



Obesity Data Analysis Project - Data Cleaning

Peter Gray

September 4, 2025

Produced with Quarto and Python

Table of contents

List of Python and R Packages	3
Data Cleaning	4
Import Data	4
Clean Data	4
Select Columns of Interest	4
Rename Data Columns	5
Rename Column Values	6
Standardise Measurements	12
Create Age Groups	13
Exploratory Analysis	14
Visualisation of Categorical Variables	16
Data Analysis	29
Calorie Intake per Ethnicity	29
Box Plot of Calorie Intake per Age Group	30
Weighted Analysis	32

List of Python and R Packages

Python
NumPy
Pandas
Matplotlib
seaborn
random
IPython.display
tabulate
os
plotnine

Data Cleaning

Import Data

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

#Don't do Scientific notation
pd.options.display.float_format = '{:.2f}'.format

demographics =
    pd.read_sas(r"/home/pgr16/Documents/Data_Analysis/Obesity_Analysis_and_Prediction/Data/DEMO_LI.SAS7BAK")

day_1_intake =
    pd.read_sas(r"/home/pgr16/Documents/Data_Analysis/Obesity_Analysis_and_Prediction/Data/DR1TOTD.SAS7BAK")

food_codes =
    pd.read_sas(r"/home/pgr16/Documents/Data_Analysis/Obesity_Analysis_and_Prediction/Data/DRXFCD.SAS7BAK")

measures =
    pd.read_sas(r"/home/pgr16/Documents/Data_Analysis/Obesity_Analysis_and_Prediction/Data/BMX_LI.SAS7BAK")
```

Clean Data

Select Columns of Interest

```
dm_columns = ["SEQN", "SDDSRVYR", "RIDSTATR", "RIDAGEYR", "RIAGENDR",
    "RIDRETH1", "DMDBORN4", "DMDEDUC2", "DMDMARTZ", "WTINT2YR"]

demographics = demographics[dm_columns]

intake_columns = ["SEQN", "WTDRD1", "WTDR2D", "DR1DRSTZ", "DRABF", "DR1DAY",
    "DR1MRESP", "DBD100", "DRQSPREP", "DR1TNUMF", "DRQSDIET", "DRQSDT1",
    "DRQSDT2", "DRQSDT3", "DRQSDT4", "DRQSDT5", "DRQSDT6", "DRQSDT11",
    "DR1TKCAL", "DR1TPROT", "DR1TCARB", "DR1TSUGR", "DR1TFIBE", "DR1TTFAT",
    "DR1TSFAT", "DR1TMFAT", "DR1TPFAT", "DR1TCHOL", "DR1TCAFF", "DR1TALCO",
    "DR1_320Z"]

day_1_intake = day_1_intake[intake_columns]

food_code_columns = ["DRXFCD", "DRXFCSD"]

food_codes = food_codes[food_code_columns]
```

Rename Data Columns

```
demographics.rename(columns={"SEQN": "SUBJID", "SDDSRVYR" : "CYCLE", "RIDAGEYR"
                           : "AGE_YEARS_SCREENING", "RIDSTATR": "INTERVIEW_STATUS", "RIAGENDR" :
                           "GENDER", "RIDRETH1": "RACE", "DMDBORN4" : "BIRTH_COUNTRY", "DMDEDUC2" :
                           "EDUCATION", "DMDMARTZ": "MARITAL_STATUS", "WTINT2YR" : "2_YEAR_WEIGHT"},  
                           inplace=True)

