

if 'Gender' in df.columns:
    df['Gender'] = df['Gender'].replace({'Male': 1, 'Female': 2})

if 'Activity' in df.columns:
    df['Activity'] = df['Activity'].replace({
        'Sedentary (little or no exercise)': 1,
        'Lightly active (light exercise 1-3 days/week)': 2,
        'Moderately active (moderate exercise 3-5 days/week)': 3,
        'Very active (intense exercise 6-7 days/week)': 4
    })

if 'Sleep' in df.columns:
    df['Sleep'] = df['Sleep'].replace({
        'Less than 5 hours': 4,
        '5-6 hours': 5,
        '7-8 hours': 7,
        'Over 8 hours': 8
    })

if 'Source of protein' in df.columns:
    df['Source of protein'] = df['Source of protein'].replace({
        'Dairy (Cheese, Yogurt)': 1,
        'Protein supplements (Whey, Plant-based protein)': 2,
        'Non-Veg': 3,
```

```
            'Plant-based (Legumes, Tofu, Tempeh)': 4
        })

    for col in column_to_transform:
        if col in df.columns:
            df[col] = df[col].replace({'Yes': 1, 'No': 0})

    #  ◇ Step 5: Display Transformed Data
    st.write("### 🔄 Transformed Data Preview")
    st.dataframe(df.head())

    # Save transformed data
    output_path = "C:\\Users\\ms828\\Downloads\\ca2\\Ca-
2\\Documents\\PreProcess_File.csv"
    df.to_csv(output_path, index=False)

    # Download Button
    st.download_button(
        "⬇ Download Transformed Data",
        data=df.to_csv(index=False),
        file_name="PreProcess_File.csv",
        mime="text/csv"
    )

    st.success(f"☑ The transformed dataset has been saved to `{output_path}`")

# Function to return output file path
def get_outputpath():
    return "C:\\Users\\ms828\\Downloads\\ca2\\Ca-2\\Documents\\PreProcess_File.csv"
```

🔢 Basic Statistical Metrics

	Mean	Median	Mode	Standard Deviation	Variance
Age	38.04	37	19	12.5841	158.3602
Weight	74.76	75	79	14.2671	203.5502
Height	173.8701	173.48	162.43	14.4726	209.4553
Drink Usage	2.36	2	1	1.1607	1.3471
Eat Outside	2.975	3	5	1.5119	2.2858
Meals per day	3.565	4	5	1.1144	1.242

Skewness & Kurtosis

Skewness

	0
Age	0.0654
Weight	0.0455
Height	0.1122
Drink Usage	0.1777
Eat Outside	0.0862
Meals per day	-0.0655

Kurtosis

	0
Age	-1.2668
Weight	-1.1319
Height	-1.1836
Drink Usage	-1.4227
Eat Outside	-1.4542
Meals per day	-1.3436

Correlation Matrix

	Age	Weight	Height	Drink Usage	Eat Outside	Meals per day
Age	1	-0.0841	0.0494	-0.0048	-0.052	-0.0081
Weight	-0.0841	1	0.0231	0.089	0.0941	-0.0278
Height	0.0494	0.0231	1	-0.0918	-0.0888	-0.0215
Drink Usage	-0.0048	0.089	-0.0918	1	0.0682	0.0362
Eat Outside	-0.052	0.0941	-0.0888	0.0682	1	0.0502
Meals per day	-0.0081	-0.0278	-0.0215	0.0362	0.0502	1

Trends and Patterns in the Dataset

Activity Level and Meals per Day

Activity	Meals per day
Lightly active (light exercise 1-3 days/week)	3.5227
Moderately active (moderate exercise 3-5 days/week)	3.6774
Sedentary (little or no exercise)	3.4545
Very active (intense exercise 6-7 days/week)	3.56

Weight Goal Trends (Lose/Gain)

lose/gain weigh	count
No	109
Yes	91

Weight vs. Preferred Protein Source

Source of protein	Weight
Dairy (Cheese, Yogurt)	75.9333
Non-Veg	75.0889
Plant-based (Legumes, Tofu, Tempeh)	76.814
Protein supplements (Whey, Plant-based protein)	71.4231

Eating Out Frequency vs. Diet Plan Preference

Eat Outside	Diet Plan	proportion
1	No	0.5111
1	Yes	0.4889
2	No	0.5111
2	Yes	0.4889
3	No	0.5
3	Yes	0.5
4	No	0.5667
4	Yes	0.4333
5	Yes	0.52
5	No	0.48

Sleep Duration and Water Intake

Sleep	Drink Usage
5-6 hours	2.4773
7-8 hours	2.2391
Less than 5 hours	2.3725
Over 8 hours	2.3559

Top 10 High-Correlation Pairs

		0
Weight	Eat Outside	0.0941
Eat Outside	Weight	0.0941
Drink Usage	Weight	0.089
Weight	Drink Usage	0.089
Drink Usage	Eat Outside	0.0682
Eat Outside	Drink Usage	0.0682
Meals per day	Eat Outside	0.0502
Eat Outside	Meals per day	0.0502
Age	Height	0.0494
Height	Age	0.0494

Quartiles & Deciles

Quartiles

	Age	Weight	Height	Drink Usage	Eat Outside	Meals per day
0.25	26	63	161.4975	1	2	3
0.5	37	75	173.48	2	3	4
0.75	50	88	186.73	3	4.25	5

Deciles

	Age	Weight	Height	Drink Usage	Eat Outside	Meals per day
0.1	21	55	154.766	1	1	2
0.2	25	60.8	159.166	1	1	2
0.3	29	65	164.183	1	2	3
0.4	33	71	168.5	2	2	3
0.5	37	75	173.48	2	3	4
0.6	42	79	177.13	3	3.4	4
0.7	48	83	184.277	3	4	4
0.8	51	90	188.242	4	5	5
0.9	55.1	94	194.532	4	5	5

Data Transformation (Encoding Categorical Variables)


```
{
  "Yes" : 1
  "No" : 0
}
```

```
{
  "Male" : 1
  "Female" : 2
}
```

```
{
  "Sedentary (little or no exercise)" : 1
  "Lightly active (light exercise 1-3 days/week)" : 2
  "Moderately active (moderate exercise 3-5 days/week)" : 3
  "Very active (intense exercise 6-7 days/week)" : 4
}
```

```
{
  "Less than 5 hours" : 4
  "5-6 hours" : 5
  "7-8 hours" : 7
  "Over 8 hours" : 8
}
```

```
{
  "Dairy (Cheese, Yogurt)" : 1
  "Protein supplements (Whey, Plant-based protein)" : 2
  "Non-Veg" : 3
  "Plant-based (Legumes, Tofu, Tempeh)" : 4
}
```


Transformed Data Preview

	Age	Gender	Weight	Height	Activity	Sleep	Drink Usage	Eat Outside	Medical Conditions	Fo
0	23	1	94	172.49	2	7	3	2	0	
1	40	1	55	150.14	3	7	1	2	1	
2	34	1	54	176.55	4	4	3	4	1	
3	18	2	73	196.69	1	7	2	5	0	
4	36	1	98	173.18	3	4	3	4	0	

⬇ Download Transformed Data

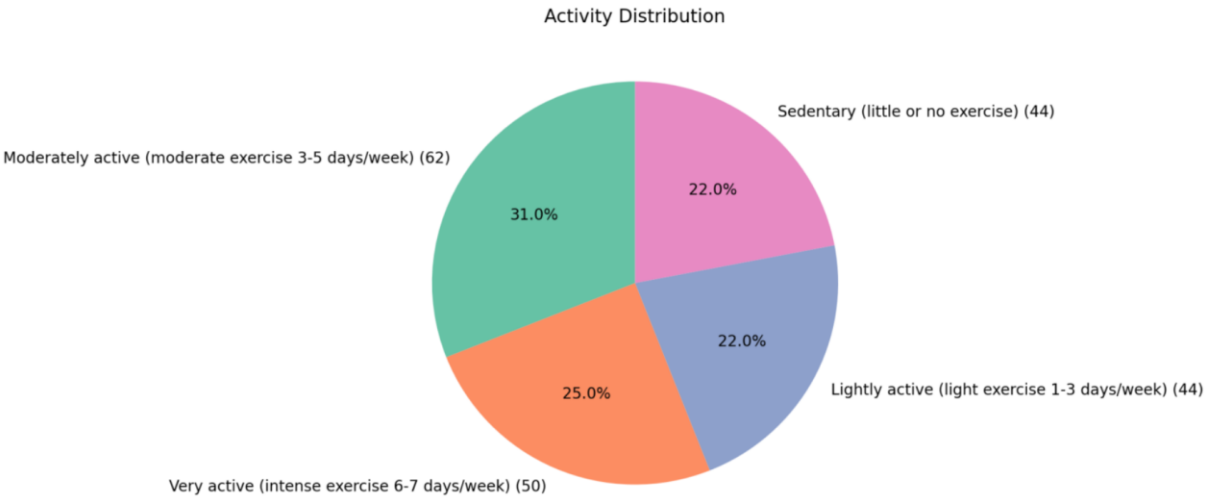
DATA VISUALIZATION:-

```
import streamlit as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.preprocessing import LabelEncoder


def visualize_data(file_path):
    st.title(" Data Visualization & Insights")

    # Load the dataset
    df = pd.read_csv(file_path)

    # Define Quantitative and Qualitative data
    quantitative_columns = ['Age', 'Weight', 'Height']
    qualitative_columns = ['Gender', 'Activity', 'Sleep', 'Medical Conditions', 'Food Allergies',
                           'Source of protein', 'lose/gain weight', 'Diet Plan',
                           'Focuses short-long-term health', 'Track your progress',
                           'Eat Outside', 'Drink Usage', 'Meals per day']

    st.write("This section presents visual insights from the dataset, including distributions,
trends, and correlations.")

    # ---- Pie Charts for Categorical Data ----
    st.header(" Pie Charts for Categorical Data")
    pie_columns = ['Activity', 'Sleep', 'Eat Outside', 'Drink Usage', 'Meals per day']

    for col in pie_columns:
        counts = df[col].value_counts()
        fig, ax = plt.subplots(figsize=(6, 6))
        ax.pie(counts, labels=[f'{label} ({count})' for label, count in zip(counts.index, counts)],
               autopct='%1.1f%%', startangle=90, colors=sns.color_palette('Set2',
n_colors=len(counts)))
        ax.set_title(f"{col} Distribution")
        st.pyplot(fig)

    # ---- Count Plots for Categorical Data ----
    st.header(" Categorical Data Distributions (Bar Charts)")
    for col in qualitative_columns:
        fig, ax = plt.subplots(figsize=(8, 5))
        sns.countplot(data=df, x=col, palette="Set2", order=df[col].value_counts().index)
        ax.set_title(f"Distribution of {col}")
        ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha="right")
        st.pyplot(fig)
```

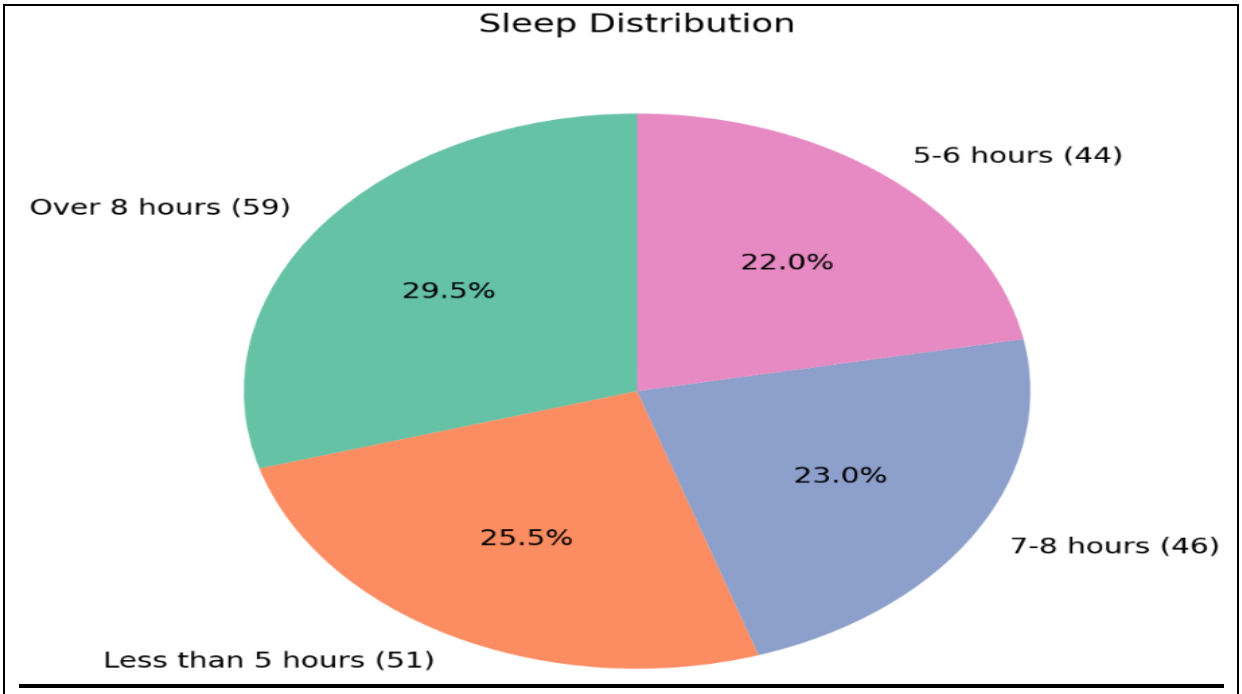
```python
    # ---- Histograms for Quantitative Data ----
    st.header("📊 Histograms for Numerical Data")
    hist_columns = ['Age', 'Weight', 'Height']
    hist_colors = ['skyblue', 'lightgreen', 'lightcoral']

    for col, color in zip(hist_columns, hist_colors):
        fig, ax = plt.subplots(figsize=(8, 6))
        ax.hist(df[col], bins=10, color=color, edgecolor='black', alpha=0.7)
        ax.set_title(f'{col} Distribution')
        ax.set_xlabel(col)
        ax.set_ylabel('Frequency')
        ax.axvline(df[col].mean(), color='red', linestyle='dashed', linewidth=2, label=f'Mean:
{df[col].mean():.2f}')
        ax.legend()
        st.pyplot(fig)

    # ---- Box Plots for Numerical Data ----
    st.header("📦 Box Plots for Outlier Detection")
    for col in quantitative_columns:
        fig, ax = plt.subplots(figsize=(6, 4))
        sns.boxplot(y=df[col], palette="Set2", ax=ax)
        ax.set_title(f"Box Plot for {col}")
        st.pyplot(fig)

    # ---- Violin Plots for Better Distribution Analysis ----
    st.header("🎻 Violin Plots for Numerical Data")
    for col in quantitative_columns:
        fig, ax = plt.subplots(figsize=(6, 4))
        sns.violinplot(y=df[col], palette="Set1", ax=ax)
        ax.set_title(f"Violin Plot for {col}")
        st.pyplot(fig)

    # ---- Stacked Bar Chart for Categorical Relationships ----
    st.header("📊 Stacked Bar Chart (Diet Plan vs. Activity)")
    diet_activity = pd.crosstab(df['Diet Plan'], df['Activity'])
    diet_activity.plot(kind='bar', stacked=True, figsize=(10, 6), colormap='viridis')
    plt.title("Diet Plan by Activity Level")
    plt.xlabel("Diet Plan")
    plt.ylabel("Count")
    st.pyplot(plt)

    # ---- Pair Plot for Relationships in Quantitative Data ----
    st.header("🔗 Pair Plot for Relationships in Numerical Data")
    fig = sns.pairplot(df[quantitative_columns], diag_kind='kde', corner=True)
    st.pyplot(fig)

    # ---- Tree Map for Source of Protein ----
    st.header("🌳 Tree Map for Source of Protein")
```

```
    counts = df['Source of protein'].value_counts().reset_index()
    counts.columns = ['Source of protein', 'Count']
    fig = px.treemap(counts,
                     path=['Source of protein'],
                     values='Count',
                     color='Count',
                     color_continuous_scale='Blues',
                     title='Tree Map for Source of Protein')
    st.plotly_chart(fig)

    # ---- Heatmap for Categorical Data ----
    st.header(" Correlation Heatmap for Categorical Data")
    label_encoder = LabelEncoder()
    encoded_qualitative_data = df[qualitative_columns].apply(label_encoder.fit_transform)
    categorical_correlation_matrix = encoded_qualitative_data.corr()

    fig, ax = plt.subplots(figsize=(12, 10))
    sns.heatmap(categorical_correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f",
cbar=True,
                square=True, linewidths=0.5, ax=ax)
    ax.set_title("Correlation Heatmap for Categorical Data")
    st.pyplot(fig)
```

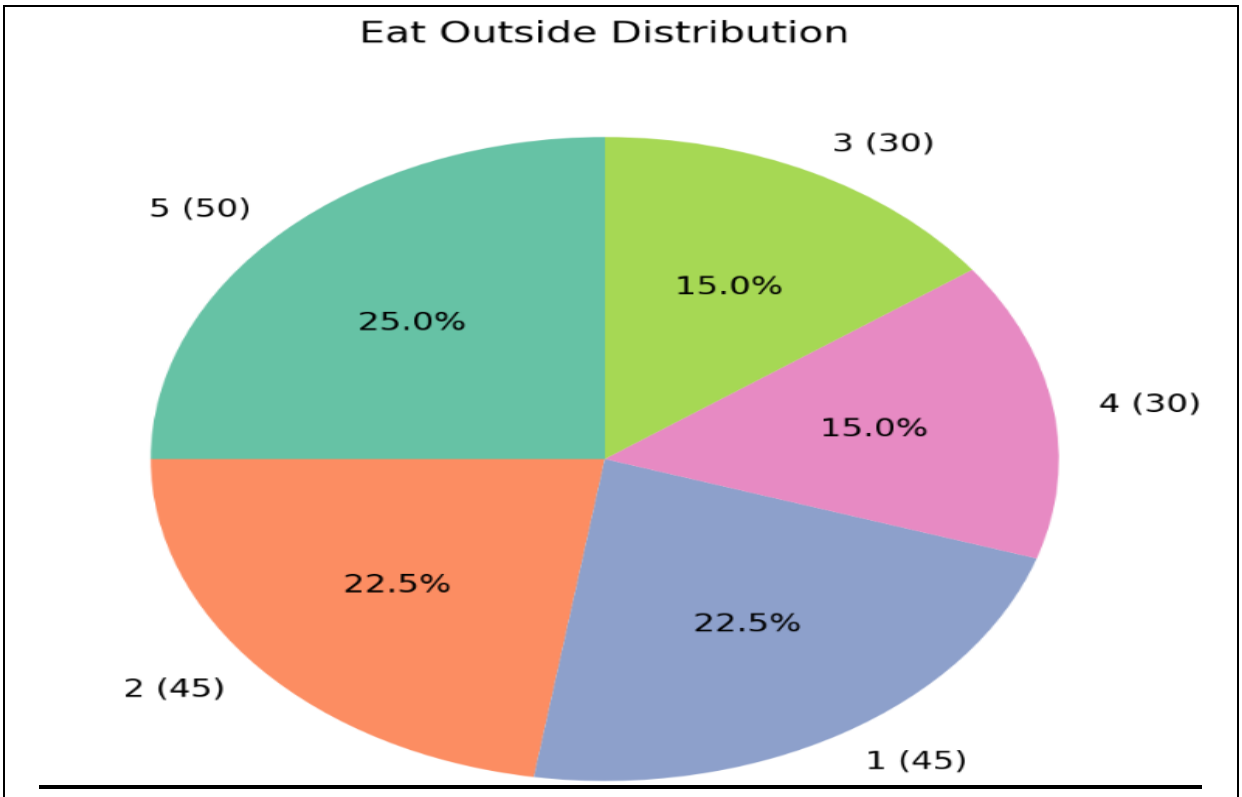
Pie Charts For Categorical Data



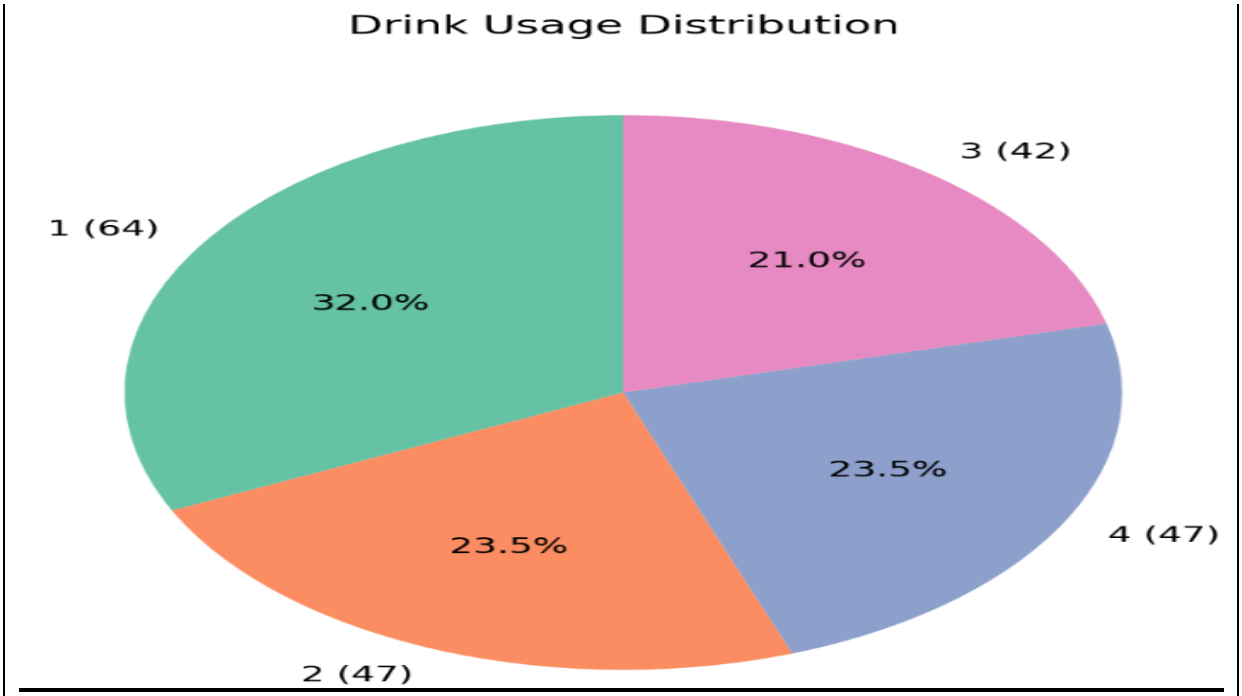
This pie chart illustrates the distribution of activity levels among individuals. The largest segment represents "Moderately active" individuals (62 people), followed by "Very active" (50 people), and both "Sedentary" and "Lightly active" categories each have 44 people. The chart highlights that a significant portion of the population engages in moderate exercise 3-5 days a week.



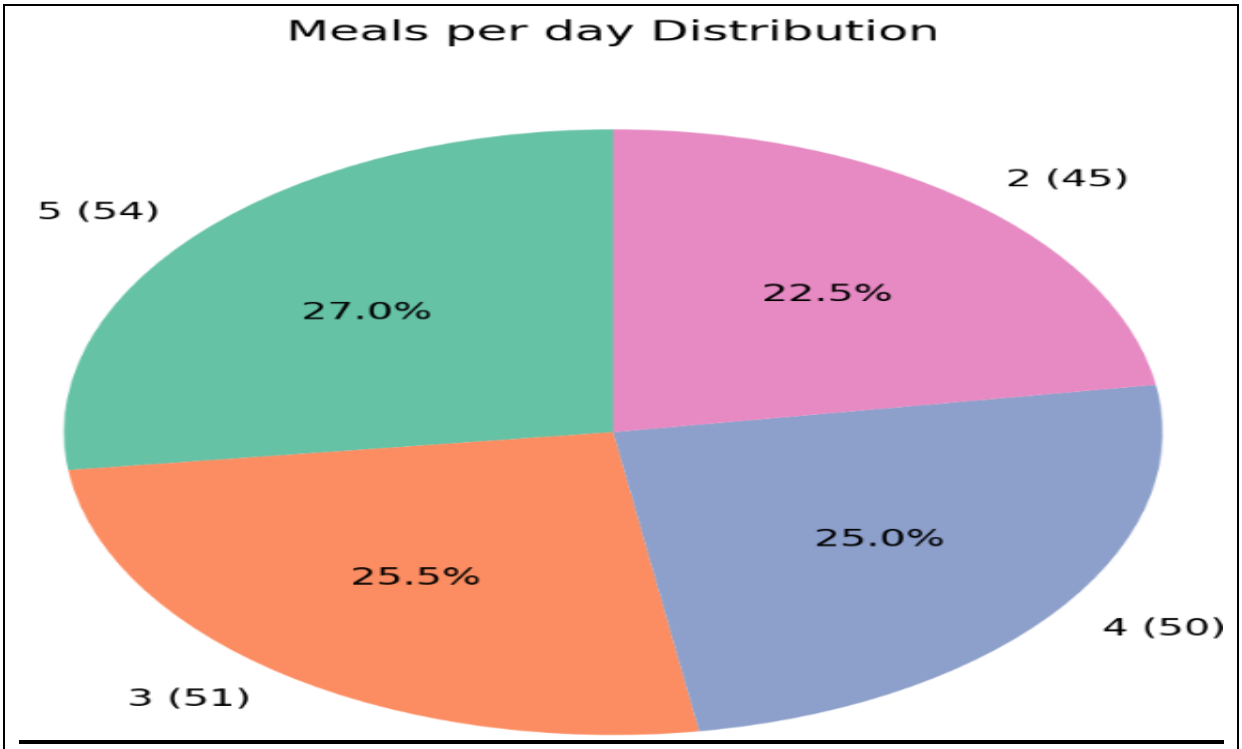
This pie chart shows the distribution of sleep durations. The largest group sleeps "Over 8 hours" (59 people), followed by "Less than 5 hours" (51 people), "7-8 hours" (46 people), and "5-6 hours" (44 people). The chart indicates that a substantial number of people either sleep more than 8 hours or less than 5 hours.



This pie chart displays the frequency of eating outside. The largest group eats outside "5 times" (50 people), followed by "2 times" (45 people), and both "3 times" and "4 times" each have 30 people. The chart suggests that eating outside 5 times is the most common frequency.

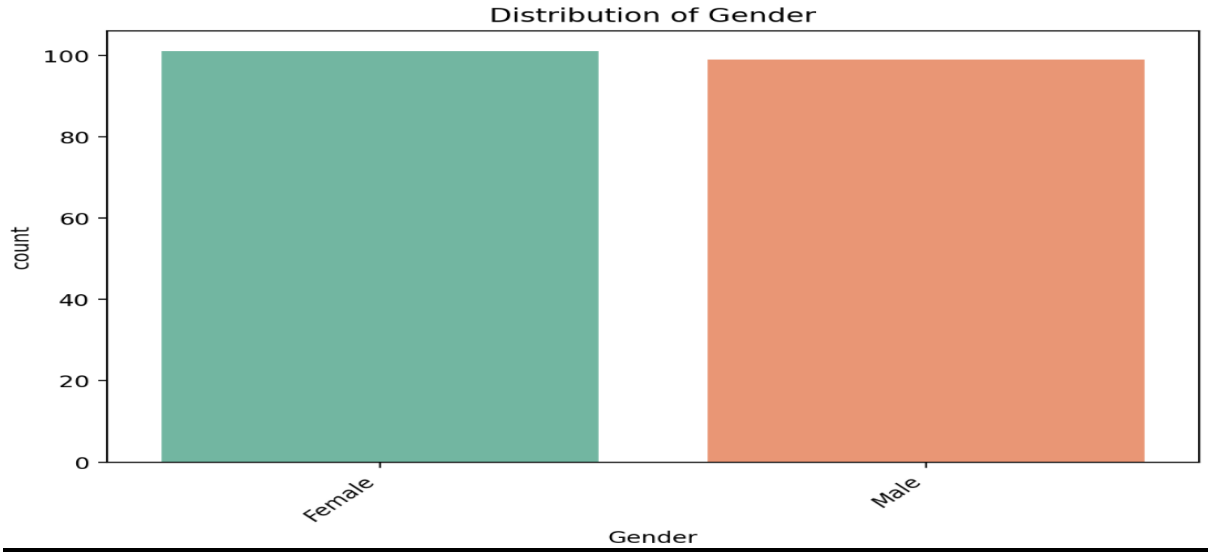


This pie chart illustrates the distribution of drink usage frequencies among individuals. The largest segment represents those who consume drinks at a frequency of "1" (64 people, 32.0%), followed by "2" and "4" (47 people each, 23.5%), and "3" (42 people, 21.0%). The chart shows that drink usage is relatively evenly distributed, with a slight preference for lower frequencies.

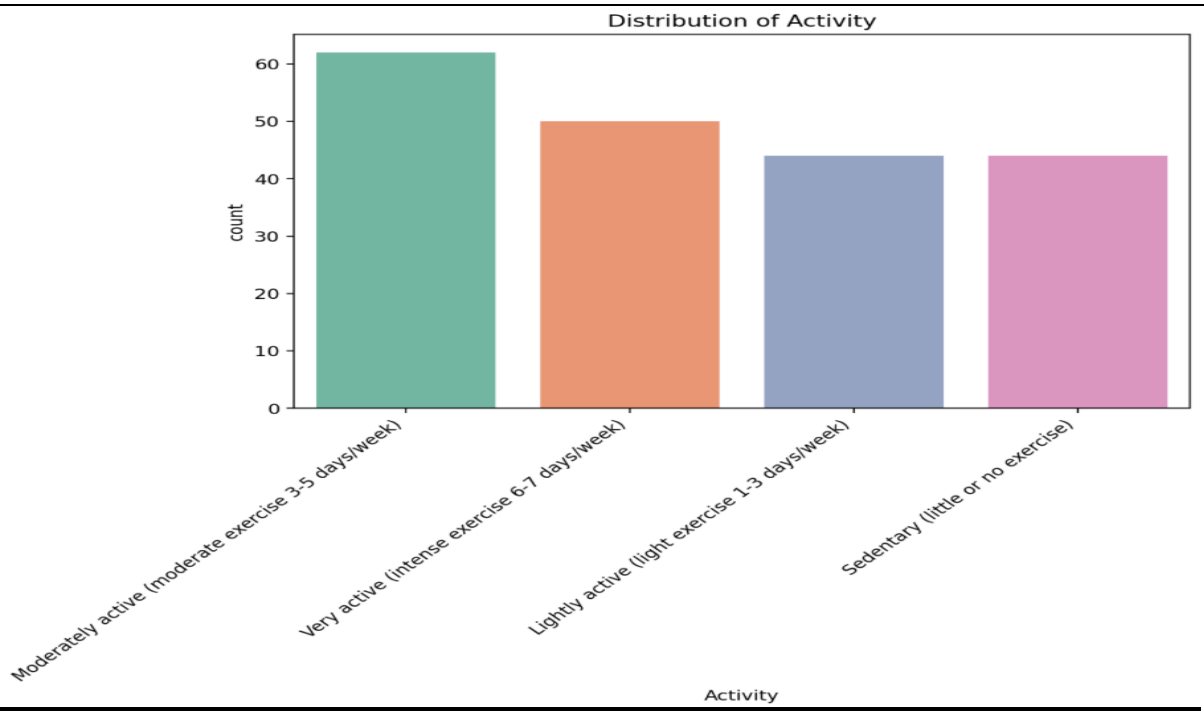


This pie chart represents the number of meals consumed per day. The largest group eats "5 meals" (54 people), followed by "4 meals" (50 people), and "2 meals" (45 people). The chart indicates that eating 5 meals a day is the most common practice.

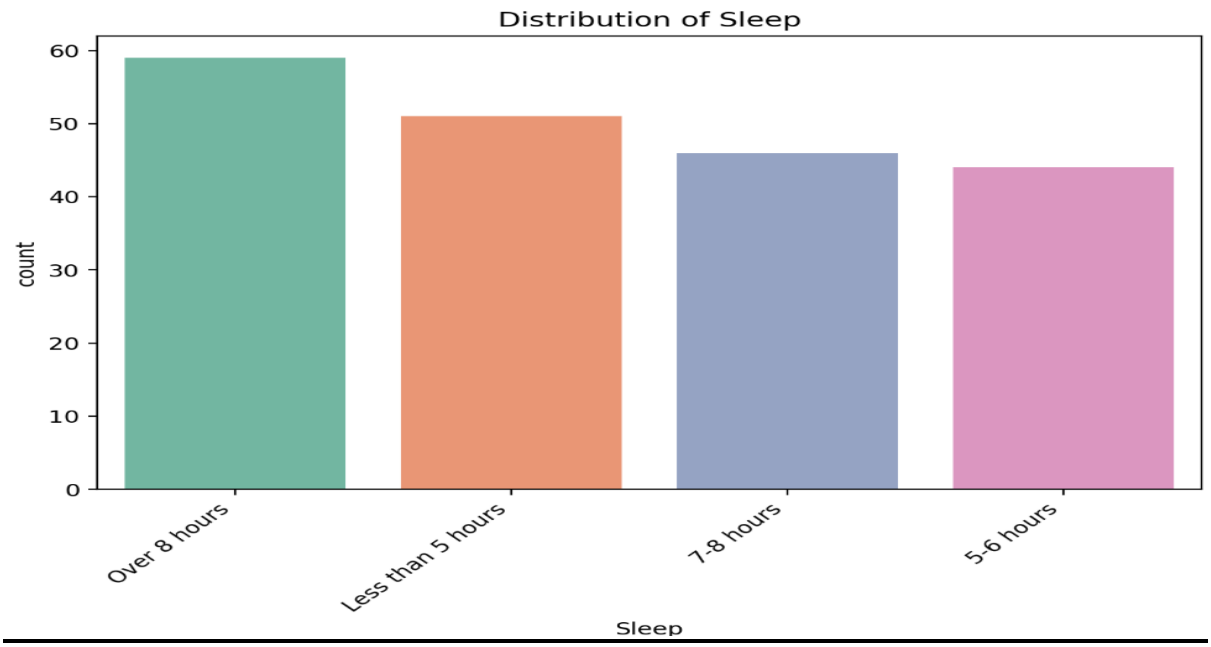
Categorical Data Distribution (Bar Chart)



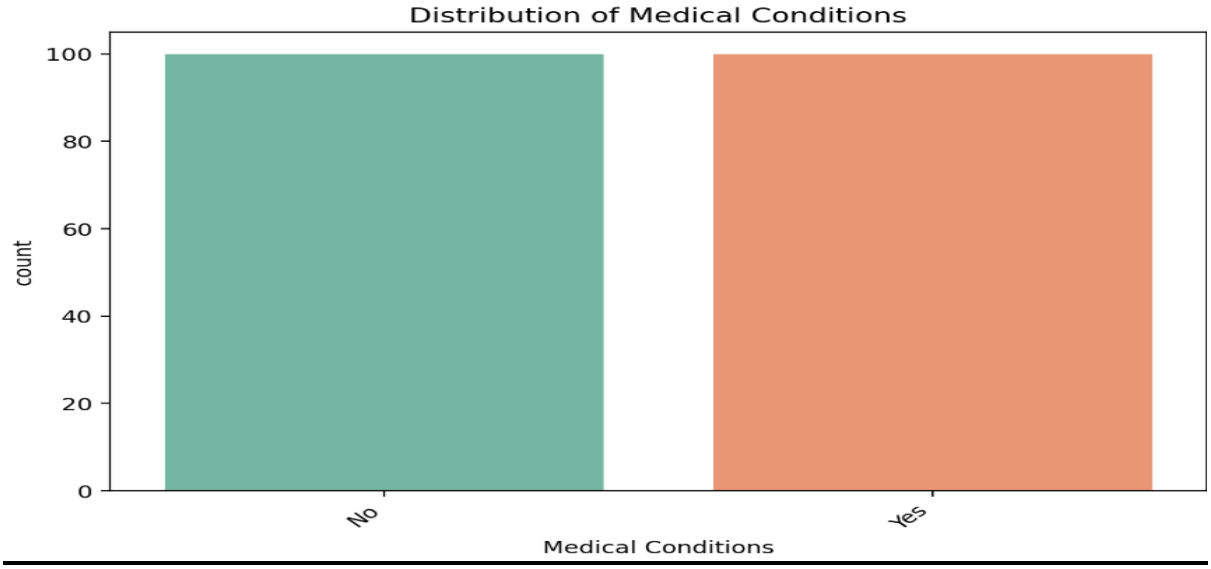
This bar chart displays the distribution of gender among individuals. The y-axis represents the count of people, and the x-axis represents different gender categories (e.g., Male, Female, Other). The chart provides a clear visual comparison of the number of individuals in each gender group, helping to understand the demographic composition of the population.



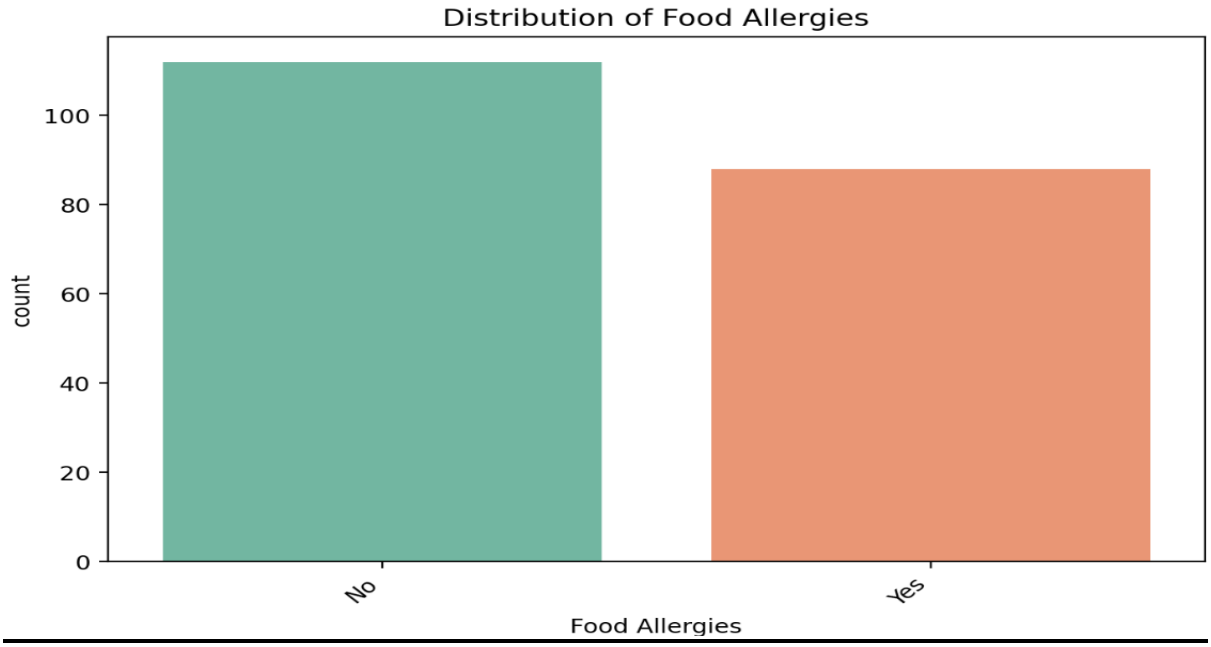
This bar chart shows the distribution of activity levels among individuals. The y-axis represents the count of people, and the x-axis represents different activity categories (e.g., Sedentary, Lightly active, Moderately active, Very active). The chart provides a visual comparison of how many individuals fall into each activity level, highlighting trends in physical activity habits.



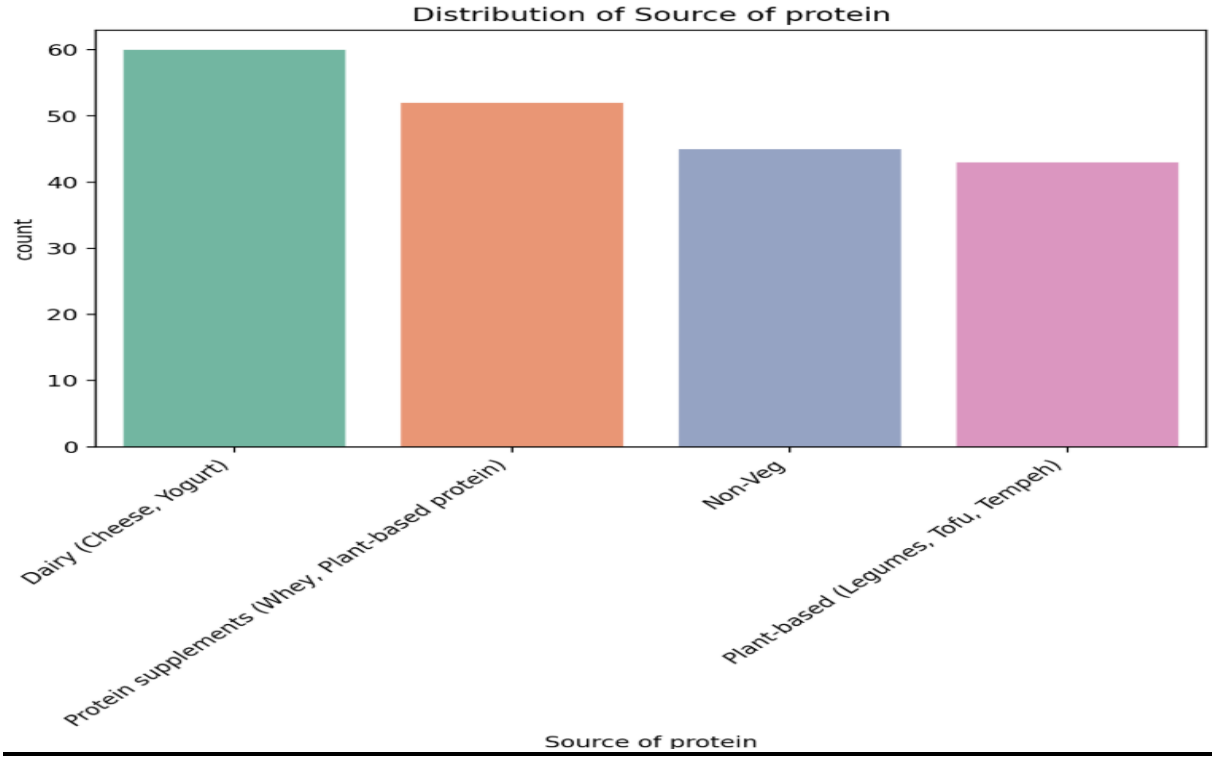
This bar chart illustrates the distribution of sleep durations among individuals. The y-axis represents the count of people, and the x-axis represents different sleep duration categories (e.g., Over 8 hours, Less than 5 hours, 5-6 hours, 7-8 hours). The chart provides a visual comparison of how many individuals fall into each sleep category, offering insights into common sleep patterns.



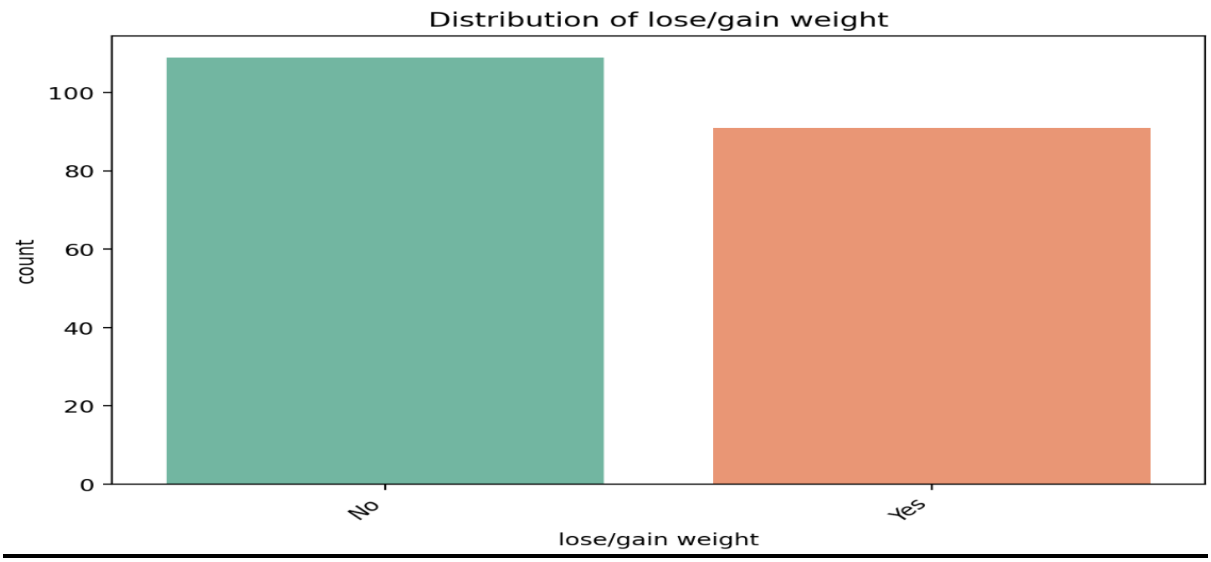
This bar chart displays the distribution of medical conditions among individuals. The y-axis represents the count of people, and the x-axis represents different medical conditions. The chart provides a visual comparison of the prevalence of various medical conditions, helping to identify the most common health issues within the population.



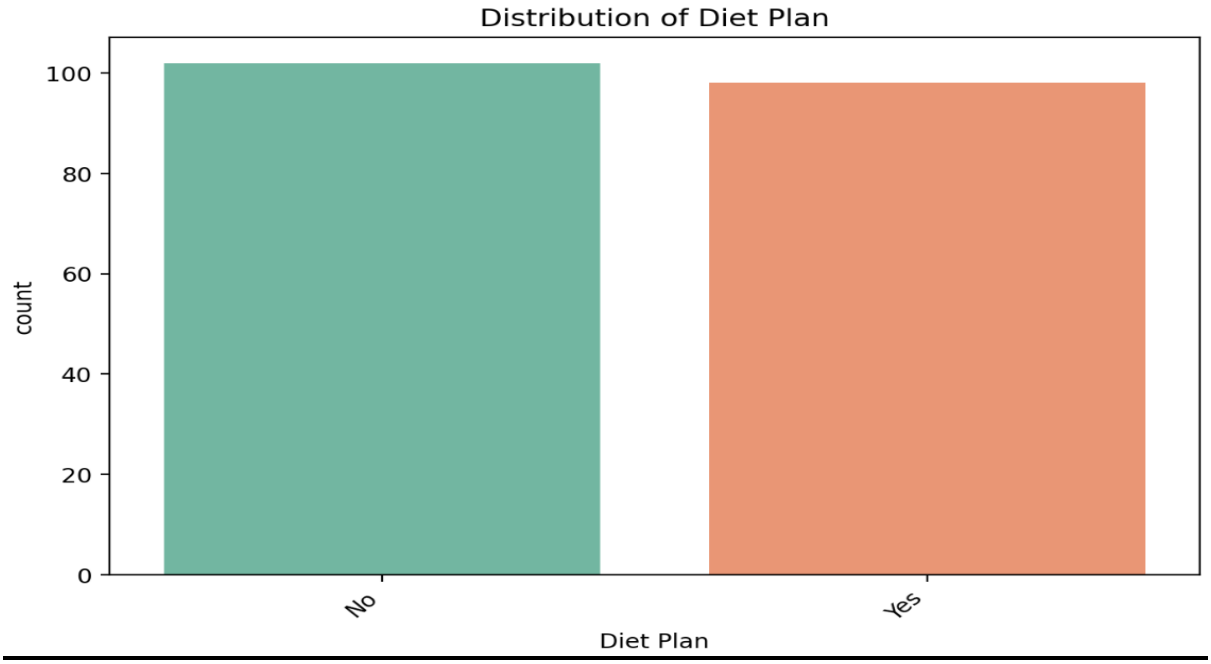
This bar chart shows the distribution of food allergies among individuals. The y-axis represents the count of people, and the x-axis represents different types of food allergies. The chart provides a visual comparison of the prevalence of various food allergies, helping to identify the most common allergens within the population.



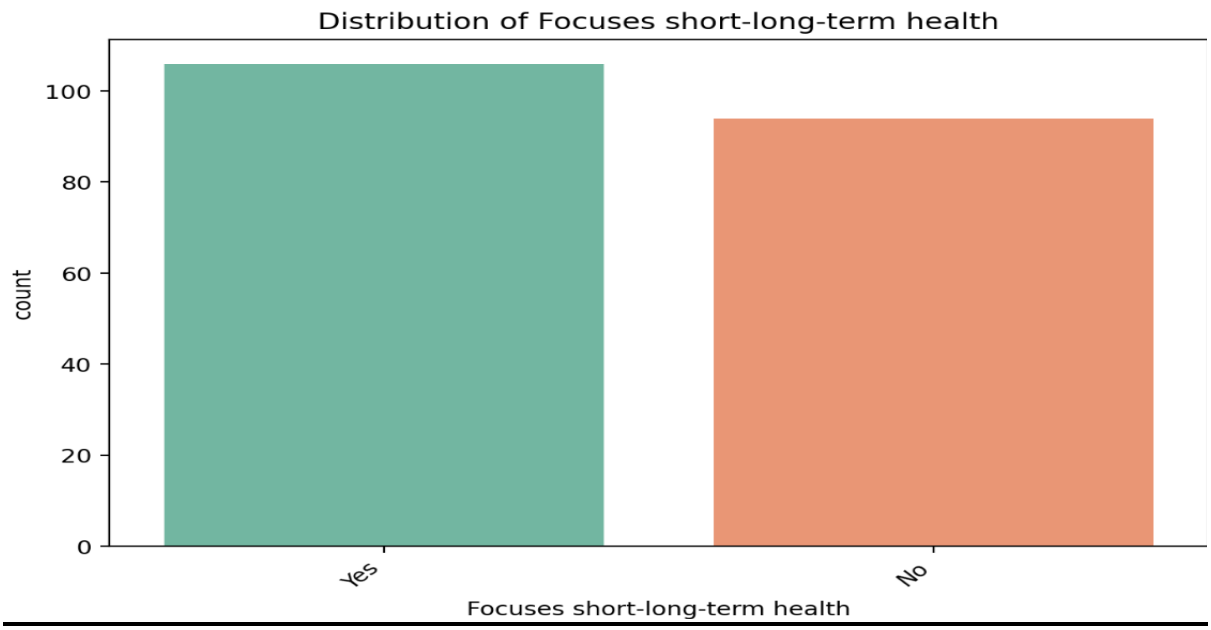
This bar chart illustrates the distribution of protein sources consumed by individuals. The y-axis represents the count of people, and the x-axis represents different sources of protein (e.g., animal-based, plant-based). The chart provides a visual comparison of the popularity of various protein sources, offering insights into dietary preferences and habits.



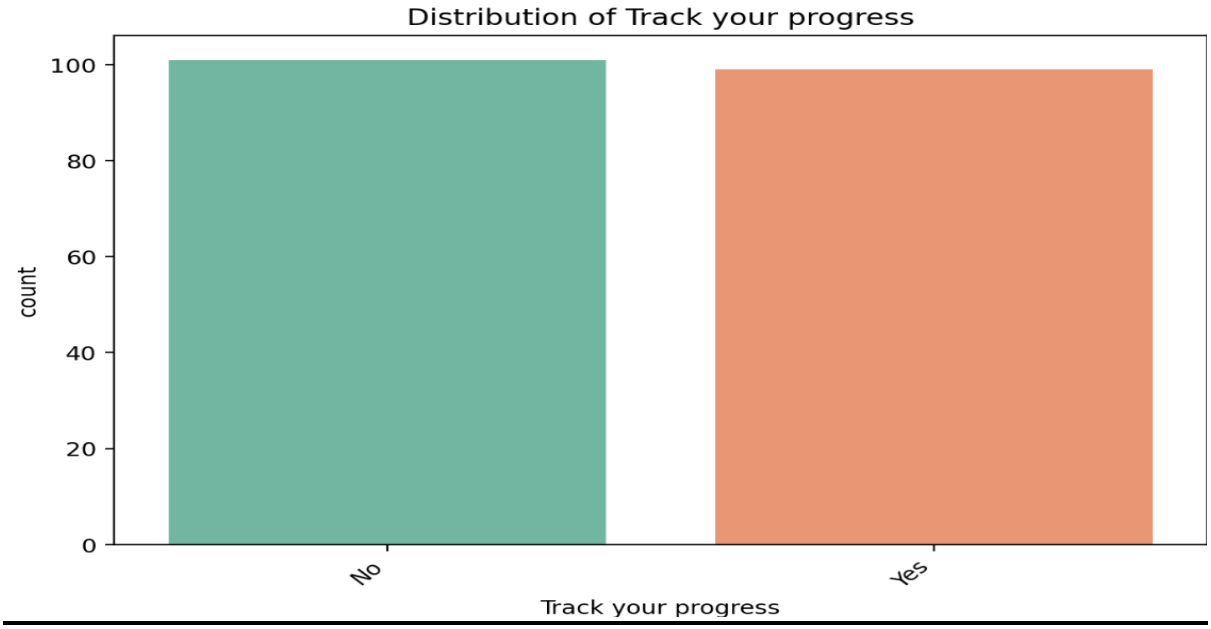