day_1_intake.rename(columns={"SEQN": "SUBJID", "WTDRD1" : "SAMPLE_WEIGHT_DAY1",
                           "WTDR2D" : "SAMPLE_WEIGHT_DAY2", "DR1DRSTZ" : "RECALL_RELIABILITY", "DRABF"
                           : "INFANT_BREAST_FED", "DR1DAY" : "INTAKE_DAY_WEEK", "DR1MRESP":
                           "MAIN RESPONDENT", "DBD100" : "SALT_AT_TABLE", "DRQSPREP" : "SALT_IN_PREP",
                           "DRQSDIET" : "SPECIAL_DIET" , "DRQSDT1" : "WEIGHT_LOSS_DIET", "DRQSDT2" :
                           "LOW_FAT_DIET", "DRQSDT3" : "LOW_SALT_DIET", "DRQSDT4" : "LOW_SUGER_DIET",
                           "DRQSDT5" : "LOW_FIBER_DIET", "DRQSDT6" : "HIGH_FIBER_DIET", "DRQSDT11" :
                           "CELIAC_DIET", "DR1TNUMF" : "NUMBER_FOOD_DRINKS", "DR1TKCAL" : "CALORIES",
                           "DR1TPROT" : "NUT_PROTEIN_GRAMS", "DR1TCARB" : "NUT_CARBS_GRAMS",
                           "DR1TSUGR" : "NUT_SUGAR_GRAMS", "DR1TFIBE" : "NUT_FIBRE_GRAMS", "DR1TTFAT"
                           : "NUT_FAT_GRAMS", "DR1TSFAT" : "NUT_SAT_FAT_GRAMS", "DR1TMFAT":
                           "NUT_MONO_SAT_FAT_GRAMS", "DR1TPFAT" : "NUT_POLY_SAT_FAT_GRAMS", "DR1TCHOL"
                           : "NUT_CHOL_MILIGRAMS", "DR1TCAFF" : "NUT_CAFFEINE_MILIGRAMS", "DR1TALCO":
                           "NUT_ALCOHOL_GRAMS", "DR1_320Z" : "TOTAL_WATER_DRANK"}, inplace=True)

food_codes.rename(columns = {"DRXFD_CD": "FOOD_CODE", "DRXFCSD" : "FOOD_NAME"},  
                           inplace=True)

#Remove Scientific Notation
food_codes['FOOD_CODE'] = food_codes['FOOD_CODE'].astype(str)
```

Rename Column Values

```
# Demogrpahics

demographics["INTERVIEW_STATUS"] = np.where(
    demographics["INTERVIEW_STATUS"] == 1,
    "Interviewed Only",
    demographics["INTERVIEW_STATUS"],
)
demographics["INTERVIEW_STATUS"] = np.where(
    demographics["INTERVIEW_STATUS"] == "2.0",
    "Interviewed and Medically Examined",
    demographics["INTERVIEW_STATUS"],
)

demographics["GENDER"] = np.where(
    demographics["GENDER"] == 1, "Male", demographics["GENDER"]
)
demographics["GENDER"] = np.where(
    demographics["GENDER"] == "2.0", "Female", demographics["GENDER"]
)

demographics["RACE"] = np.where(
    demographics["RACE"] == 1, "Mexican American", demographics["RACE"]
)
demographics["RACE"] = np.where(
    demographics["RACE"] == "2.0", "Other Hispanic", demographics["RACE"]
)
demographics["RACE"] = np.where(
    demographics["RACE"] == "3.0", "Non-Hispanic White", demographics["RACE"]
)
demographics["RACE"] = np.where(
    demographics["RACE"] == "4.0", "Non-Hispanic Black", demographics["RACE"]
)
demographics["RACE"] = np.where(
    demographics["RACE"] == "5.0",
    "Other Race - Including Multi-Racial",
    demographics["RACE"],
)

demographics["BIRTH_COUNTRY"] = np.where(
    demographics["BIRTH_COUNTRY"] == 1, "US", demographics["BIRTH_COUNTRY"]
)
demographics["BIRTH_COUNTRY"] = np.where(
    demographics["BIRTH_COUNTRY"] == "2.0", "Others",
    demographics["BIRTH_COUNTRY"]
)
```