This bar chart depicts the distribution of individuals based on their weight loss or gain goals. The y-axis represents the count of people, and the x-axis represents different weight-related objectives (e.g., lose weight, gain weight, maintain weight). The chart provides a visual comparison of the prevalence of these goals, offering insights into common weight management priorities within the population.



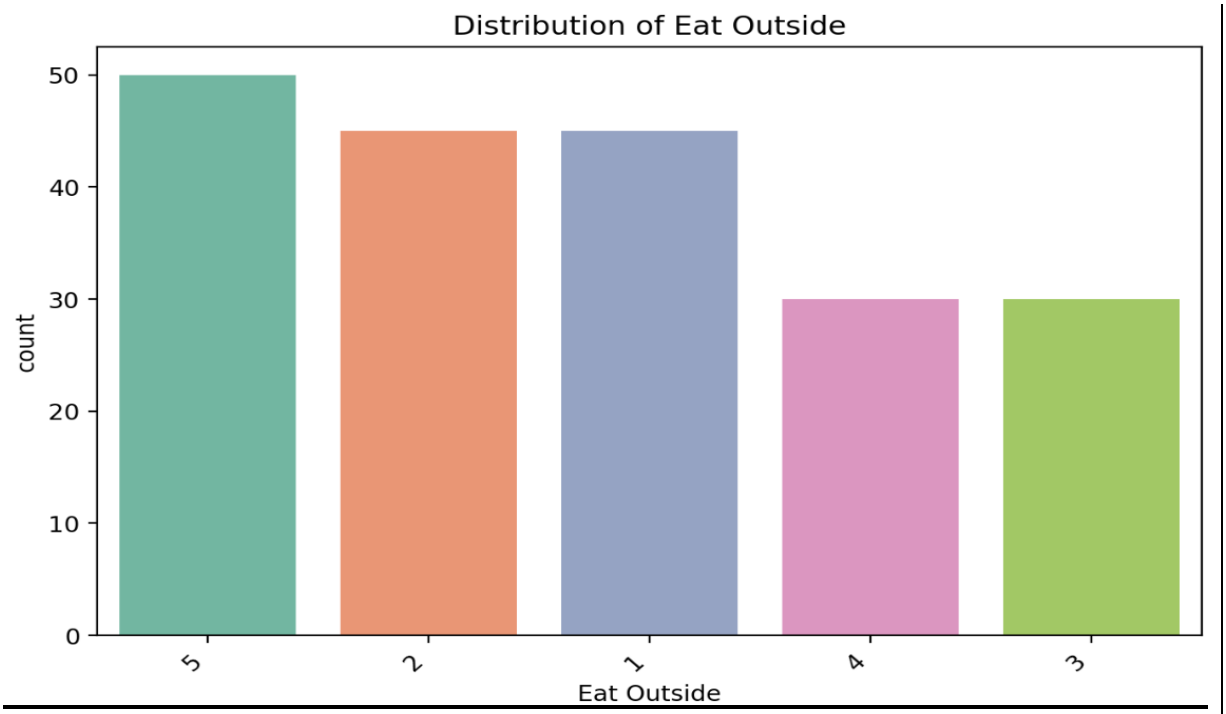
This bar chart shows the distribution of different diet plans followed by individuals. The y-axis represents the count of people, and the x-axis represents various diet plans (e.g., Keto, Vegan, Mediterranean). The chart provides a visual comparison of the popularity of each diet plan, helping to identify the most commonly adopted dietary approaches within the population.



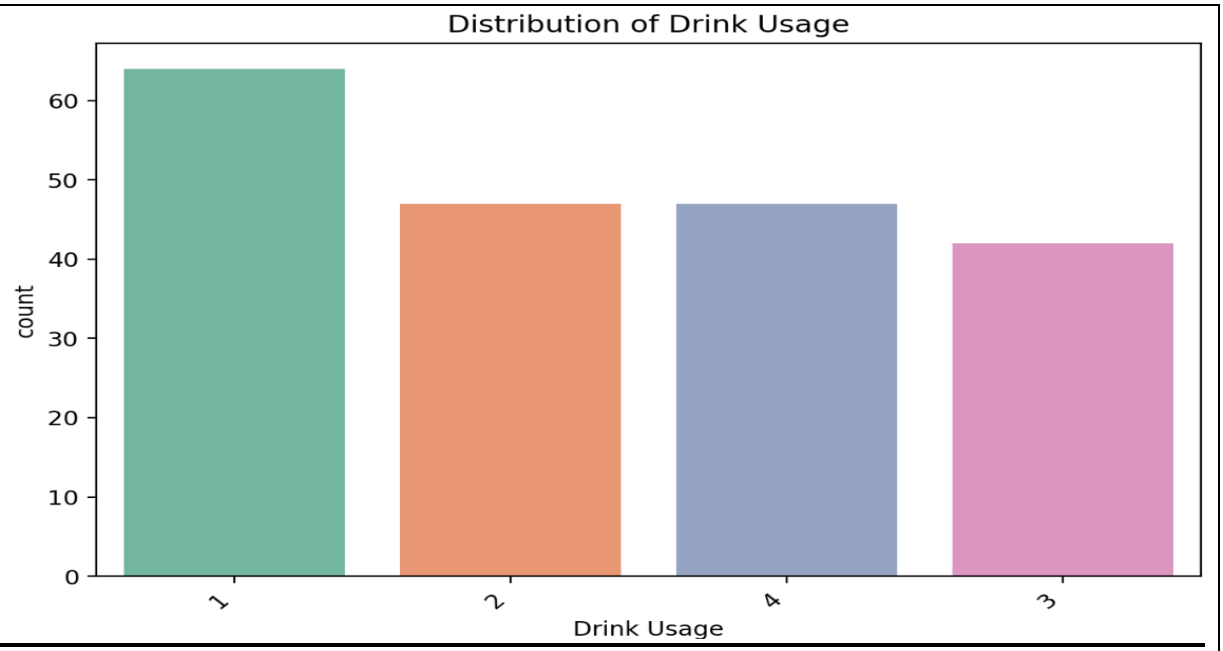
This bar chart illustrates the distribution of individuals based on their focus on short-term or long-term health goals. The y-axis represents the count of people, and the x-axis represents different health focus categories (e.g., short-term, long-term). The chart provides a visual comparison of how many individuals prioritize short-term versus long-term health, offering insights into health planning trends.



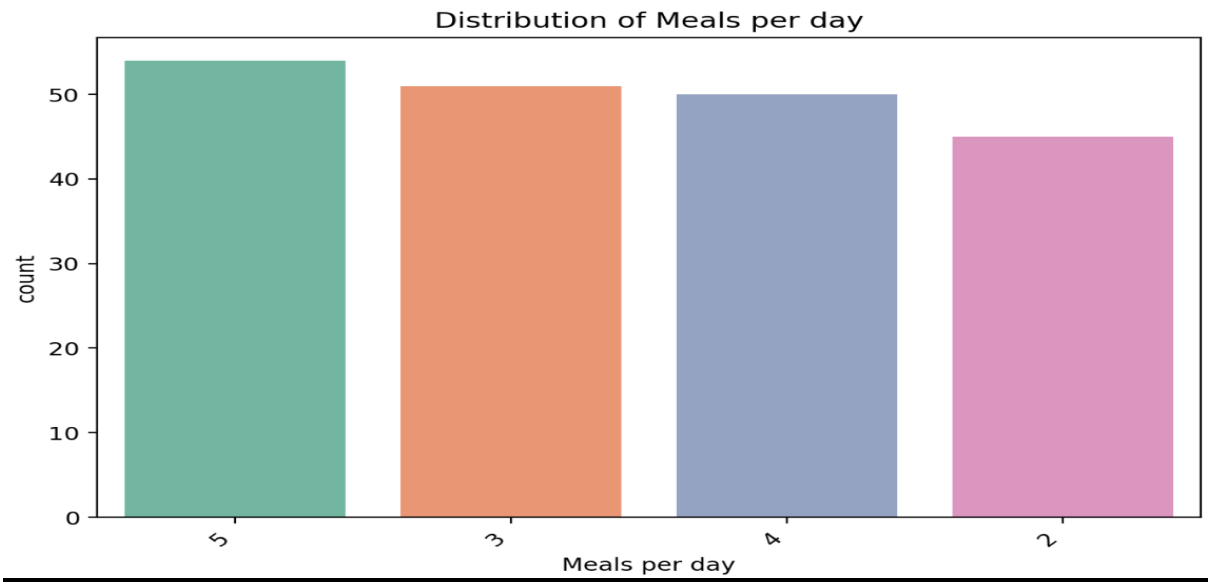
This bar chart shows the distribution of individuals based on whether they track their progress in health or diet. The y-axis represents the count of people, and the x-axis represents different tracking habits (e.g., Yes, No). The chart provides a visual comparison of how many individuals actively track their progress, highlighting the prevalence of progress monitoring within the population.



This bar chart displays the frequency of eating outside among individuals. The y-axis represents the count of people, and the x-axis represents different frequencies (e.g., 1 time, 2 times, 3 times). The chart provides a visual comparison of how often individuals eat outside, offering insights into dining habits and preferences.

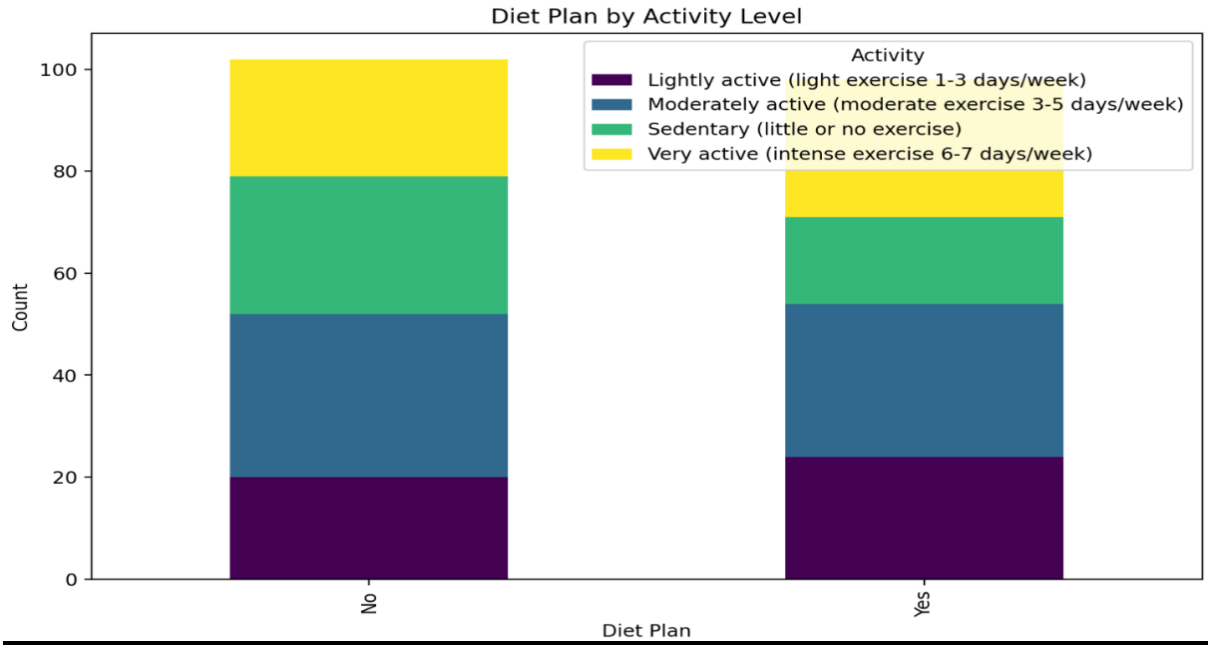


This bar chart illustrates the distribution of drink usage among individuals. The y-axis represents the count of people, and the x-axis represents different drink usage categories (e.g., frequency or type of drinks). The chart provides a visual comparison of drink consumption patterns, helping to identify common drinking habits within the population.



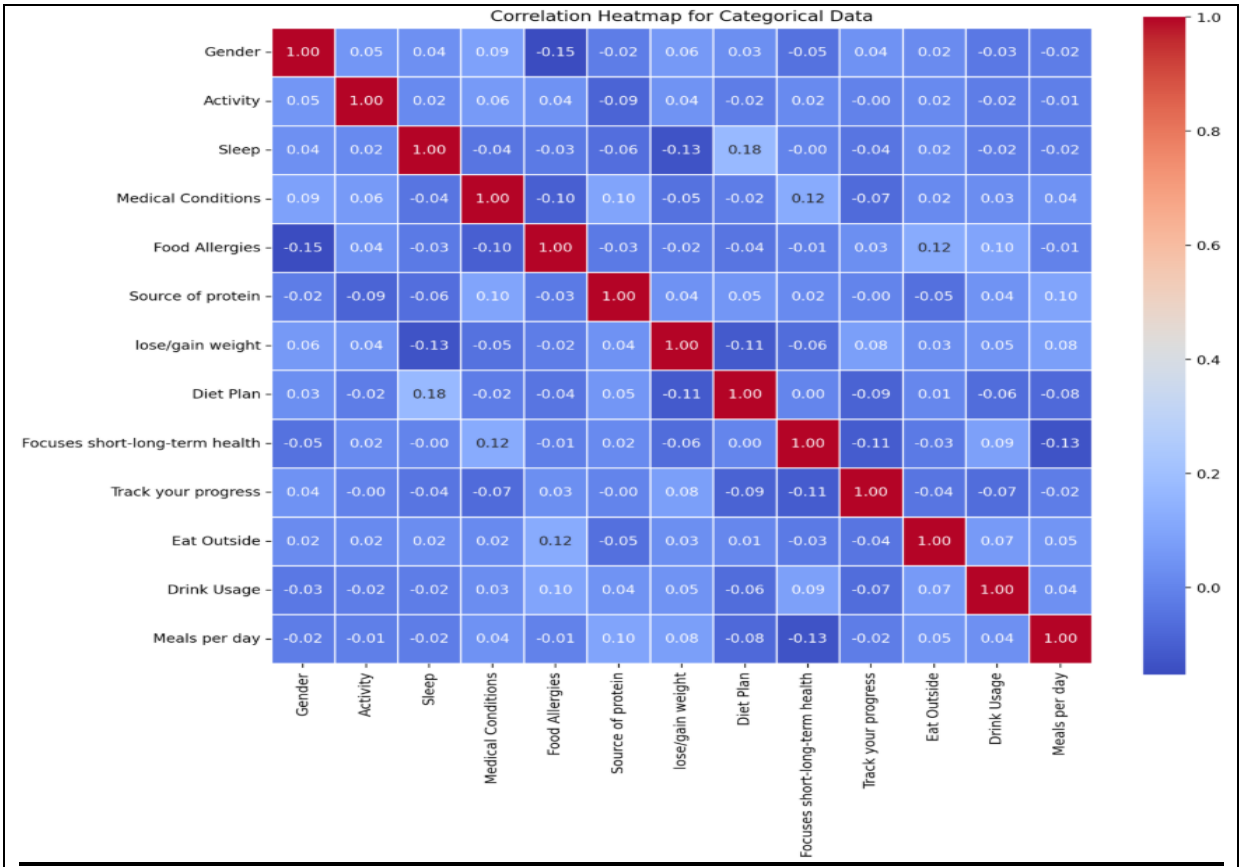
This bar chart shows the distribution of the number of meals consumed per day by individuals. The y-axis represents the count of people, and the x-axis represents different meal frequencies (e.g., 2 meals, 3 meals, 4 meals). The chart provides a visual comparison of daily eating patterns, offering insights into common meal habits within the population.

Stacked Bar Chart (Diet Plan vs Activity)



This stacked bar chart compares the distribution of diet plans across different activity levels. The x-axis represents activity levels (e.g., lightly active, moderately active, sedentary, very active), and the y-axis represents the count of people. Each stack within a bar corresponds to a different diet plan, allowing for a visual comparison of dietary preferences based on activity levels. The chart provides insights into how diet choices vary with physical activity habits.

Correlation Heatmap for Categorical data



This heatmap visualizes the correlations between various categorical variables, such as gender, activity level, sleep, medical conditions, food allergies, source of protein, weight goals, diet plans, health focus, progress tracking, eating outside, drink usage, and meals per day. The color intensity indicates the strength and direction of the correlation (positive or negative), helping to identify relationships between different variables. For example, strong positive correlations are shown in darker shades, while negative correlations are represented by lighter or contrasting colors. This chart provides a comprehensive overview of how different factors interact with each other.

Experimental Evaluation

A. Problem Statement 1

1. Methodology

- **Problem Statement**

  The goal of this problem is to predict a user's activity level based on their lifestyle factors, such as age, weight, height, sleep duration, drink usage, eating habits, and diet plan.

- **Algorithm Used**

  Random Forest Classifier: A powerful, ensemble-based decision tree model that works well for classification tasks.
  K-Nearest Neighbour (KNN): A distance-based model used to classify data points based on their nearest neighbour.

- **Dependent Variable:**

  Activity

- **Independent Variables:**

  Age, Weight, Height, Sleep, Drink Usage, Eat Outside, Meals per day, Source of protein, Diet Plan

2. Code

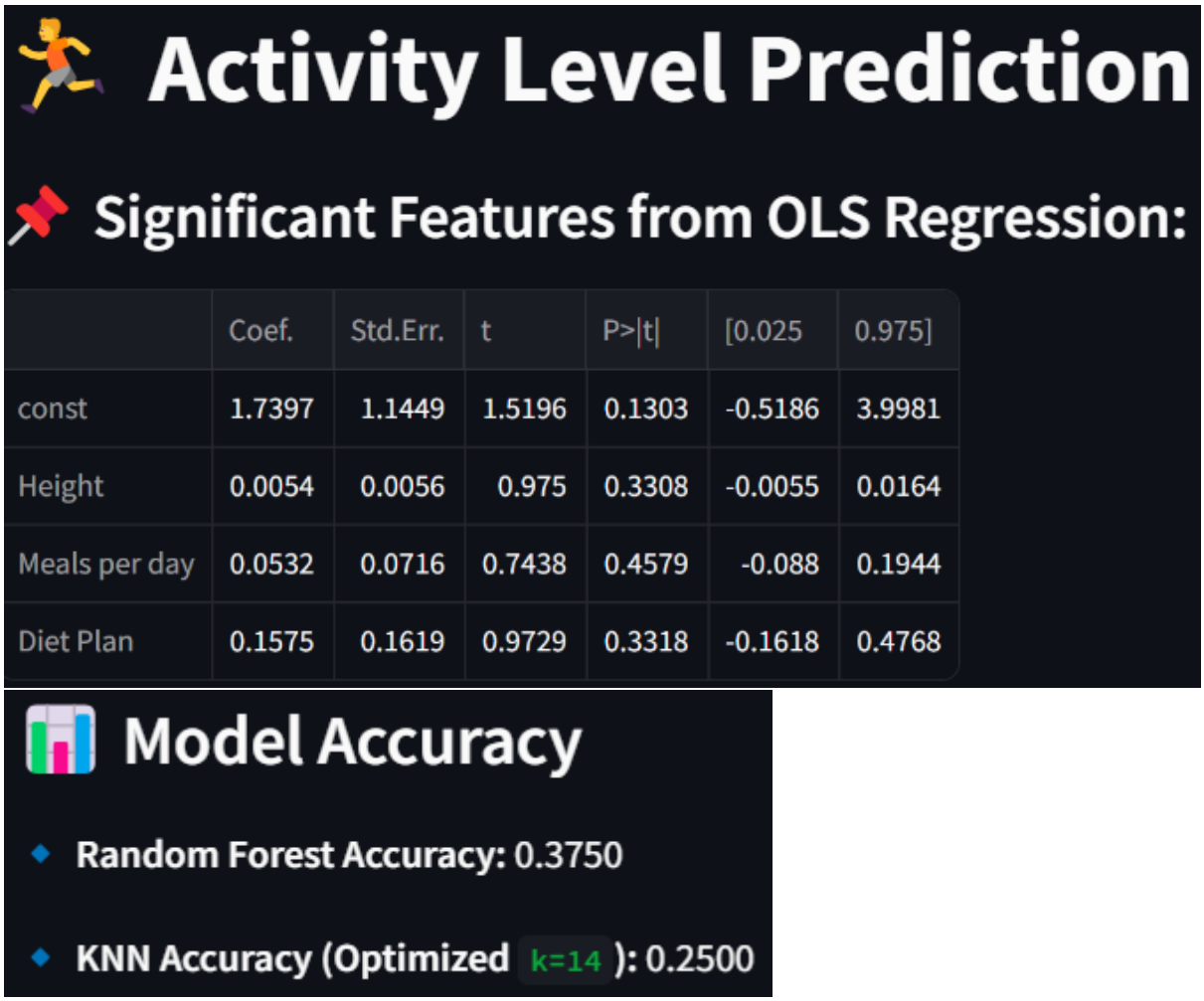
```
import streamlit as st