```

demographics["BIRTH_COUNTRY"] = np.where(
    demographics["BIRTH_COUNTRY"] == "77", "Refused",
    demographics["BIRTH_COUNTRY"]
)
demographics["BIRTH_COUNTRY"] = np.where(
    demographics["BIRTH_COUNTRY"] == "99", "Don't know",
    demographics["BIRTH_COUNTRY"]
)

demographics["EDUCATION"] = np.where(
    demographics["EDUCATION"] == 1, "Less than 9th grade",
    demographics["EDUCATION"]
)
demographics["EDUCATION"] = np.where(
    demographics["EDUCATION"] == "2.0",
    "9-11th grade (Includes 12th grade with no diploma)",
    demographics["EDUCATION"],
)
demographics["EDUCATION"] = np.where(
    demographics["EDUCATION"] == "3.0",
    "High school graduate/GED or equivalent",
    demographics["EDUCATION"],
)
demographics["EDUCATION"] = np.where(
    demographics["EDUCATION"] == "4.0",
    "Some college or AA degree",
    demographics["EDUCATION"],
)
demographics["EDUCATION"] = np.where(
    demographics["EDUCATION"] == "5.0",
    "College graduate or above",
    demographics["EDUCATION"],
)
demographics["EDUCATION"] = np.where(
    demographics["EDUCATION"] == "7.0", "Refused", demographics["EDUCATION"]
)
demographics["EDUCATION"] = np.where(
    demographics["EDUCATION"] == "9.0", np.nan, demographics["EDUCATION"]
)
demographics["EDUCATION"] = np.where(
    demographics["EDUCATION"] == "nan", np.nan, demographics["EDUCATION"]
)

demographics["MARITAL_STATUS"] = np.where(
    demographics["MARITAL_STATUS"] == 1,
    "Married/Living with partner",
    demographics["MARITAL_STATUS"],
)

```

```

demographics["MARITAL_STATUS"] = np.where(
    demographics["MARITAL_STATUS"] == "2.0",
    "Widowed/Divorced/Separated",
    demographics["MARITAL_STATUS"],
)
demographics["MARITAL_STATUS"] = np.where(
    demographics["MARITAL_STATUS"] == "3.0",
    "Never married",
    demographics["MARITAL_STATUS"],
)
demographics["MARITAL_STATUS"] = np.where(
    demographics["MARITAL_STATUS"] == "77.0", "Refused",
    demographics["MARITAL_STATUS"]
)
demographics["MARITAL_STATUS"] = np.where(
    demographics["MARITAL_STATUS"] == "99", np.nan,
    demographics["MARITAL_STATUS"]
)
demographics["MARITAL_STATUS"] = np.where(
    demographics["MARITAL_STATUS"] == "nan", np.nan,
    demographics["MARITAL_STATUS"]
)

# Day 1 Intake

day_1_intake["RECALL_RELIABILITY"] = np.where(
    day_1_intake["RECALL_RELABILITY"] == 1,
    "Reliable and met the minimum criteria",
    day_1_intake["RECALL_RELABILITY"],
)
day_1_intake["RECALL_RELABILITY"] = np.where(
    day_1_intake["RECALL_RELABILITY"] == "2.0",
    "Not reliable or not met the minimum criteria",
    day_1_intake["RECALL_RELABILITY"],
)
day_1_intake["RECALL_RELABILITY"] = np.where(
    day_1_intake["RECALL_RELABILITY"] == "3.0",
    "Reported consuming breast-milk",
    day_1_intake["RECALL_RELABILITY"],
)
day_1_intake["RECALL_RELABILITY"] = np.where(
    day_1_intake["RECALL_RELABILITY"] == "4.0",
    "Reported consuming breast-milk",
    day_1_intake["RECALL_RELABILITY"],
)
day_1_intake["RECALL_RELABILITY"] = np.where(
    day_1_intake["RECALL_RELABILITY"] == "5.0",
    "Not done",
)

```

```

        day_1_intake["RECALL_RELIABILITY"] ,
    )

day_1_intake["INFANT_BREAST_FED"] = np.where(
    day_1_intake["INFANT_BREAST_FED"] == 1, "Yes",
    day_1_intake["INFANT_BREAST_FED"]
)
day_1_intake["INFANT_BREAST_FED"] = np.where(
    day_1_intake["INFANT_BREAST_FED"] == "2.0", "No",
    day_1_intake["INFANT_BREAST_FED"]
)
day_1_intake["INFANT_BREAST_FED"] = np.where(
    day_1_intake["INFANT_BREAST_FED"] == "nan",
    np.nan,
    day_1_intake["INFANT_BREAST_FED"],
)
day_1_intake["INTAKE_DAY_WEEK"] = np.where(
    day_1_intake["INTAKE_DAY_WEEK"] == 1, "Sunday",
    day_1_intake["INTAKE_DAY_WEEK"]
)
day_1_intake["INTAKE_DAY_WEEK"] = np.where(
    day_1_intake["INTAKE_DAY_WEEK"] == "2.0", "Monday",
    day_1_intake["INTAKE_DAY_WEEK"]
)
day_1_intake["INTAKE_DAY_WEEK"] = np.where(
    day_1_intake["INTAKE_DAY_WEEK"] == "3.0", "Tuesday",
    day_1_intake["INTAKE_DAY_WEEK"]
)
day_1_intake["INTAKE_DAY_WEEK"] = np.where(
    day_1_intake["INTAKE_DAY_WEEK"] == "4.0",
    "Wednesday",
    day_1_intake["INTAKE_DAY_WEEK"],
)
day_1_intake["INTAKE_DAY_WEEK"] = np.where(
    day_1_intake["INTAKE_DAY_WEEK"] == "5.0",
    "Thursday",
    day_1_intake["INTAKE_DAY_WEEK"],
)
day_1_intake["INTAKE_DAY_WEEK"] = np.where(
    day_1_intake["INTAKE_DAY_WEEK"] == "6.0", "Friday",
    day_1_intake["INTAKE_DAY_WEEK"]
)
day_1_intake["INTAKE_DAY_WEEK"] = np.where(
    day_1_intake["INTAKE_DAY_WEEK"] == "7.0",
    "Saturday",
    day_1_intake["INTAKE_DAY_WEEK"],
)

```

```

)
day_1_intake["INTAKE_DAY_WEEK"] = np.where(
    day_1_intake["INTAKE_DAY_WEEK"] == "nan", np.nan,
    day_1_intake["INTAKE_DAY_WEEK"]
)

relationship_map = {
    1: "SP",
    2: "Mother of SP",
    3: "Father of SP",
    5: "Spouse of SP",
    6: "Child of SP",
    7: "Grandparent of SP",
    8: "Friend, partner, non-relative",
    9: "Translator, not a HH member",
    10: "Child care provider, caretaker",
    11: "Other relative",
    77: "Refused",
    99: np.nan,
}
day_1_intake["MAIN RESPONDENT"] =
    day_1_intake["MAIN RESPONDENT"].map(relationship_map)

salt_usage_map = {
    1: "Ordinary salt",
    2: "Lite salt",
    3: "Salt substitute",
    4: "Doesn't use or add salt products at the table",
    91: "Other",
    99: np.nan,
}
day_1_intake["SALT_AT_TABLE"] =
    day_1_intake["SALT_AT_TABLE"].map(salt_usage_map)

salt_usage_table_map = {
    1: "Rarely",
    2: "Occasionally",
    3: "Very often",
    7: "Refused",
    9: "Don't know",
    99: np.nan,
}
day_1_intake["SALT_AT_TABLE"] =
    day_1_intake["SALT_AT_TABLE"].map(salt_usage_table_map)

```