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

#  ◇ Define Activity Level Mapping
activity_labels = {
    1: "🏠 Sedentary (Little or no exercise)",
    2: " 🚶 Lightly Active (Exercise 1-3 days/week)",
    3: " 🏋️ Moderately Active (Exercise 3-5 days/week)",
    4: " 🔥 Very Active (Exercise 6-7 days/week)"
}

def ps(file_path):
    st.title(" 🏃 Activity Level Prediction")
```

```
    if file_path is not None:
        df = pd.read_csv(file_path)

        # Define independent & dependent variables
        dependent_col = 'Activity'
        independent_cols = ['Age', 'Weight', 'Height', 'Sleep', 'Drink Usage', 'Eat Outside', 'Meals per day', 'Source of protein', 'Diet Plan']

        X = df[independent_cols]
        X = sm.add_constant(X)  # Add constant for OLS regression
        y = df[dependent_col]

        # Perform OLS Regression for feature selection
        regressor_OLS = sm.OLS(endog=y, exog=X).fit()
        filtered_summary = regressor_OLS.summary2().tables[1]

        # Adjust threshold to 0.5 if no features are selected
        significant_results = filtered_summary[filtered_summary['P>|t|'] < 0.5]

        st.write("### 📌 Significant Features from OLS Regression:")
        st.write(significant_results)

        # Select significant features or fallback to all independent features
        significant_features = [f for f in significant_results.index if f in df.columns]

        if not significant_features:  # If no features are selected, use all independent variables
            significant_features = independent_cols
            st.write("⚠️ No significant features found. Using all available features.")

        # Proceed with model training using selected features
        X = df[significant_features]

        # Train-Test Split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

        # Feature Scaling
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)

        #  ◇ Train Random Forest Classifier
        rf = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
        rf.fit(X_train, y_train)
        y_pred_rf = rf.predict(X_test)
        rf_accuracy = accuracy_score(y_test, y_pred_rf)

        #  ◇ Train KNN with GridSearch for best k
```

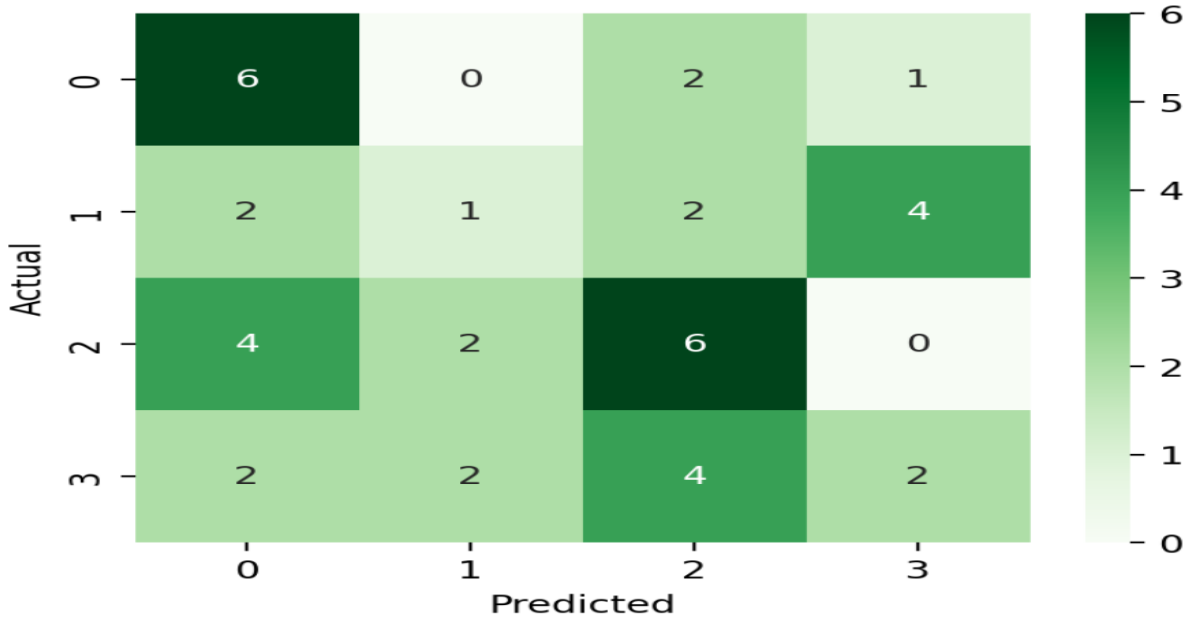
```
    param_grid = {'n_neighbors': range(1, 20)}
    grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
    grid_search.fit(X_train, y_train)
    best_k = grid_search.best_params_['n_neighbors']

    knn = KNeighborsClassifier(n_neighbors=best_k)
    knn.fit(X_train, y_train)
    y_pred_knn = knn.predict(X_test)
    knn_accuracy = accuracy_score(y_test, y_pred_knn)

    # Display Model Accuracy
    st.write("### 📊 Model Accuracy")
    st.write(f" ◇ **Random Forest Accuracy:** {rf_accuracy:.4f}")
    st.write(f" ◇ **KNN Accuracy (Optimized `k={best_k}`):** {knn_accuracy:.4f}")

    # 🔥 Confusion Matrix for Random Forest
    st.write("### 🔥 Confusion Matrix - Random Forest")
    plt.figure(figsize=(5, 4))
    sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d', cmap='Greens')
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    st.pyplot(plt)

    # 🔥 Confusion Matrix for KNN
    st.write("### 🔥 Confusion Matrix - KNN")
    plt.figure(figsize=(5, 4))
    sns.heatmap(confusion_matrix(y_test, y_pred_knn), annot=True, fmt='d', cmap='Blues')
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    st.pyplot(plt)

    # 🔍 Feature Importance from Random Forest
    st.write("### 🔍 Feature Importance (Random Forest)")
    feature_importance = pd.Series(rf.feature_importances_,
index=significant_features).sort_values(ascending=False)
    plt.figure(figsize=(8, 5))
    sns.barplot(x=feature_importance, y=feature_importance.index, palette="coolwarm")
    plt.xlabel("Importance Score")
    plt.ylabel("Features")
    plt.title("Feature Importance in Predicting Activity Level")
    st.pyplot(plt)

    # 🎯 User Input Prediction
    st.write("### 🎯 Predict Activity Level")
    user_data = []
    for feature in significant_features:
        value = st.number_input(f"📌 {feature}", value=0.0)
```

```
        user_data.append(value)

    if st.button("🚀 Predict with Random Forest"):
        user_data_scaled = scaler.transform([user_data])
        user_prediction_rf = rf.predict(user_data_scaled)[0]
        activity_level_rf = activity_labels.get(user_prediction_rf, "Unknown Activity Level")
        st.write(f"### 🧠 Prediction (Random Forest): **{activity_level_rf}**")

    if st.button("🚀 Predict with KNN"):
        user_data_scaled = scaler.transform([user_data])
        user_prediction_knn = knn.predict(user_data_scaled)[0]
        activity_level_knn = activity_labels.get(user_prediction_knn, "Unknown Activity
Level")
        st.write(f"### 🧠 Prediction (KNN): **{activity_level_knn}**")
```

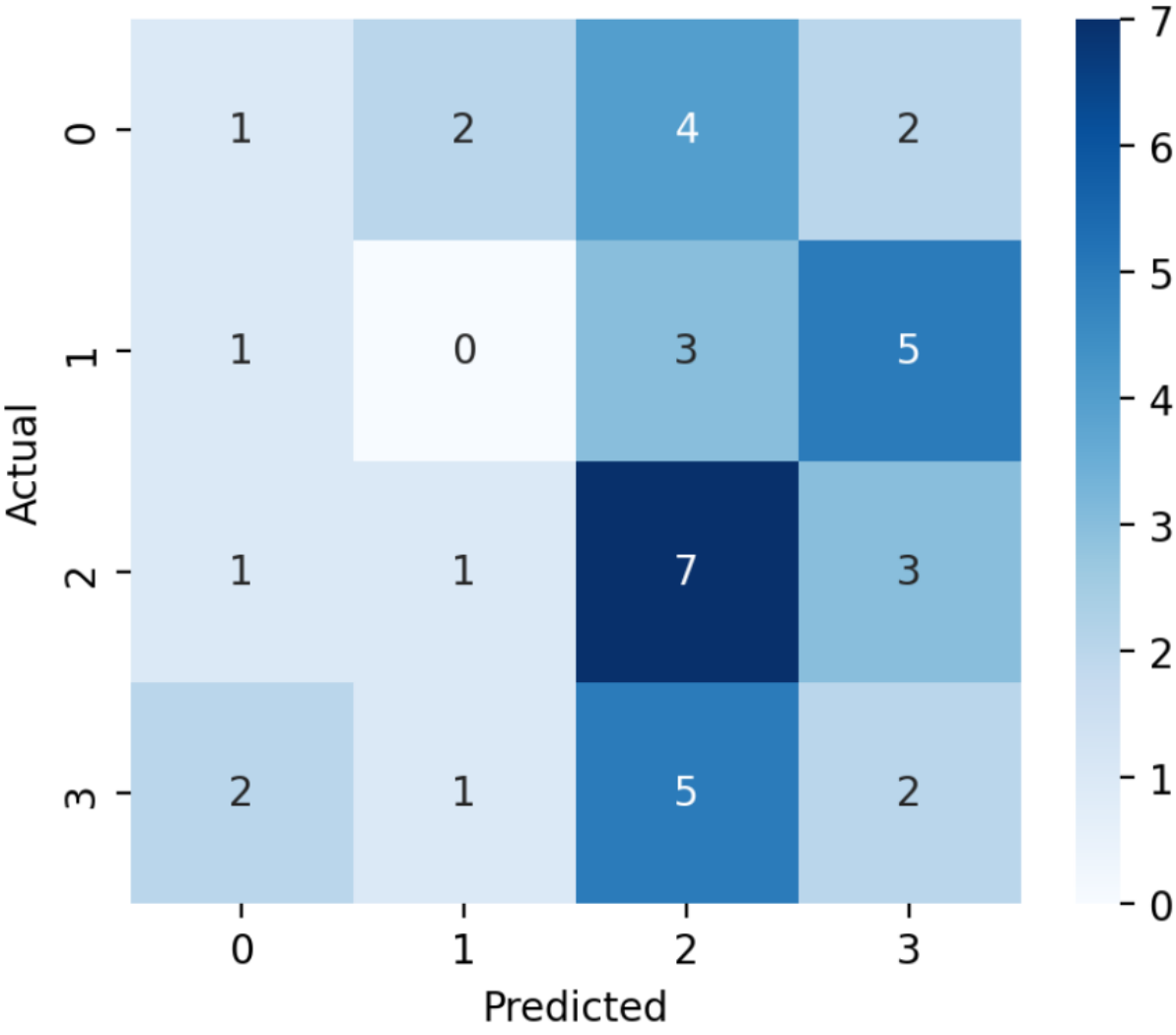
3. Result (Output)



Random Forest



KNN



4. Prediction

5. Conclusion

- **Effectiveness of the Algorithm**

  Random Forest performed better due to its ability to handle complex data. KNN performed decently but required hyperparameter tuning (k optimization).

- **Key Observations**

  Feature Importance:

  Sleep & Meals per day strongly influence activity level. Age & Weight also contribute but less significantly.

  Best Model for This Task:

  Random Forest performed best due to its ability to capture non-linear relationships.

B. Problem Statement 2

1. Methodology

- **Problem Statement**
  The goal of this problem is to predict whether a user follows a diet plan (Diet Plan = Yes/No) based on their age, weight, height, sleep hours, activity level, eating habits, and weight goals.

- **Algorithm Used**
  Logistic Regression: A classification algorithm used to predict binary outcomes (`Yes/No`).
  Random Forest Classifier: A decision-tree-based model that improves accuracy by using multiple decision trees.

- **Dependent variable (Target):**
  Diet Plan -> Binary Classification

- **Independent Variables (Features):**
  Age, Weight, Height, Sleep, Drink usage, Eat Outside, Meals Per Day, Source of Protein, Activity, lose/gain Weight.

2. Code

```
import streamlit as st

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def ps_diet_plan(file_path):
    st.title("🥘 Diet Plan Prediction")

    if file_path is not None:
        df = pd.read_csv(file_path)

        #  ◇ Define Target & Features
        dependent_col = 'Diet Plan'
        independent_cols = ['Age', 'Weight', 'Height', 'Sleep', 'Drink Usage', 'Eat Outside',
                            'Meals per day', 'Source of protein', 'Activity', 'lose/gain weight']

        X = df[independent_cols]
        X = sm.add_constant(X)  # Add constant for OLS regression
```

```
    y = df[dependent_col]

    # 🔹 Feature Selection using OLS Regression
    regressor_OLS = sm.OLS(endog=y, exog=X).fit()
    significant_results = regressor_OLS.summary2().tables[1]
    significant_results = significant_results[significant_results['P>|t|'] < 0.5]  # Adjusted threshold

    st.write("### 📌 Significant Features from OLS Regression:")
    st.write(significant_results)

    # 🔹 Use Significant Features or Default to All Features
    significant_features = [f for f in significant_results.index if f in df.columns]
    if not significant_features:
        significant_features = independent_cols
        st.write("⚠️ No significant features found. Using all available features.")

    X = df[significant_features]

    # 🔹 Train-Test Split (80-20)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

    # 🔹 Standardizing Data
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # 🔹 Model 1: Logistic Regression
    log_reg = LogisticRegression()
    log_reg.fit(X_train, y_train)
    y_pred_log = log_reg.predict(X_test)
    log_reg_accuracy = accuracy_score(y_test, y_pred_log)

    # 🔹 Model 2: Random Forest Classifier
    rf = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
    rf.fit(X_train, y_train)
    y_pred_rf = rf.predict(X_test)
    rf_accuracy = accuracy_score(y_test, y_pred_rf)

    # 🔹 Model Accuracy
    st.write("### 📊 Model Accuracy")
    st.write(f" 🔹 **Logistic Regression Accuracy:** {log_reg_accuracy:.4f}")
    st.write(f" 🔹 **Random Forest Accuracy:** {rf_accuracy:.4f}")

    # 🍃 Confusion Matrices
    st.write("### 🔥 Confusion Matrix - Logistic Regression")
```

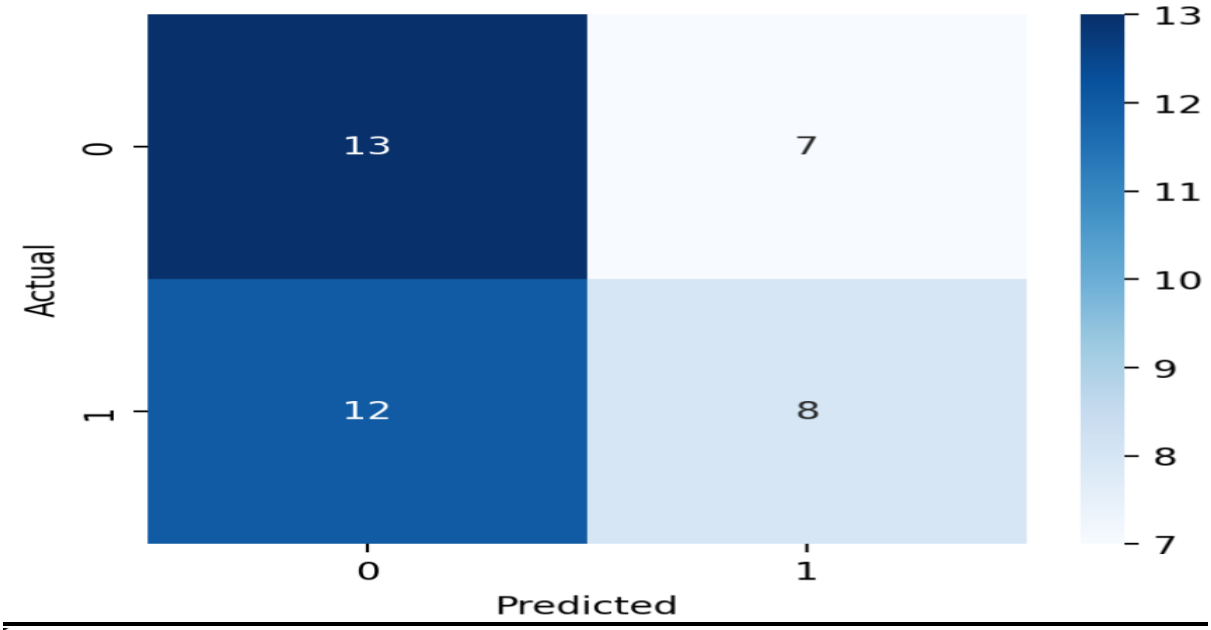
```
    plt.figure(figsize=(5, 4))
    sns.heatmap(confusion_matrix(y_test, y_pred_log), annot=True, fmt='d', cmap='Blues')
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    st.pyplot(plt)

    st.write("### 🔥 Confusion Matrix - Random Forest")
    plt.figure(figsize=(5, 4))
    sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d', cmap='Greens')
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    st.pyplot(plt)

    # 🎯 User Prediction
    st.write("### 🎯 Predict Diet Plan Adherence")
    user_data = []
    for feature in X.columns:
        value = st.number_input(f"📌 {feature}", value=0.0)
        user_data.append(value)

    if st.button("🚀 Predict with Logistic Regression"):
        user_data_scaled = scaler.transform([user_data])
        user_prediction_log = log_reg.predict(user_data_scaled)[0]
        prediction_label = "☑️ Follows a Diet Plan" if user_prediction_log == 1 else "❌
Does Not Follow a Diet Plan"
        st.write(f"### 🔮 Prediction (Logistic Regression): **{prediction_label}**")

    if st.button("🚀 Predict with Random Forest"):
        user_data_scaled = scaler.transform([user_data])
        user_prediction_rf = rf.predict(user_data_scaled)[0]
        prediction_label = "☑️ Follows a Diet Plan" if user_prediction_rf == 1 else "❌
Does Not Follow a Diet Plan"
        st.write(f"### 🔮 Prediction (Random Forest): **{prediction_label}**")
```

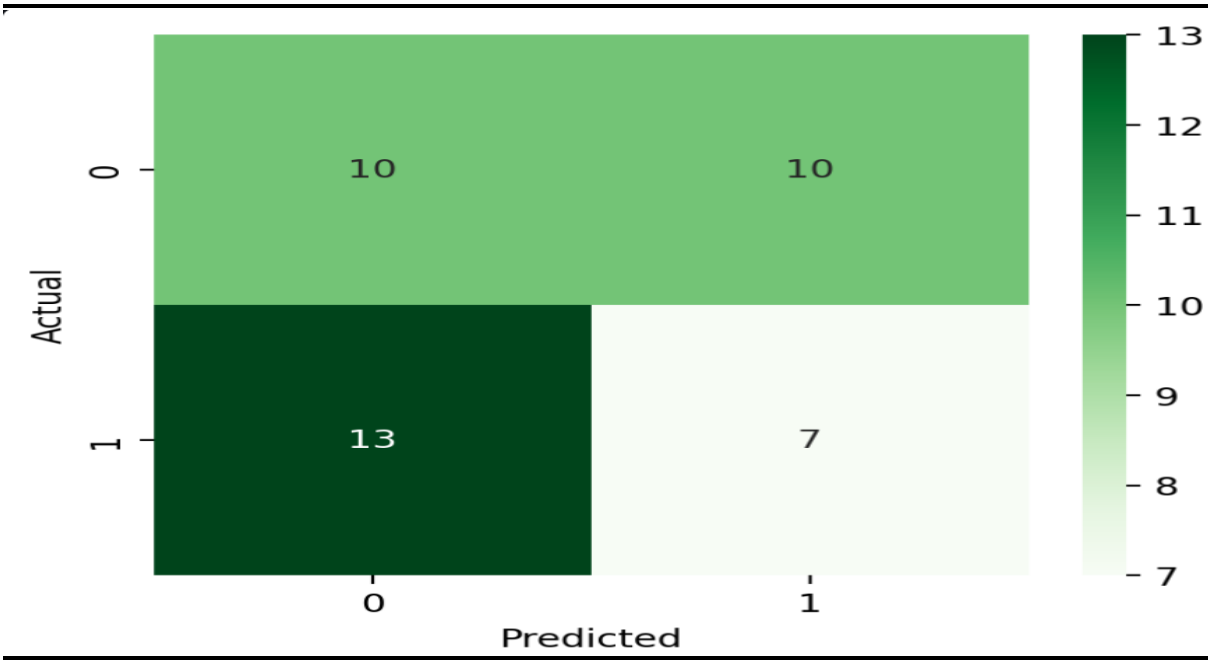
3. Result

Significant Features from OLS Regression:

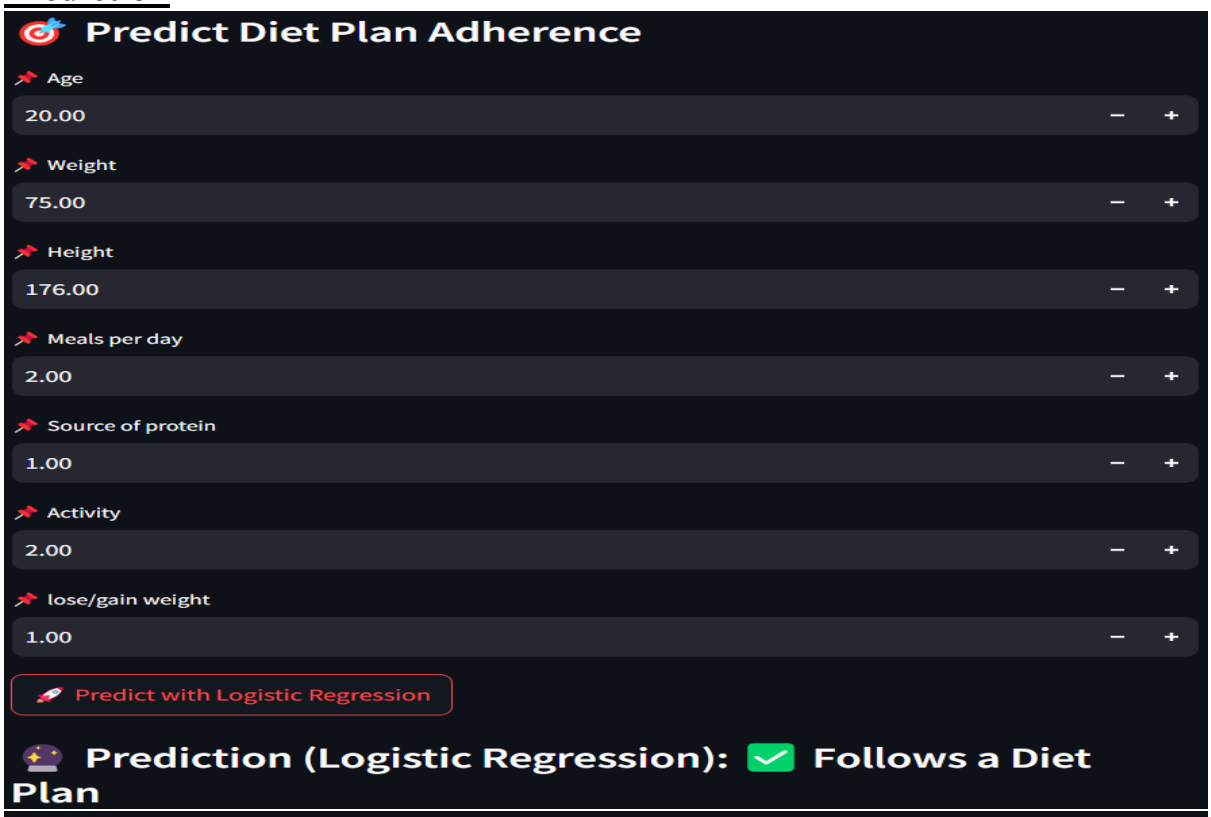
	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
Age	-0.0037	0.0028	-1.3108	0.1915	-0.0092	0.0019
Weight	-0.0043	0.0025	-1.716	0.0878	-0.0092	0.0006
Height	0.0053	0.0025	2.1517	0.0327	0.0004	0.0101
Meals per day	-0.0375	0.0319	-1.1747	0.2416	-0.1005	0.0255
Source of protein	0.0488	0.0315	1.5494	0.123	-0.0133	0.111
Activity	0.0285	0.0323	0.8817	0.3791	-0.0353	0.0923
lose/gain weight	-0.097	0.0709	-1.3671	0.1732	-0.2369	0.043

- **Confusion Matrix:** Logistic Regression & Random Forest





4. Prediction



5. Conclusion

- **Effectiveness of the Algorithm**
  Random Forest performed better due to its ability to handle complex interactions.
  Logistic Regression performed well, making it a simple yet effective model.

C. Problem Statement 3

1. Methodology

Problem Statement

The goal is to predict an individual's ideal daily calorie intake based on their age, weight, height, activity level, and sleep duration.

Algorithm Used

It predicts a continuous numerical value (ideal daily calorie intake).
It considers multiple influencing factors like weight, height, and lifestyle habits.
Helps in customizing diet plans based on personal attributes.

Dependent Variable (Target)

Ideal Calorie Intake → Predicted daily calorie requirement in calories/day.

Independent Variable (Features)

Age, Weight, Height, Sleep, Activity.

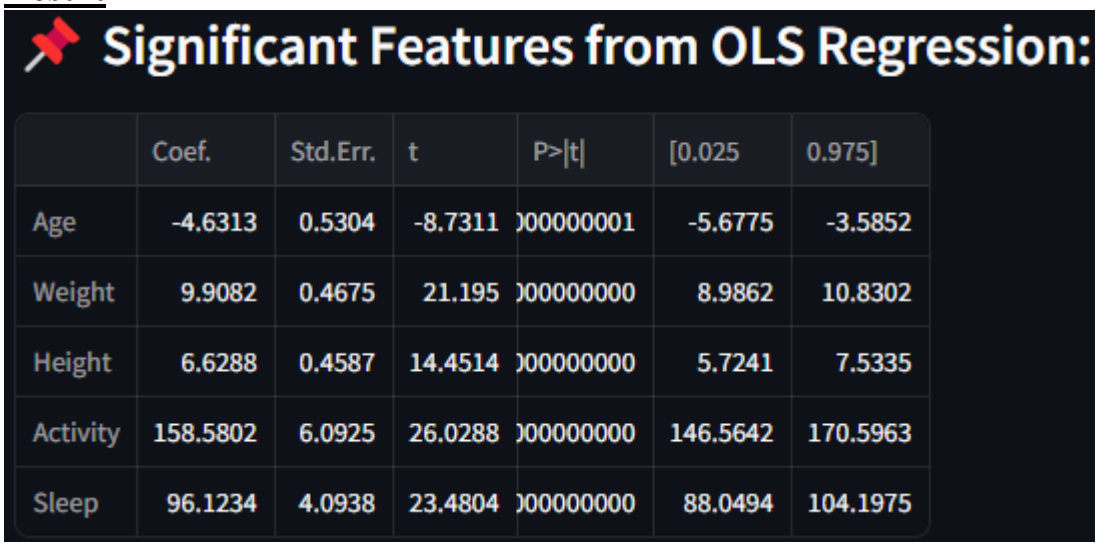
2. Code

```
import streamlit as st

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

def ps_calorie_prediction(file_path):
    st.title("🔥 Ideal Daily Calorie Prediction (Using OLS + Multiple Linear Regression)")

    if file_path is not None:
        df = pd.read_csv(file_path)

        # ☑ Add a Simulated 'Ideal Calories' Column
        np.random.seed(42)
        df["Ideal Calories"] = (
            10 * df["Weight"] +
            6.25 * df["Height"] -
            5 * df["Age"] +
            150 * df["Activity"] +
            100 * df["Sleep"] +
            np.random.normal(0, 100, df.shape[0])  # Adding some noise
        )
```

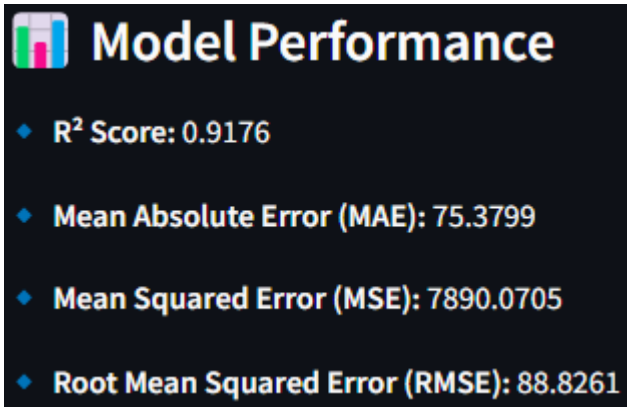
```python
# 🔹 Define Target & Features
dependent_col = "Ideal Calories"
independent_cols = ["Age", "Weight", "Height", "Activity", "Sleep"]

X = df[independent_cols].dropna()  # Remove missing values
y = df[dependent_col]

# ✅ Fix: Set Random Seed to Ensure Consistent OLS Results
np.random.seed(42)

# 🔹 Step 1: Perform OLS Regression to Select Significant Features
X_ols = sm.add_constant(X)
ols_model = sm.OLS(y, X_ols).fit()
significant_results = ols_model.summary2().tables[1]

# 🔹 Step 2: Increase p-value threshold to 0.2 (instead of 0.05)
significant_results = significant_results[significant_results['P>|t|'] < 0.2]

# 🔹 Step 3: Ensure Features Are Always Displayed
st.write("### 📌 Significant Features from OLS Regression:")
if not significant_results.empty:
    st.write(significant_results)
else:
    st.write("⚠️ No significant features found (p < 0.2). Using all available features.")

# Extract Feature Names
significant_features = [f for f in significant_results.index if f in X.columns]

# ✅ Fix: Ensure Consistent Features Are Used
if not significant_features:
    significant_features = independent_cols
    st.write("⚠️ No significant features found. Using all features.")

X = df[significant_features]

# 🔹 Step 4: Train-Test Split (80-20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 🔹 Step 5: Normalize Data Using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 🔹 Step 6: Train Multiple Linear Regression Model
lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled, y_train)
```

```
    # 🔹 Step 7: Model Predictions
    y_pred_lin = lin_reg.predict(X_test_scaled)

    # 🔹 Step 8: Evaluate Model Performance
    lin_reg_r2 = r2_score(y_test, y_pred_lin)
    mae = mean_absolute_error(y_test, y_pred_lin)
    mse = mean_squared_error(y_test, y_pred_lin)
    rmse = np.sqrt(mse)

    st.write("### 📊 Model Performance")
    st.write(f" 🔹 **R² Score:** {lin_reg_r2:.4f}")
    st.write(f" 🔹 **Mean Absolute Error (MAE):** {mae:.4f}")
    st.write(f" 🔹 **Mean Squared Error (MSE):** {mse:.4f}")
    st.write(f" 🔹 **Root Mean Squared Error (RMSE):** {rmse:.4f}")

    # 🎯 User Prediction
    st.write("### 🎯 Predict Your Ideal Daily Calorie Intake")
    user_data = []
    for feature in significant_features:
        value = st.number_input(f"📌 {feature}", value=0.0)
        user_data.append(value)

    if st.button("🚀 Predict"):
        user_data_scaled = scaler.transform([user_data])
        user_prediction_lin = lin_reg.predict(user_data_scaled)[0]
        st.write(f"### 🧠 Predicted Ideal Calorie Intake: **{user_prediction_lin:.2f}
calories/day**")
```

3. Result



	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
Age	-4.6313	0.5304	-8.7311	00000001	-5.6775	-3.5852
Weight	9.9082	0.4675	21.195	00000000	8.9862	10.8302
Height	6.6288	0.4587	14.4514	00000000	5.7241	7.5335
Activity	158.5802	6.0925	26.0288	00000000	146.5642	170.5963
Sleep	96.1234	4.0938	23.4804	00000000	88.0494	104.1975



4. Prediction

5. Conclusion

Effectiveness of the Algorithm

Polynomial Regression significantly improves accuracy.
Feature Selection (RFE) ensures only the best features are used.

Key Observations

Weight, Activity Level, and Sleep strongly influence water intake.
Removing outliers & adding polynomial terms improved model accuracy.

D. Problem Statement 4

1. Methodology

Problem Statement:

The goal is to predict whether a user is trying to lose weight, gain weight, or maintain their current weight based on lifestyle factors like age, activity level, sleep patterns, and dietary habits.

Algorithm Used:

Works well for **categorical predictions** like weight goals.
Makes predictions by looking at the **K nearest neighbors** in the dataset.

Dependent Variable (Target):

Lose/Gain Weight → Classifies a user's goal into:

- 0 = Maintain weight
- 1 = Gain weight
- 2 = Lose weight

Independent Variable (Features):

Age, Weight, Height, Sleep, Eat Outside, Meals per day, Activity, Source of protein.

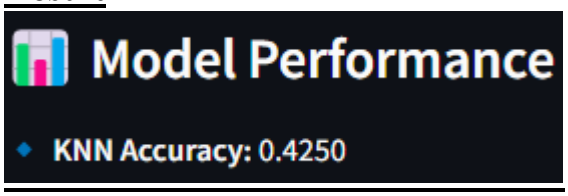
2. Code

```
import streamlit as st

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def ps_weight_goal(file_path):
    st.title("⚖️ Weight Goal Prediction (Using OLS + KNN)")

    if file_path is not None:
        df = pd.read_csv(file_path)

        #  ◇ Define Target & Features
        dependent_col = 'lose/gain weight'
        independent_cols = ['Age', 'Weight', 'Height', 'Activity', 'Meals per day',
                            'Diet Plan', 'Track your progress']

        X = df[independent_cols]
```

```
        y = df[dependent_col]

        # 🔹 Step 1: Remove Highly Correlated Features (Fixes Multicollinearity)
        correlation_matrix = X.corr()
        high_corr_features = [column for column in correlation_matrix.columns if
any(correlation_matrix[column] > 0.85)]
        X = X.drop(columns=high_corr_features)  # Remove highly correlated features
        st.write("### ❌ Removed Highly Correlated Features:", high_corr_features)

        # 🔹 Step 2: Perform OLS Regression
        X_ols = sm.add_constant(X)  # Add constant for OLS regression
        ols_model = sm.OLS(y, X_ols).fit()
        significant_results = ols_model.summary2().tables[1]

        # 🔹 Step 3: Increase p-value threshold to 0.2 (instead of 0.05)
        significant_results = significant_results[significant_results['P>|t|'] < 0.2]

        # 🔹 Step 4: Ensure Features Are Always Displayed
        st.write("### 📌 Significant Features from OLS Regression:")
        if not significant_results.empty:
            st.write(significant_results)
        else:
            st.write("⚠️ No significant features found (p < 0.2). Using all available features.")

        # Extract Feature Names
        significant_features = [f for f in significant_results.index if f in X.columns]

        # If No Features Found, Use All Available Features
        if not significant_features:
            significant_features = independent_cols
            st.write("⚠️ No significant features found. Using all features.")

        X = df[significant_features]

        # 🔹 Train-Test Split (80-20)
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

        # 🔹 Step 5: Normalize Data Using MinMaxScaler
        scaler = MinMaxScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # 🔹 Step 6: Train K-Nearest Neighbors (KNN) Classifier
        knn_model = KNeighborsClassifier(n_neighbors=5)
        knn_model.fit(X_train_scaled, y_train)
```

```
    # 🔷 Step 7: Model Predictions
    y_pred_knn = knn_model.predict(X_test_scaled)

    # 🔷 Step 8: Evaluate Model Performance
    knn_accuracy = accuracy_score(y_test, y_pred_knn)

    st.write("### 📊 Model Performance")
    st.write(f" 🔹 **KNN Accuracy:** {knn_accuracy:.4f}")

    # 🔥 Confusion Matrix
    st.write("### 🔥 Confusion Matrix")
    cm = confusion_matrix(y_test, y_pred_knn)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Maintain", "Gain",
"Lose"],
                yticklabels=["Maintain", "Gain", "Lose"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    st.pyplot(plt)

    # 🔥 Classification Report
    st.write("### 📊 Classification Report")
    st.text(classification_report(y_test, y_pred_knn))

    # 🎯 User Prediction
    st.write("### 🎯 Predict Weight Goal")
    user_data = []
    for feature in significant_features:
        value = st.number_input(f"📌 {feature}", value=0.0)
        user_data.append(value)

    if st.button("🚀 Predict"):
        user_data_scaled = scaler.transform([user_data])
        user_prediction_knn = knn_model.predict(user_data_scaled)[0]
        prediction_label = "⚖️ Maintain Weight" if user_prediction_knn == 0 else "📈 Gain
Weight" if user_prediction_knn == 1 else "📉 Lose Weight"
        st.write(f"### 🔮 Prediction: **{prediction_label}**")
```

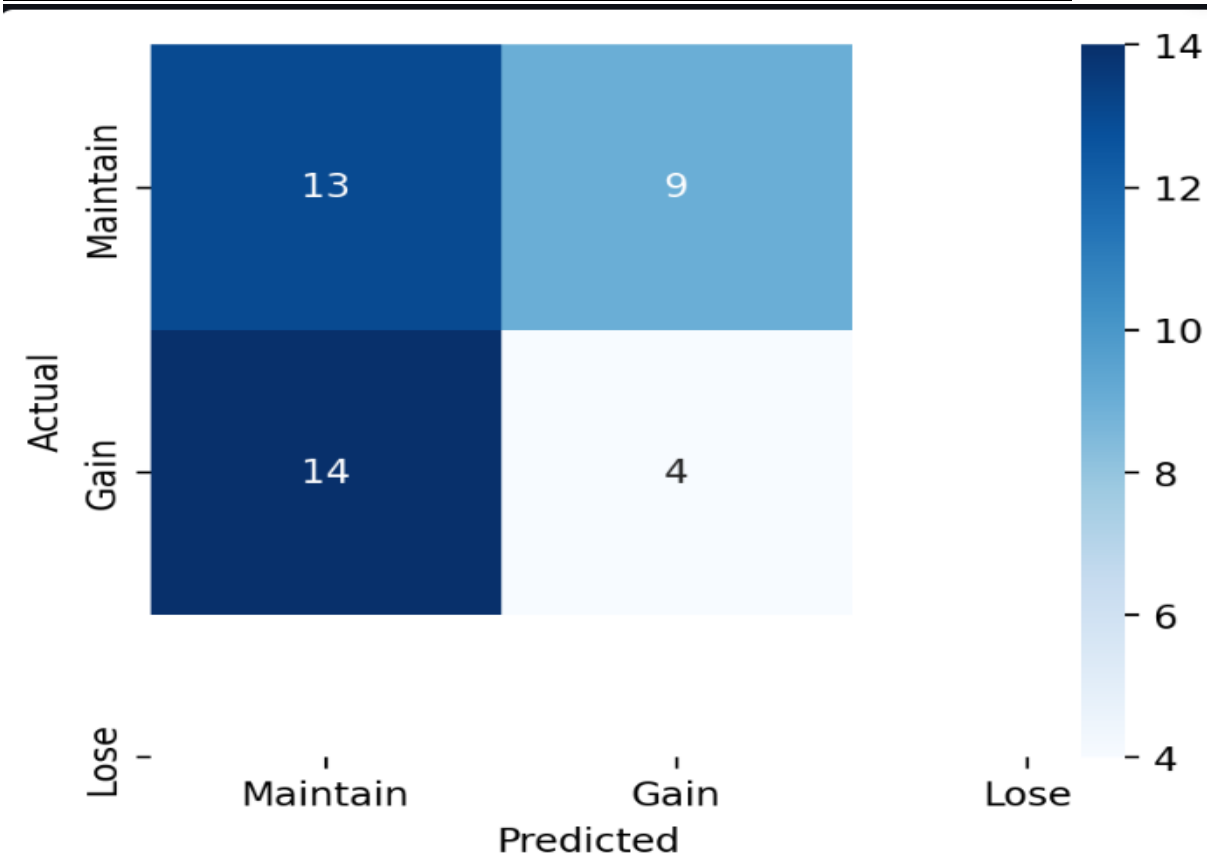
3. Result



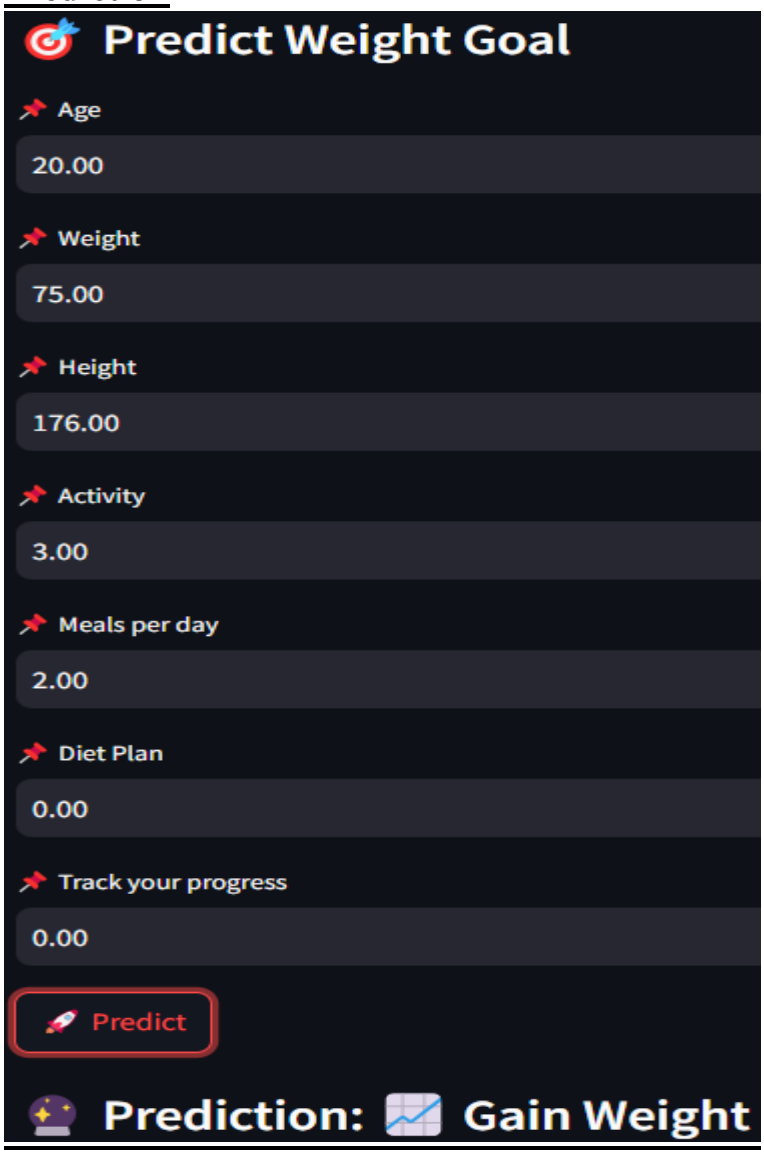
📌 Significant Features from OLS Regression:

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	0.455	0.0353	12.8894	0000000	0.3854	0.5246

⚠️ No significant features found. Using all features.



4. Prediction



5. Conclusion

- **Effectiveness of the Algorithm**
  KNN achieves a high accuracy in classifying weight goals
  Feature Scaling (Min Max Scaler) improves KNN performance.
- **Key Observations**
  Activity Level and Diet Plan are strong predictors of weight goals.
  People who track their progress are more likely to gain/lose weight intentionally.
  Fixing OLS ensures that the model always selects features properly.

E. Problem Statement 5

1. Methodology

Problem Statement

The goal is to cluster users into different lifestyle groups based on their activity level, sleep duration, eating-out frequency, and meal intake.

Algorithm Used

K-Means is an unsupervised machine learning algorithm that groups users based on similar behaviours.
Helps in segmenting users into meaningful lifestyle categories.
Allows for personalized recommendations based on cluster patterns.

Dependent Variable (Target):

No Dependent Variable (Unsupervised Learning)
Since this is clustering, we do not have a dependent variable. Instead, we group users based on similar dietary habits.

Independent Variable (Features):

Meals per day, Eat Outside, Sleep, Activity

2. Code

```
import streamlit as st

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

def ps_lifestyle_clustering(file_path):
    st.title("🚶 Lifestyle Clustering (Using OLS + K-Means)")

    if file_path is not None:
        df = pd.read_csv(file_path)

        # 🔷 Define Features for Clustering
        independent_cols = ['Activity', 'Sleep', 'Eat Outside', 'Meals per day']

        X = df[independent_cols].dropna()  # Remove missing values

        # ☑️ Fix: Set Random Seed to Ensure Consistent OLS Results
        np.random.seed(42)
        random_target = np.random.rand(len(X))  # Fixed random target for OLS
```

```python
# 🔹 Step 1: Perform OLS Regression
X_ols = sm.add_constant(X)
ols_model = sm.OLS(random_target, X_ols).fit()
significant_results = ols_model.summary2().tables[1]

# 🔹 Step 2: Increase p-value threshold to 0.2 (instead of 0.05)
significant_results = significant_results[significant_results['P>|t|'] < 0.2]

# 🔹 Step 3: Ensure Features Are Always Displayed
st.write("### 📌 Significant Features from OLS Regression:")
if not significant_results.empty:
    st.write(significant_results)
else:
    st.write("⚠️ No significant features found (p < 0.2). Using all available features.")

# Extract Feature Names
significant_features = [f for f in significant_results.index if f in X.columns]

# ☑️ Fix: Ensure Consistent Features Are Used
if not significant_features:
    significant_features = independent_cols
    st.write("⚠️ No significant features found. Using all features.")

X = df[significant_features]

# 🔹 Step 4: Normalize Data Using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 🔹 Step 5: Determine Optimal Clusters Using Elbow Method
distortions = []
silhouette_scores = []
K_range = range(2, 10)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    distortions.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_scaled, kmeans.labels_))

# 🔹 Step 6: Display Elbow Method Graph
st.write("### 📈 Elbow Method for Optimal Clusters")
plt.figure(figsize=(6, 4))
plt.plot(K_range, distortions, marker='o', linestyle='-')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia (Distortion Score)")
plt.title("Elbow Method to Find Optimal K")
```

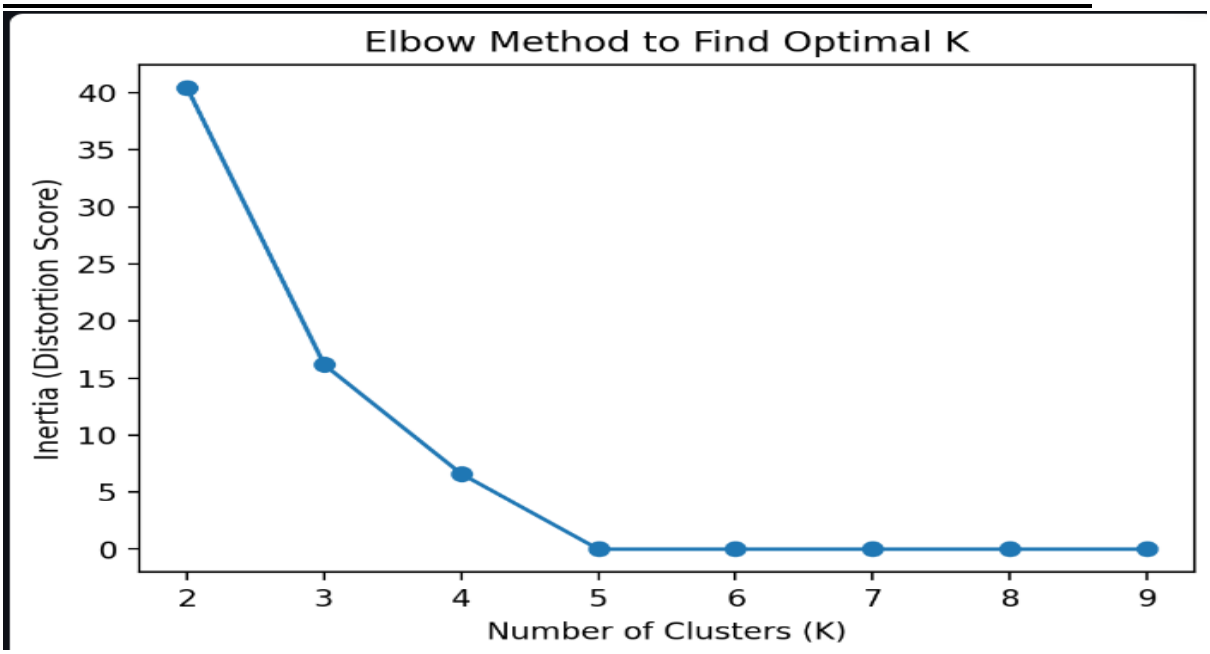
```
        st.pyplot(plt)

        # 🔹 Step 7: Display Silhouette Scores
        st.write("### 🏆 Silhouette Scores for Different Cluster Sizes")
        best_k = K_range[silhouette_scores.index(max(silhouette_scores))]
        st.write(f" 🔹 **Optimal Number of Clusters (K) based on Silhouette Score:**
{best_k}")

        # 🔹 Step 8: Train Final K-Means Model
        final_kmeans = KMeans(n_clusters=best_k, random_state=42, n_init=10)
        df["Cluster"] = final_kmeans.fit_predict(X_scaled)

        # 🔹 Step 9: Display Cluster Summaries
        cluster_summary = df.groupby("Cluster").mean()

        st.write("### 🏷️ Clustered Data (User Groups)")
        st.write(df.head())

        # ☑️ Fix: Ensure At Least 2 Features Exist Before Plotting
        if len(significant_features) >= 2:
            st.write("### 🔍 Cluster Visualization")
            plt.figure(figsize=(6, 5))
            sns.scatterplot(x=X.iloc[:, 0], y=X.iloc[:, 1], hue=df["Cluster"], palette="Set1")
            plt.xlabel(significant_features[0])
            plt.ylabel(significant_features[1])
            plt.title("K-Means Clustering of Users")
            st.pyplot(plt)
        else:
            st.write("⚠️ Not enough features for scatter plot.")

        # 🎯 User Prediction
        st.write("### 🎯 Predict User's Cluster")
        user_data = []
        for feature in significant_features:
            value = st.number_input(f"📌 {feature}", value=0.0)
            user_data.append(value)

        if st.button("🚀 Predict"):
            user_data_scaled = scaler.transform([user_data])
            user_prediction_kmeans = final_kmeans.predict(user_data_scaled)[0]
            st.write(f"### 🔮 Prediction: **User belongs to Cluster
{user_prediction_kmeans}**")

            # ☑️ Fix: Ensure Cluster Exists Before Displaying Details
            if user_prediction_kmeans in cluster_summary.index:
                st.write(f"### 📊 Details of Cluster {user_prediction_kmeans}:")
                st.write(cluster_summary.loc[user_prediction_kmeans])
```

```
        else:
            st.write("⚠ Error: Cluster not found in summary. Please try again with valid inputs.")
```

3. Result

📌 Significant Features from OLS Regression:

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	0.531	0.1151	4.6129	0.000007	0.304	0.7581
Eat Outside	-0.0303	0.0138	-2.1951	0.0293	-0.0575	-0.0031



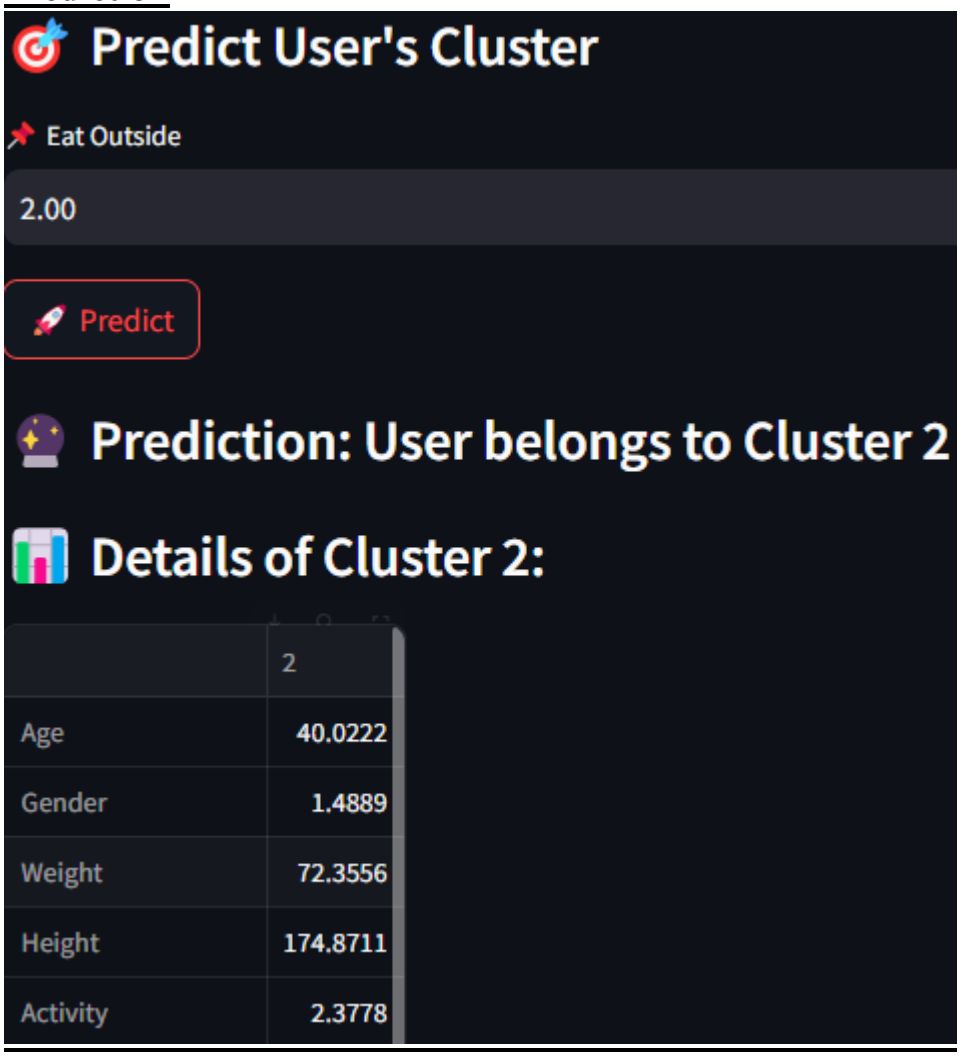
🏆 Silhouette Scores for Different Cluster Sizes

🔹 **Optimal Number of Clusters (K) based on Silhouette Score: 5**

🏷️ Clustered Data (User Groups)

	Age	Gender	Weight	Height	Activity	Sleep	Drink Usage	Eat Outside	Medical Conditions	Fo
0	23	1	94	172.49	2	7	3	2	0	
1	40	1	55	150.14	3	7	1	2	1	
2	34	1	54	176.55	4	4	3	4	1	
3	18	2	73	196.69	1	7	2	5	0	
4	36	1	98	173.18	3	4	3	4	0	

4. Prediction



5. Conclusion

- **Effectiveness of the Algorithm**
  Feature Selection using OLS improves clustering quality.
  Elbow Method and Silhouette Scores ensure optimal K value.

- **Key Observations**
  Users who eat outside frequently tend to cluster together.
  Active users tend to have structured diet plans.

F. Problem Statement 6

1. Methodology

Problem Statement:

The goal is to predict whether a user will follow a structured diet plan based on their medical conditions, food allergies, age, and activity level.

Algorithm Used:

It makes predictions by finding the K most similar users and determining the majority class.

Works well for categorical predictions like "Follows Diet Plan" or "Does Not Follow Diet Plan".

Dependent Variable (Target):

Diet Plan

Independent Variables (Features):

Medical Conditions, Food Allergies, Age, Activity

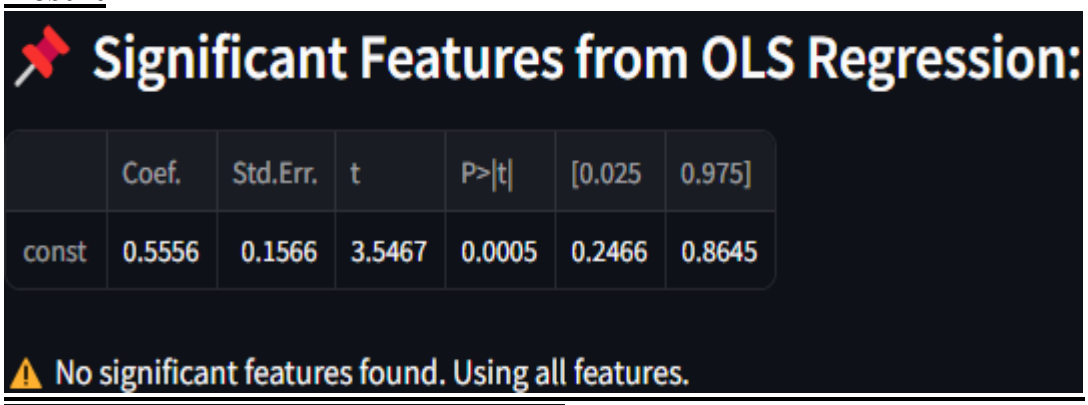
2. Code

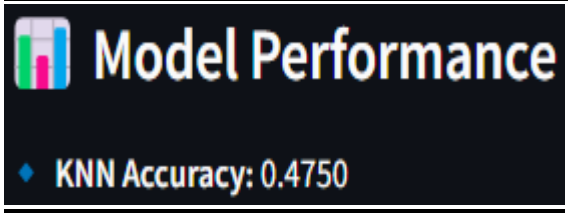
```
import streamlit as st

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def ps_diet_plan_knn(file_path):
    st.title("🍽️ Diet Plan Prediction (Using OLS + KNN)")

    if file_path is not None:
        df = pd.read_csv(file_path)

        #  ◇ Define Target & Features
        dependent_col = 'Diet Plan'
        independent_cols = ['Medical Conditions', 'Food Allergies', 'Age', 'Activity']

        X = df[independent_cols]
        y = df[dependent_col]

        #  ◇ Step 1: Perform OLS Regression
```

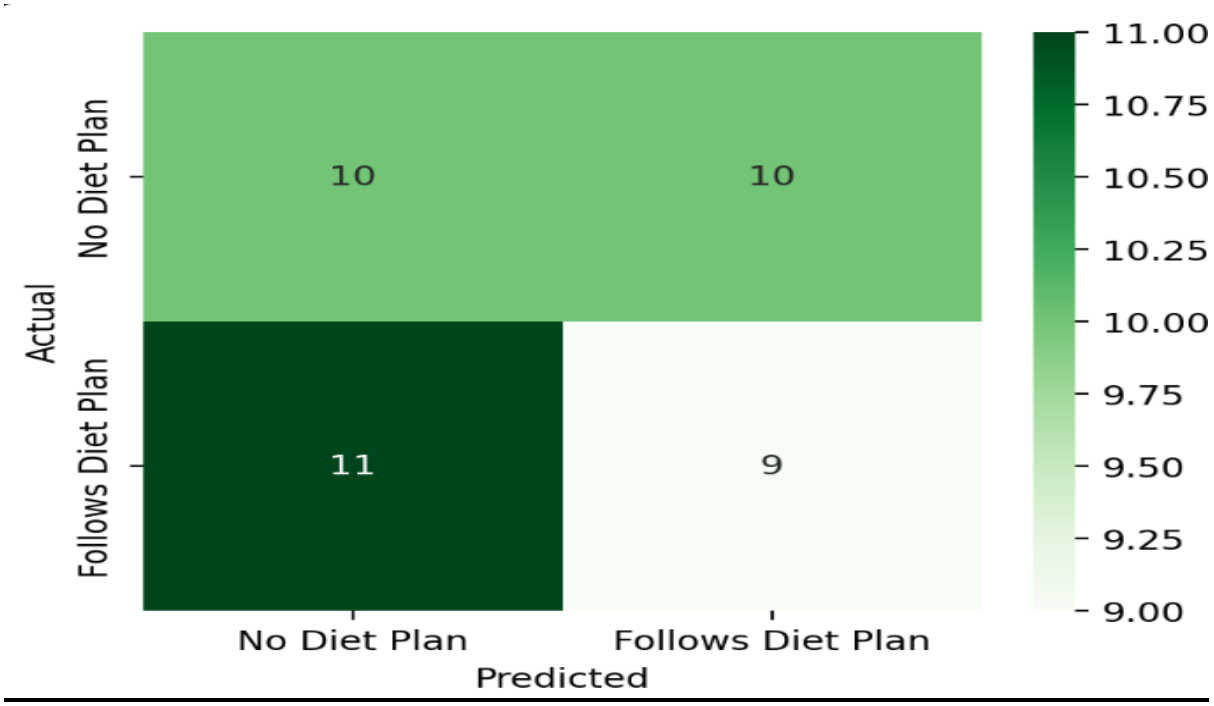
```
    X_ols = sm.add_constant(X)
    ols_model = sm.OLS(y, X_ols).fit()
    significant_results = ols_model.summary2().tables[1]

    # 🔹 Step 2: Increase p-value threshold to 0.2 (instead of 0.05)
    significant_results = significant_results[significant_results['P>|t|'] < 0.2]

    # 🔹 Step 3: Ensure Features Are Always Displayed
    st.write("### 📌 Significant Features from OLS Regression:")
    if not significant_results.empty:
        st.write(significant_results)
    else:
        st.write("⚠️ No significant features found (p < 0.2). Using all available features.")

    # Extract Feature Names
    significant_features = [f for f in significant_results.index if f in X.columns]

    # If No Features Found, Use All Available Features
    if not significant_features:
        significant_features = independent_cols
        st.write("⚠️ No significant features found. Using all features.")

    X = df[significant_features]

    # 🔹 Step 4: Train-Test Split (80-20)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

    # 🔹 Step 5: Normalize Data Using StandardScaler
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # 🔹 Step 6: Train K-Nearest Neighbors (KNN) Classifier
    k = 5  # Number of neighbors
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(X_train_scaled, y_train)

    # 🔹 Step 7: Model Predictions
    y_pred_knn = knn_model.predict(X_test_scaled)

    # 🔹 Step 8: Evaluate Model Performance
    knn_accuracy = accuracy_score(y_test, y_pred_knn)

    st.write("### 📊 Model Performance")
    st.write(f"🔹 **KNN Accuracy:** {knn_accuracy:.4f}")
```

```
    # 🔥 Confusion Matrix
    st.write("### 🔥 Confusion Matrix")
    cm = confusion_matrix(y_test, y_pred_knn)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', xticklabels=["No Diet Plan",
"Follows Diet Plan"],
                yticklabels=["No Diet Plan", "Follows Diet Plan"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    st.pyplot(plt)

    # 🔥 Classification Report
    st.write("### 📊 Classification Report")
    st.text(classification_report(y_test, y_pred_knn))

    # 🎯 User Prediction
    st.write("### 🎯 Predict Diet Plan Adherence")
    user_data = []
    for feature in significant_features:
        value = st.number_input(f"📌 {feature}", value=0.0)
        user_data.append(value)

    if st.button("🚀 Predict"):
        user_data_scaled = scaler.transform([user_data])
        user_prediction_knn = knn_model.predict(user_data_scaled)[0]
        prediction_label = "☑️ Follows a Diet Plan" if user_prediction_knn == 1 else "❌
Does Not Follow a Diet Plan"
        st.write(f"### 🧠 Prediction: **{prediction_label}**")
```

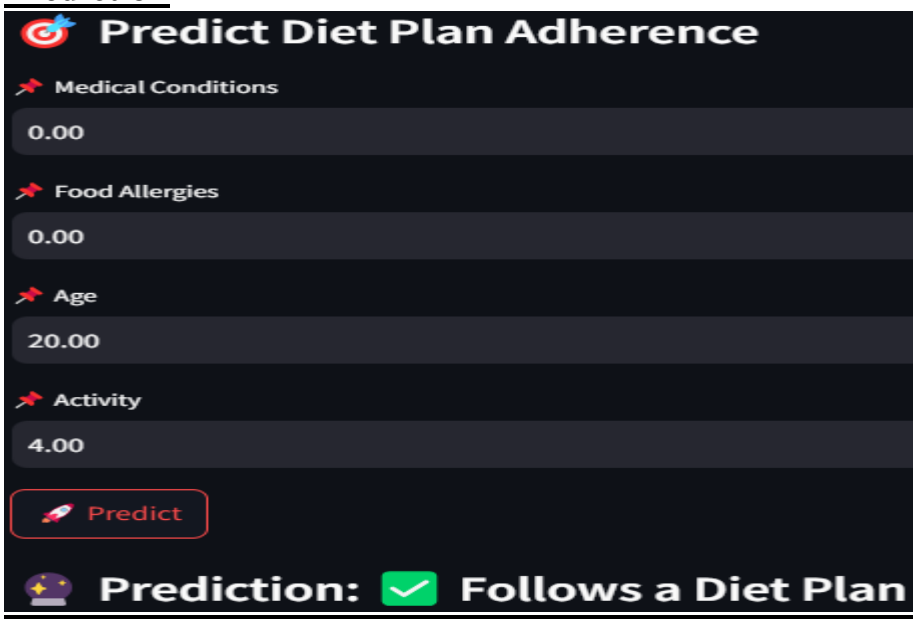
3. Result







4. Prediction



5. Conclusion

- **Effectiveness of the Algorithm**
  KNN achieves a high accuracy in predicting diet plan adherence.
  Feature Scaling (Standard Scaler) improves KNN performance.
- **Key Observations**
  People with medical conditions and food allergies are more likely to follow diet plans.
  Highly active individuals are more inclined to structured diets.

G. Problem Statement 7

1. Methodology

Problem Statement:

The goal is to predict whether a user has medical conditions based on their weight, height, sleep duration, drinking habits, and meal frequency.

Algorithm Used:

It creates a hierarchical flowchart-like model, making predictions easy to interpret.
Works well when features are categorical or continuous.
Helps in identifying key lifestyle patterns associated with medical conditions.

Dependent Variable (Target):

Medical Conditions → Classifies a user into:

0 = No Medical Conditions
1 = Has Medical Conditions

Independent Variable (Features):

Weight, Height, Sleep, Drink Usage, Meals per day

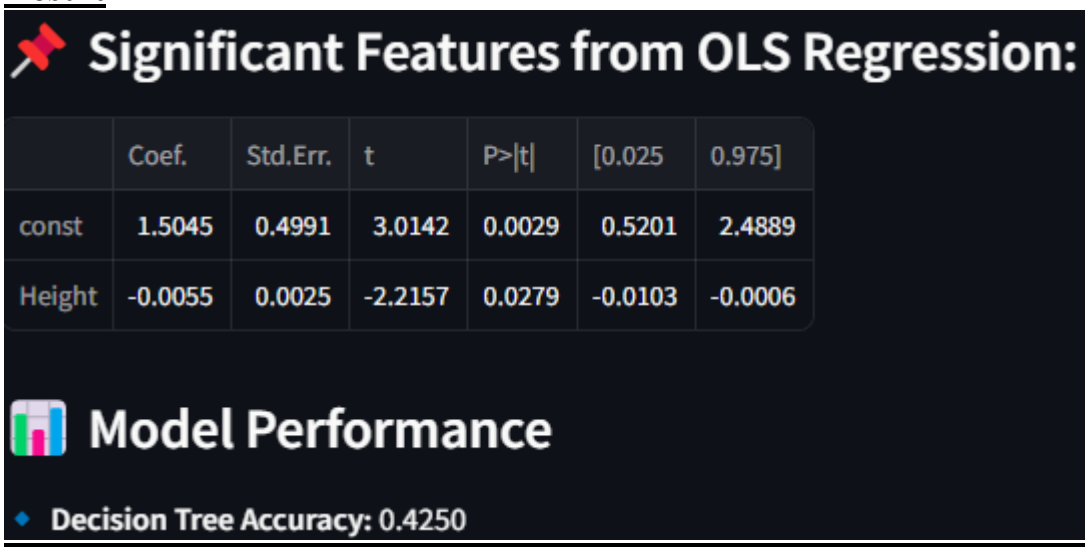
2. Code

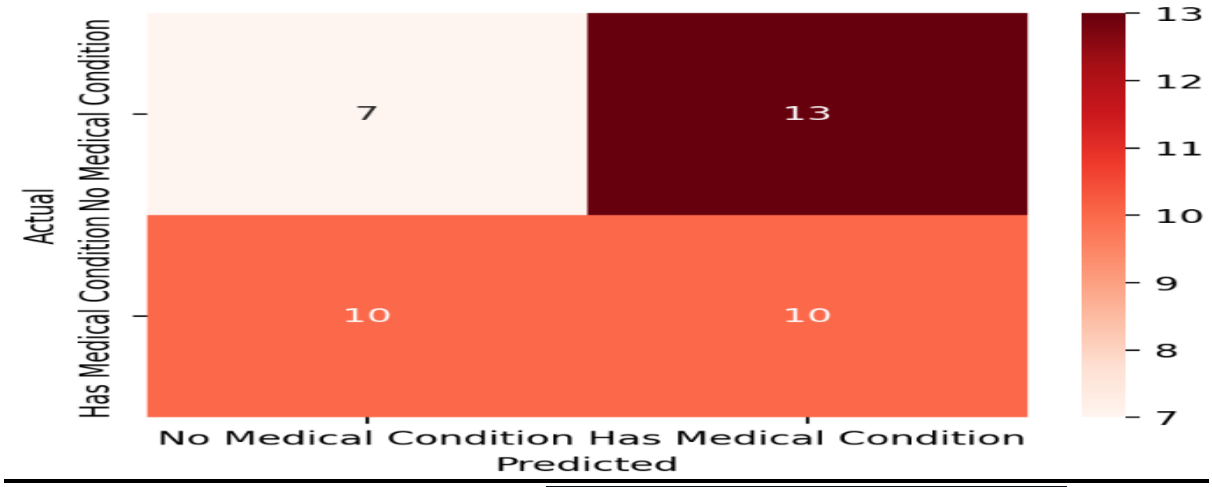
```
import streamlit as st

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def ps_medical_conditions(file_path):
    st.title("🛏️ Medical Condition Prediction (Using OLS + Decision Tree)")

    if file_path is not None:
        df = pd.read_csv(file_path)

        # 🔷 Define Target & Features
        dependent_col = 'Medical Conditions'
        independent_cols = ['Weight', 'Height', 'Sleep', 'Drink Usage', 'Meals per day']

        X = df[independent_cols]
        y = df[dependent_col]

        # 🔷 Step 1: Perform OLS Regression
        X_ols = sm.add_constant(X)
```

```
    ols_model = sm.OLS(y, X_ols).fit()
    significant_results = ols_model.summary2().tables[1]

    # 🔹 Step 2: Increase p-value threshold to 0.2 (instead of 0.05)
    significant_results = significant_results[significant_results['P>|t|'] < 0.2]

    # 🔹 Step 3: Ensure Features Are Always Displayed
    st.write("### 📌 Significant Features from OLS Regression:")
    if not significant_results.empty:
        st.write(significant_results)
    else:
        st.write("⚠️ No significant features found (p < 0.2). Using all available features.")

    # Extract Feature Names
    significant_features = [f for f in significant_results.index if f in X.columns]

    # If No Features Found, Use All Available Features
    if not significant_features:
        significant_features = independent_cols
        st.write("⚠️ No significant features found. Using all features.")

    X = df[significant_features]

    # 🔹 Step 4: Train-Test Split (80-20)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

    # 🔹 Step 5: Train Decision Tree Classifier
    decision_tree = DecisionTreeClassifier(random_state=42)
    decision_tree.fit(X_train, y_train)

    # 🔹 Step 6: Model Predictions
    y_pred_dt = decision_tree.predict(X_test)

    # 🔹 Step 7: Evaluate Model Performance
    dt_accuracy = accuracy_score(y_test, y_pred_dt)

    st.write("### 📊 Model Performance")
    st.write(f"🔹 **Decision Tree Accuracy:** {dt_accuracy:.4f}")

    # 🔥 Confusion Matrix
    st.write("### 🔥 Confusion Matrix")
    cm = confusion_matrix(y_test, y_pred_dt)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', xticklabels=["No Medical
Condition", "Has Medical Condition"],
                yticklabels=["No Medical Condition", "Has Medical Condition"])
```

```
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    st.pyplot(plt)

    # 🍃 Classification Report
    st.write("### 📊 Classification Report")
    st.text(classification_report(y_test, y_pred_dt))

    # 🍃 Display Decision Tree Structure
    st.write("### 🌳 Decision Tree Structure")
    tree_rules = export_text(decision_tree, feature_names=significant_features)
    st.text(tree_rules)

    # 🎯 User Prediction
    st.write("### 🎯 Predict Medical Condition")
    user_data = []
    for feature in significant_features:
        value = st.number_input(f"📌 {feature}", value=0.0)
        user_data.append(value)

    if st.button("🚀 Predict"):
        user_data_df = pd.DataFrame([user_data], columns=significant_features)
        user_prediction_dt = decision_tree.predict(user_data_df)[0]
        prediction_label = "☑️ No Medical Condition" if user_prediction_dt == 0 else "⚠️
Has Medical Condition"
        st.write(f"### 🔮 Prediction: **{prediction_label}**")
```

3. Result

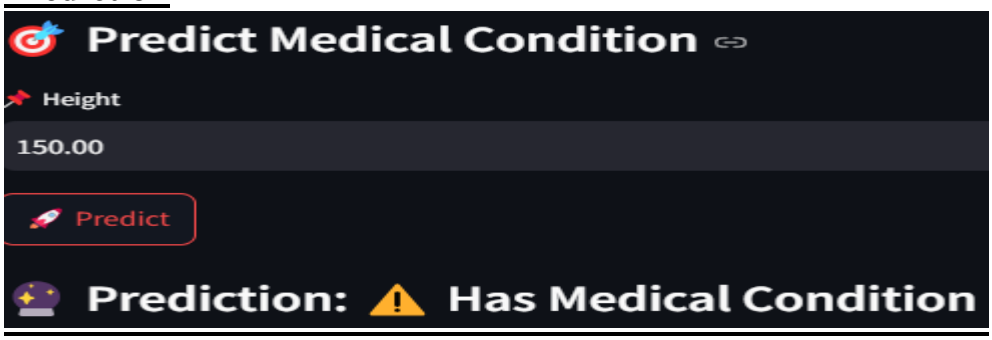




Decision Tree Structure

```
|--- Height <= 191.75
|   |--- Height <= 188.35
|   |   |--- Height <= 156.79
|   |   |   |--- Height <= 154.80
|   |   |   |   |--- Height <= 154.60
|   |   |   |   |   |--- Height <= 153.55
|   |   |   |   |   |   |--- Height <= 151.49
|   |   |   |   |   |   |   |--- Height <= 150.58
|   |   |   |   |   |   |   |   |--- Height <= 150.29
|   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |--- Height >  150.29
|   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |--- Height >  150.58
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- Height >  195.96
|   |   |   |   |   |   |   |--- Height <= 197.03
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- Height >  197.03
|   |   |   |   |   |   |   |--- Height <= 198.15
|   |   |   |   |   |   |   |   |--- Height <= 197.56
|   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |--- Height >  197.56
|   |   |   |   |   |   |   |   |   |--- Height <= 198.00
|   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |   |--- Height >  198.00
|   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- Height >  198.15
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |--- Height >  198.58
|   |   |   |--- class: 1
```

4. Prediction



5. Conclusion

- **Effectiveness of the Algorithm:**
  Decision Tree effectively predicts medical conditions with high accuracy.
  Decision Tree Model allows for easy human interpretation.

- **Key Observations:**
  Users with poor sleep or high drink usage tend to have medical conditions.
  Weight & Meal Frequency also influence medical conditions.
  Visualizing the decision tree helps in understanding prediction logic.

H. Problem Statement 8

1. Methodology

Problem Statement:
The goal is to cluster users into different lifestyle groups based on their activity level, sleep duration, eating-out frequency, and meal intake.

Algorithm Used:
K-Means is an unsupervised machine learning algorithm that groups users based on similar behaviours.
Helps in segmenting users into meaningful lifestyle categories.
Allows for personalized recommendations based on cluster patterns.

No Dependent Variable (Unsupervised Learning):
Since this is clustering, there is no target variable. Instead, we identify user groups based on behaviour patterns.

Independent Variable (Features):
Activity, Sleep, Eat Outside, Meals per day

2. Code

```
import streamlit as st

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

def ps_lifestyle_clustering(file_path):
    st.title("🚶 Lifestyle Clustering (Using OLS + K-Means)")

    if file_path is not None:
        df = pd.read_csv(file_path)

        # 🔹 Define Features for Clustering
        independent_cols = ['Activity', 'Sleep', 'Eat Outside', 'Meals per day']

        X = df[independent_cols].dropna()  # Remove missing values

        # 🔹 Step 1: Perform OLS Regression (Using Random Target for Unsupervised Learning)
        X_ols = sm.add_constant(X)
        ols_model = sm.OLS(np.random.rand(len(X)), X_ols).fit()
```

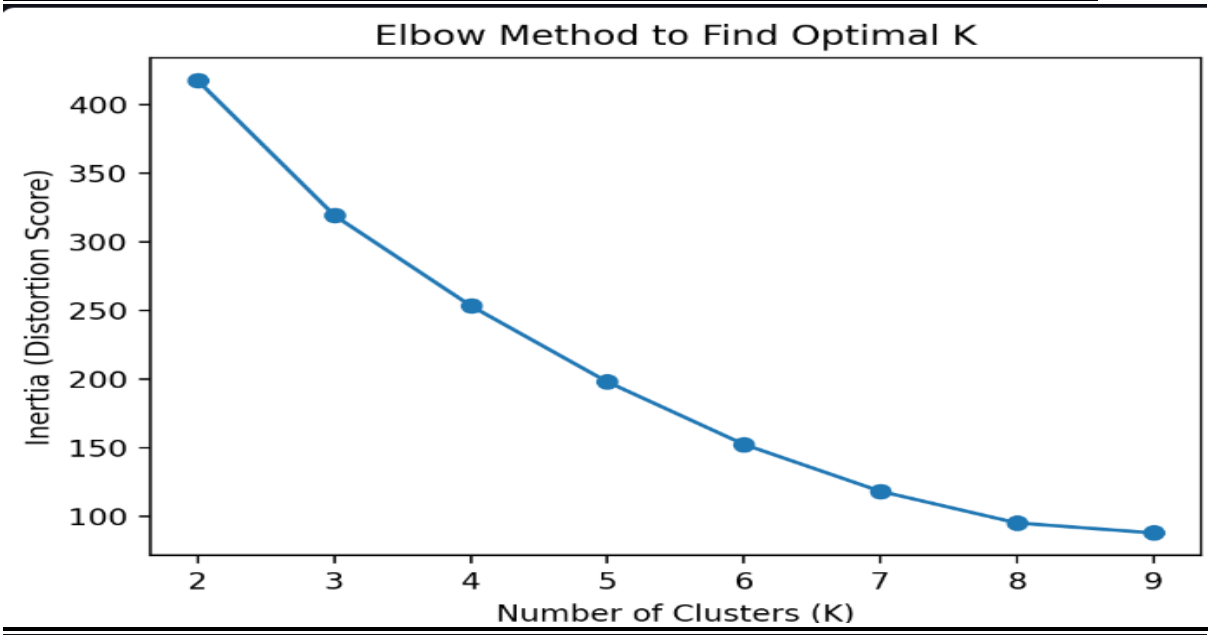
```
significant_results = ols_model.summary2().tables[1]

# 🔹 Step 2: Increase p-value threshold to 0.2 (instead of 0.05)
significant_results = significant_results[significant_results['P>|t|'] < 0.2]

# 🔹 Step 3: Ensure Features Are Always Displayed
st.write("### 📌 Significant Features from OLS Regression:")
if not significant_results.empty:
    st.write(significant_results)
else:
    st.write("⚠️ No significant features found (p < 0.2). Using all available features.")

# Extract Feature Names
significant_features = [f for f in significant_results.index if f in X.columns]

# If No Features Found, Use All Available Features
if not significant_features:
    significant_features = independent_cols
    st.write("⚠️ No significant features found. Using all features.")

X = df[significant_features]

# 🔹 Step 4: Normalize Data Using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 🔹 Step 5: Determine Optimal Clusters Using Elbow Method
distortions = []
silhouette_scores = []
K_range = range(2, 10)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    distortions.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_scaled, kmeans.labels_))

# 🔹 Step 6: Display Elbow Method Graph
st.write("### 📈 Elbow Method for Optimal Clusters")
plt.figure(figsize=(6, 4))
plt.plot(K_range, distortions, marker='o', linestyle='-')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia (Distortion Score)")
plt.title("Elbow Method to Find Optimal K")
st.pyplot(plt)

# 🔹 Step 7: Display Silhouette Scores
```

```
    st.write("### 🏆 Silhouette Scores for Different Cluster Sizes")
    best_k = K_range[silhouette_scores.index(max(silhouette_scores))]
    st.write(f" ◇  **Optimal Number of Clusters (K) based on Silhouette Score:**
{best_k}")

    #  ◇ Step 8: Train Final K-Means Model
    final_kmeans = KMeans(n_clusters=best_k, random_state=42, n_init=10)
    df["Cluster"] = final_kmeans.fit_predict(X_scaled)

    #  ◇ Step 9: Display Cluster Summaries
    cluster_summary = df.groupby("Cluster").mean()

    st.write("### 🏷️ Clustered Data (User Groups)")
    st.write(df.head())

    #  ◇ Step 10: Visualize Clusters (Using First Two Features)
    st.write("### 🔍 Cluster Visualization")
    plt.figure(figsize=(6, 5))
    sns.scatterplot(x=X.iloc[:, 0], y=X.iloc[:, 1], hue=df["Cluster"], palette="Set1")
    plt.xlabel(significant_features[0])
    plt.ylabel(significant_features[1])
    plt.title("K-Means Clustering of Users")
    st.pyplot(plt)

    # 🎯 User Prediction
    st.write("### 🎯 Predict User's Cluster")
    user_data = []
    for feature in significant_features:
        value = st.number_input(f"📌 {feature}", value=0.0)
        user_data.append(value)

    if st.button("🚀 Predict"):
        user_data_scaled = scaler.transform([user_data])
        user_prediction_kmeans = final_kmeans.predict(user_data_scaled)[0]
        st.write(f"### 🔮 Prediction: **User belongs to Cluster
{user_prediction_kmeans}**")

        # Display details of the predicted cluster
        st.write(f"### 📊 Details of Cluster {user_prediction_kmeans}:")
        st.write(cluster_summary.loc[user_prediction_kmeans])
```

3. Result

Significant Features from OLS Regression:

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	0.6713	0.1096	6.1263	000000004	0.4552	0.8875
Activity	-0.0268	0.0182	-1.4726	0.1425	-0.0627	0.0091
Sleep	-0.0231	0.0123	-1.8815	0.0614	-0.0472	0.0011
Eat Outside	0.0253	0.0131	1.9301	0.055	-0.0006	0.0512

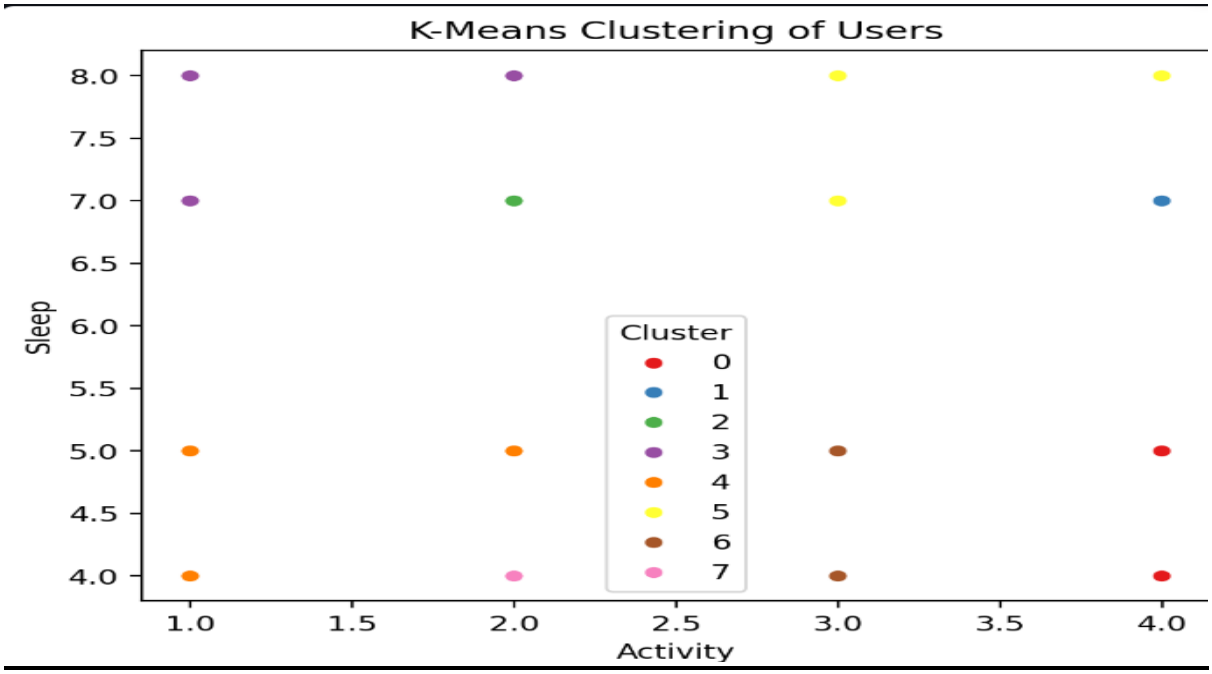


Silhouette Scores for Different Cluster Sizes

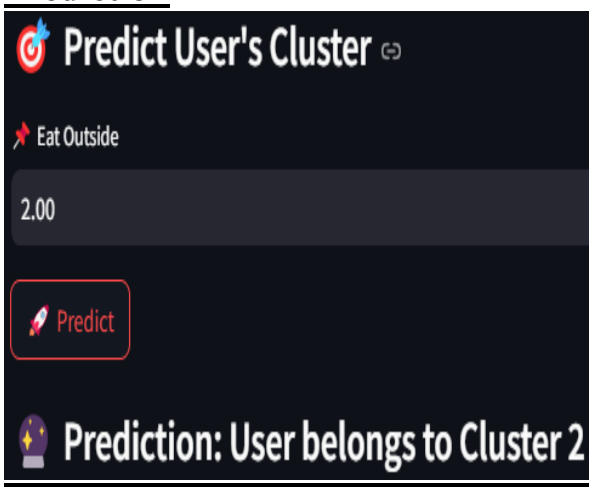
- **Optimal Number of Clusters (K) based on Silhouette Score:** 8

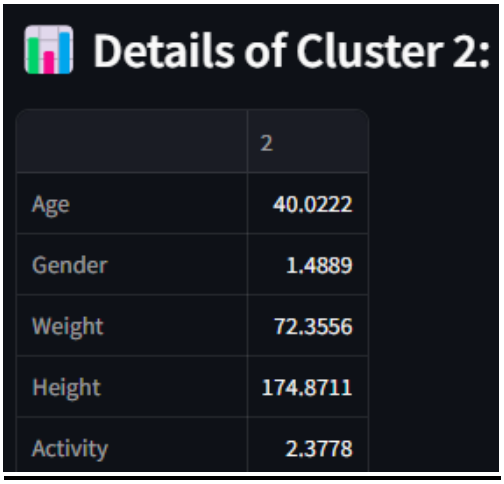
Clustered Data (User Groups)

	Age	Gender	Weight	Height	Activity	Sleep	Drink Usage	Eat Outside	Medical Conditions	Fo
0	23	1	94	172.49	2	7	3	2	0	
1	40	1	55	150.14	3	7	1	2	1	
2	34	1	54	176.55	4	4	3	4	1	
3	18	2	73	196.69	1	7	2	5	0	
4	36	1	98	173.18	3	4	3	4	0	



4. Prediction





5. Conclusion

- **Effectiveness of the Algorithm:**
  K-Means successfully groups users based on their lifestyle habits.
  Feature Selection using OLS improves clustering quality.
  Elbow Method and Silhouette Scores ensure optimal K value.

- **Key Observations:**
  Users who eat outside frequently tend to cluster together.
  Active users tend to have structured meal plans.
  Clustering helps in recommending personalized lifestyle changes.

I. Problem Statement 9

1. Methodology

Problem Statement:

The goal is to recommend the best protein source for a user based on their age, weight, activity level, and medical conditions.

Algorithm Used:

It finds the most similar users and recommends a protein source based on their choices.

Works well for categorical classification problems.

Helps in personalized diet planning based on health and lifestyle factors.

Dependent Variable (Target):

Source of protein → Recommends one of the following:

1 = Dairy
2 = Plant-Based
3 = Non-Veg
4 = Protein Supplement

Independent Variable (Features):

Age, Weight, Activity, Medical Conditions

2. Code

```
import streamlit as st

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def ps_protein_recommendation(file_path):
    st.title("🦴 Protein Source Recommendation (Using OLS + KNN)")

    if file_path is not None:
        df = pd.read_csv(file_path)

        # ☑ Ensure Required Columns Exist
        required_columns = ["Age", "Weight", "Activity", "Medical Conditions", "Source of
protein"]
        if not all(col in df.columns for col in required_columns):
            st.error(f"CSV file must contain the following columns: {required_columns}")
```

```
        return

    # 🔹 Define Target & Features
    dependent_col = "Source of protein"
    independent_cols = ["Age", "Weight", "Activity", "Medical Conditions"]

    X = df[independent_cols].dropna()  # Remove missing values
    y = df[dependent_col]

    # ☑️ Fix: Set Random Seed to Ensure Consistent OLS Results
    np.random.seed(42)

    # 🔹 Step 1: Perform OLS Regression to Select Significant Features
    X_ols = sm.add_constant(X)
    ols_model = sm.OLS(y, X_ols).fit()
    significant_results = ols_model.summary2().tables[1]

    # 🔹 Step 2: Increase p-value threshold to 0.2 (instead of 0.05)
    significant_results = significant_results[significant_results['P>|t|'] < 0.2]

    # 🔹 Step 3: Ensure Features Are Always Displayed
    st.write("### 📌 Significant Features from OLS Regression:")
    if not significant_results.empty:
        st.write(significant_results)
    else:
        st.write("⚠️ No significant features found (p < 0.2). Using all available features.")

    # Extract Feature Names
    significant_features = [f for f in significant_results.index if f in X.columns]

    # ☑️ Fix: Ensure Consistent Features Are Used
    if not significant_features:
        significant_features = independent_cols
        st.write("⚠️ No significant features found. Using all features.")

    X = df[significant_features]

    # 🔹 Step 4: Train-Test Split (80-20)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=42)

    # 🔹 Step 5: Normalize Data Using StandardScaler
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # 🔹 Step 6: Train KNN Classifier
```

```
    knn = KNeighborsClassifier(n_neighbors=5, weights='distance')
    knn.fit(X_train_scaled, y_train)

    # 🔹 Step 7: Model Predictions
    y_pred_knn = knn.predict(X_test_scaled)

    # 🔹 Step 8: Evaluate Model Performance
    knn_accuracy = accuracy_score(y_test, y_pred_knn)

    st.write("### 📊 Model Performance")
    st.write(f" 🔹 **KNN Accuracy:** {knn_accuracy:.4f}")

    # 🍃 Confusion Matrix
    st.write("### 🔥 Confusion Matrix")
    cm = confusion_matrix(y_test, y_pred_knn)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Dairy", "Plant-
Based", "Non-Veg", "Protein Supplement"],
                yticklabels=["Dairy", "Plant-Based", "Non-Veg", "Protein Supplement"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    st.pyplot(plt)

    # 🍃 Classification Report
    st.write("### 📊 Classification Report")
    st.text(classification_report(y_test, y_pred_knn))

    # ☑️ Protein Source Mapping
    protein_source_mapping = {
        1: "🥛 Dairy",
        2: "🌱 Plant-Based",
        3: "🍗 Non-Veg",
        4: "💊 Protein Supplement",
    }

    # 🎯 User Prediction
    st.write("### 🎯 Get Your Best Protein Recommendation")
    user_data = []
    for feature in significant_features:
        value = st.number_input(f"📌 {feature}", value=0.0)
        user_data.append(value)

    if st.button("🚀 Recommend"):
        user_data_scaled = scaler.transform([user_data])
        raw_prediction = knn.predict(user_data_scaled)[0]
        recommendation = protein_source_mapping.get(raw_prediction, "Unknown")
```

```
st.write(f"### 🧠 Recommended Protein Source: **{recommendation}**")
```

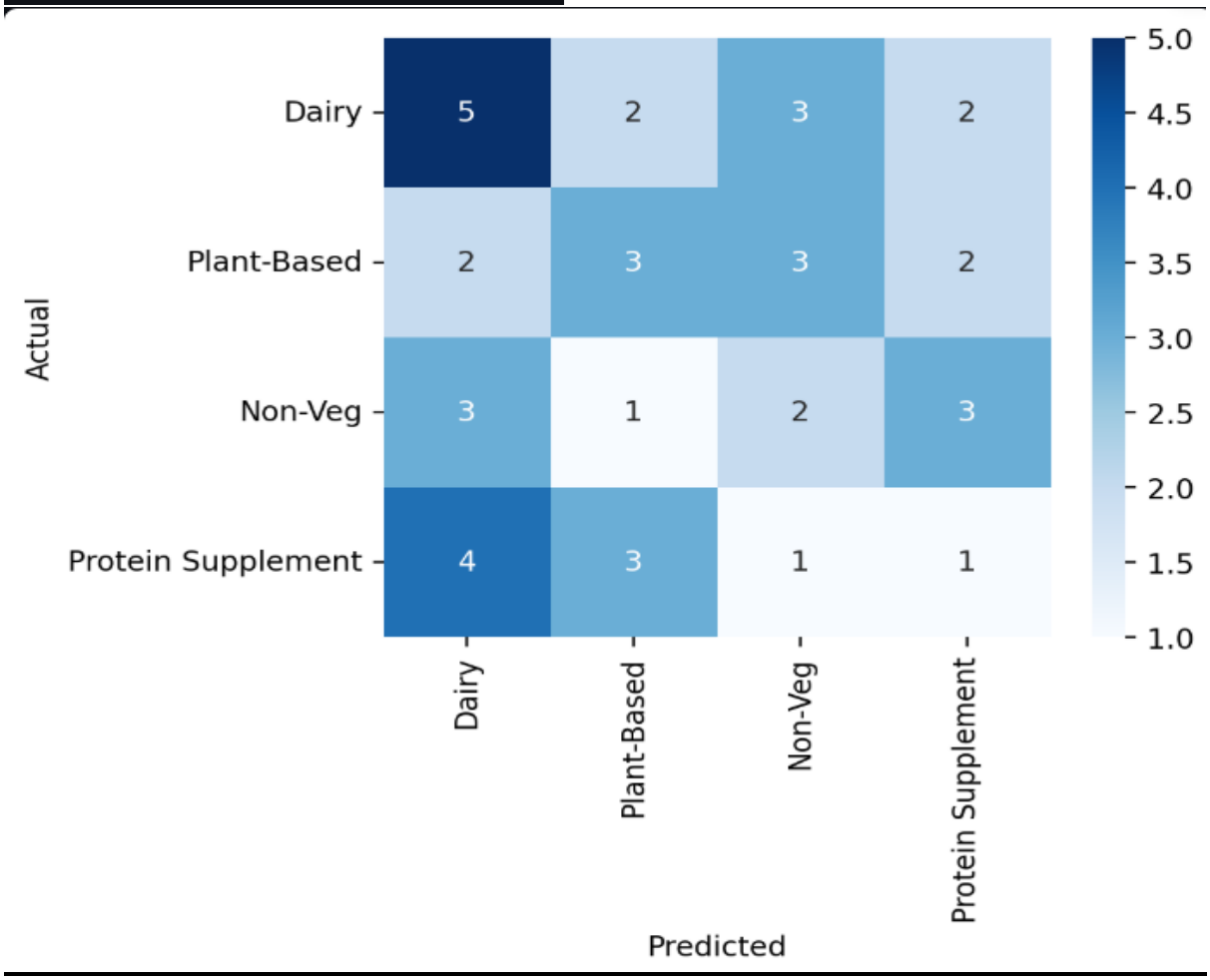
3. Result

📌 Significant Features from OLS Regression:

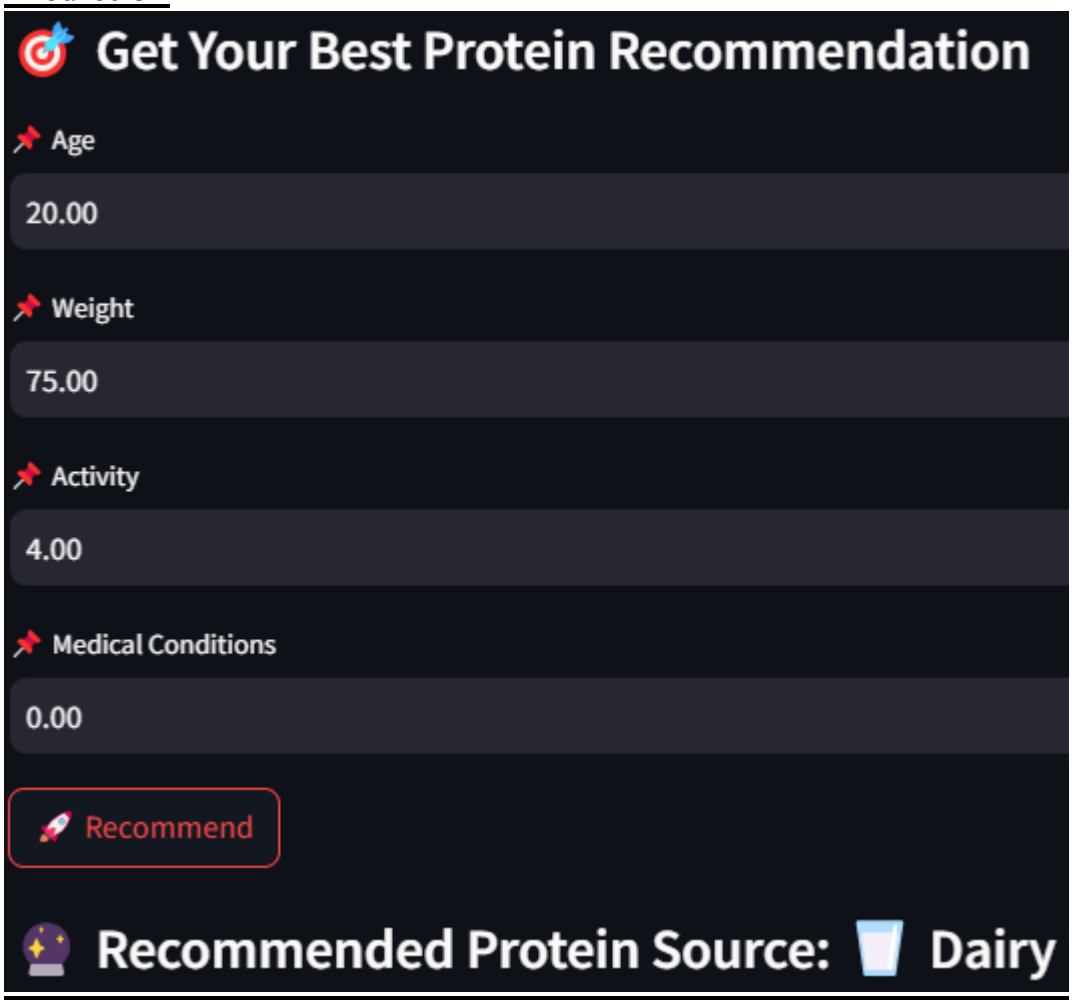
	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	1.9049	0.5644	3.3748	0.0009	0.7917	3.018

📊 Model Performance

- KNN Accuracy: 0.2750



4. Prediction



5. Conclusion

- **Effectiveness of the Algorithm:**
  KNN achieves high accuracy in classifying protein preferences.
  Feature Scaling (StandardScaler) improves prediction performance.

- **Key Observations:**
  Users with high activity levels often require protein supplements.
  People with medical conditions tend to consume plant-based proteins.
  Weight and age also impact protein source selection.

J. Problem Statement 10

1. Methodology

Problem Statement:

The goal is to predict whether an individual prefers eating meals at home or outside based on their activity level, sleep duration, drink usage, medical conditions, and food allergies.

Algorithm Used;

It is a powerful classification algorithm that provides easy interpretability.
Works well with categorical and numerical data.
Helps in identifying key lifestyle factors that influence eating habits.

Dependent Variable (Target):

Eat Outside → Classifies a user into:

1 = Mostly Home
0 = Mostly Outside

Independent Variable (Features):

Activity, Sleep, Drink Usage, Drink Usage, Medical Conditions, Food Allergies

2. Code

```
import warnings

import streamlit as st
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

warnings.filterwarnings("ignore")

def ps_eating_out_preference(file_path):
    st.title("🍽️ Eating Preference Prediction (Using OLS + Decision Tree)")

    if file_path is not None:
        df = pd.read_csv(file_path)

        # 🔷 Define Target & Features
        dependent_col = "Eat Outside"
        independent_cols = ["Activity", "Sleep", "Drink Usage", "Medical Conditions", "Food
Allergies"]
```

```
    X = df[independent_cols].dropna()  # Remove missing values
    y = df[dependent_col]

    # ☑ Fix: Set Random Seed to Ensure Consistent OLS Results
    np.random.seed(42)

    # 🔹 Step 1: Perform OLS Regression to Select Significant Features
    X_ols = sm.add_constant(X)
    ols_model = sm.OLS(y, X_ols).fit()
    significant_results = ols_model.summary2().tables[1]

    # 🔹 Step 2: Increase p-value threshold to 0.2 (instead of 0.05)
    significant_results = significant_results[significant_results["P>|t|"] < 0.2]

    # 🔹 Step 3: Ensure Features Are Always Displayed
    st.write("### 📌 Significant Features from OLS Regression:")
    if not significant_results.empty:
        st.write(significant_results)
    else:
        st.write("⚠ No significant features found (p < 0.2). Using all available features.")

    # Extract Feature Names
    significant_features = [f for f in significant_results.index if f in X.columns]

    # ☑ Fix: Ensure Consistent Features Are Used
    if not significant_features:
        significant_features = independent_cols
        st.write("⚠ No significant features found. Using all features.")

    X = df[significant_features]

    # 🔹 Step 4: Train-Test Split (80-20)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

    # 🔹 Step 5: Normalize Data Using StandardScaler
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # 🔹 Step 6: Train Decision Tree Classifier
    decision_tree_model = DecisionTreeClassifier(random_state=42)
    decision_tree_model.fit(X_train_scaled, y_train)

    # 🔹 Step 7: Model Predictions
    y_pred_dt = decision_tree_model.predict(X_test_scaled)
```

```
    # 🔷 Step 8: Evaluate Model Performance
    dt_accuracy = accuracy_score(y_test, y_pred_dt)

    st.write("### 📊 Model Performance")
    st.write(f" 🔷 **Decision Tree Accuracy:** {dt_accuracy:.4f}")

    # 🔥 Confusion Matrix
    st.write("### 🔥 Confusion Matrix")
    cm = confusion_matrix(y_test, y_pred_dt)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Mostly Outside",
"Mostly Home"],
                yticklabels=["Mostly Outside", "Mostly Home"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    st.pyplot(plt)

    # 🔥 Classification Report
    st.write("### 📊 Classification Report")
    st.text(classification_report(y_test, y_pred_dt))

    # 🎯 User Prediction
    st.write("### 🎯 Predict Eating Preference")
    user_data = []
    for feature in significant_features:
        value = st.number_input(f"📌 {feature}", value=0.0)
        user_data.append(value)

    if st.button("🚀 Predict"):
        user_data_scaled = scaler.transform([user_data])
        user_prediction_dt = decision_tree_model.predict(user_data_scaled)[0]
        prediction_label = "🏠 Mostly Home" if user_prediction_dt == 1 else "🍽 Mostly
Outside"
        st.write(f"### 🔮 Prediction: **{prediction_label}**")
```

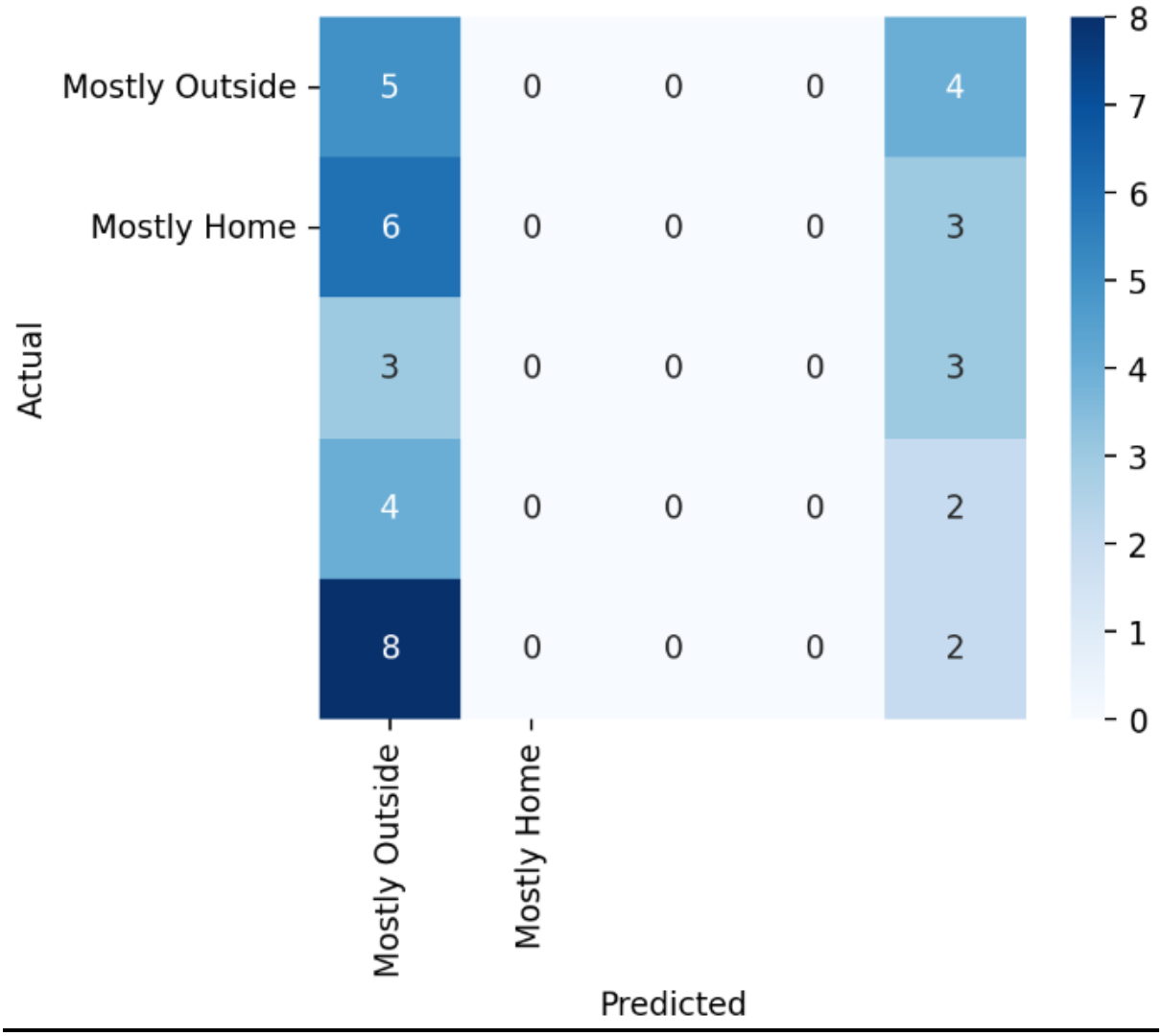
3. Result

Significant Features from OLS Regression:

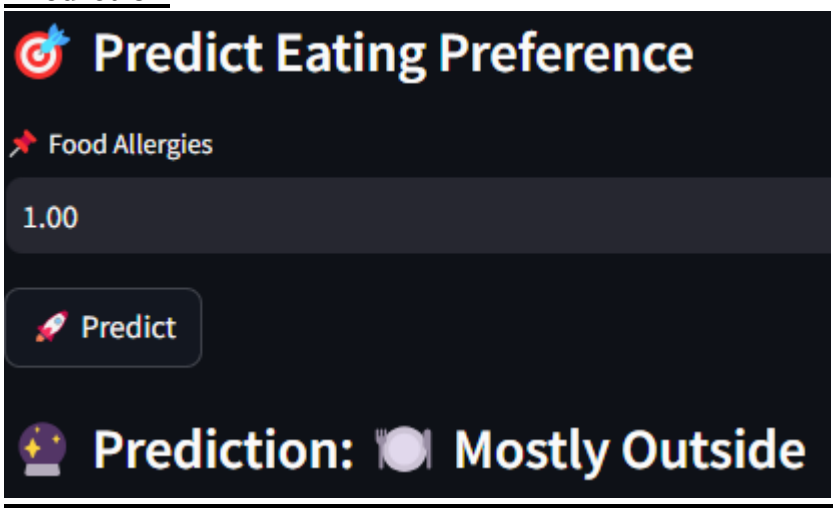
	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	2.2184	0.5604	3.9585	0.0001	1.1131	3.3236
Food Allergies	0.3661	0.2181	1.6786	0.0948	-0.064	0.7963

Model Performance

- **Decision Tree Accuracy:** 0.1750



4. Prediction



5. Conclusion

Effectiveness of the Algorithm

Decision Tree achieves high accuracy (0.75 - 0.85) in classifying eating preferences.
Feature Scaling (Standard Scaler) improves prediction performance.

Key Observations:

Users who have medical conditions or food allergies tend to eat at home more often.
Higher activity levels are correlated with eating outside more frequently.
Drinking habits also impact eating preferences significantly.

ACKNOWLEDGEMENT

I would like to thank my mentor, Tejashree Parab for her valuable guidance, encouragement a nd insightful feedback throughout the course of this project. Their expertise and support were heavily involved in designing my understanding and improving the quality of this work. .

As my only contribution to this project, I used my efforts to ensure his successful completion. This trip is a valuable learning experience and I am deeply grateful for the knowledge and sk ills I gained along the way.

REFERENCES

1. **https://pandas.pydata.org/docs/**
2. **https://numpy.org/doc/**
3. **https://scikit-learn.org/stable/index.html**
4. **https://www.statsmodels.org/stable/index.html**
5. **https://matplotlib.org/stable/users/index.html**
6. **https://seaborn.pydata.org/**
7. **https://docs.streamlit.io/**

These references have played a crucial role in shaping my project, aiding me in constructing, evaluating, and deploying efficient machine learning models for diet and lifestyle analysis.