```

salt_usage_prep_map = {
    1: "Never",
    2: "Rarely",
    3: "Occasionally",
    4: "Very often",
    9: "Don't know",
    99: np.nan,
}

day_1_intake["SALT_IN_PREP"] =
    day_1_intake["SALT_IN_PREP"].map(salt_usage_prep_map)

day_1_intake["SPECIAL_DIET"] = np.where(
    day_1_intake["SPECIAL_DIET"] == 1, "Yes", day_1_intake["SPECIAL_DIET"]
)
day_1_intake["SPECIAL_DIET"] = np.where(
    day_1_intake["SPECIAL_DIET"] == "2.0", "No", day_1_intake["SPECIAL_DIET"]
)

day_1_intake["WEIGHT_LOSS_DIET"] = np.where(
    day_1_intake["WEIGHT_LOSS_DIET"] == 1, "Yes",
    day_1_intake["WEIGHT_LOSS_DIET"]
)
day_1_intake["WEIGHT_LOSS_DIET"] = np.where(
    day_1_intake["WEIGHT_LOSS_DIET"] == "nan", np.nan,
    day_1_intake["WEIGHT_LOSS_DIET"]
)

day_1_intake["LOW_FAT_DIET"] = np.where(
    day_1_intake["LOW_FAT_DIET"] == 2, "Yes", day_1_intake["LOW_FAT_DIET"]
)
day_1_intake["LOW_FAT_DIET"] = np.where(
    day_1_intake["LOW_FAT_DIET"] == "nan", np.nan, day_1_intake["LOW_FAT_DIET"]
)

day_1_intake["LOW_SALT_DIET"] = np.where(
    day_1_intake["LOW_SALT_DIET"] == 3, "Yes", day_1_intake["LOW_SALT_DIET"]
)
day_1_intake["LOW_SALT_DIET"] = np.where(
    day_1_intake["LOW_SALT_DIET"] == "nan", np.nan,
    day_1_intake["LOW_SALT_DIET"]
)

day_1_intake["LOW_SUGER_DIET"] = np.where(
    day_1_intake["LOW_SUGER_DIET"] == 4, "Yes", day_1_intake["LOW_SALT_DIET"]
)
day_1_intake["LOW_SUGER_DIET"] = np.where(
    day_1_intake["LOW_SUGER_DIET"] == "nan", np.nan,
    day_1_intake["LOW_SALT_DIET"]
)

```

```

)
day_1_intake["LOW_FIBER_DIET"] = np.where(
    day_1_intake["LOW_FIBER_DIET"] == 5, "Yes", day_1_intake["LOW_FIBER_DIET"]
)
day_1_intake["LOW_FIBER_DIET"] = np.where(
    day_1_intake["LOW_FIBER_DIET"] == "nan", np.nan,
    day_1_intake["LOW_FIBER_DIET"]
)

day_1_intake["HIGH_FIBER_DIET"] = np.where(
    day_1_intake["HIGH_FIBER_DIET"] == 6, "Yes",
    day_1_intake["HIGH_FIBER_DIET"]
)
day_1_intake["HIGH_FIBER_DIET"] = np.where(
    day_1_intake["HIGH_FIBER_DIET"] == "nan", np.nan,
    day_1_intake["HIGH_FIBER_DIET"]
)

day_1_intake["CELIAC_DIET"] = np.where(
    day_1_intake["CELIAC_DIET"] == 6, "Yes", day_1_intake["CELIAC_DIET"]
)
day_1_intake["CELIAC_DIET"] = np.where(
    day_1_intake["CELIAC_DIET"] == "nan", np.nan, day_1_intake["CELIAC_DIET"]
)

```

Standardise Measurements

```

day_1_intake["TOTAL_WATER_DRANK"] = day_1_intake["TOTAL_WATER_DRANK"]/1000
day_1_intake.rename(columns = {"TOTAL_WATER_DRANK": "TOTAL_WATER_DRANK_LITRE"},  

    inplace= True)

```

```

weight_cols = ['SEQN', 'BMXWT', 'BMXHT']
weight = measures[weight_cols]

weight.rename(columns={"SEQN": "SUBJID", "BMXWT" : "WEIGHT", "BMXHT" :  

    "HEIGHT"}, inplace=True)

```

```

/tmp/ipykernel_85268/3893318471.py:5: SettingWithCopyWarning:  

A value is trying to be set on a copy of a slice from a DataFrame

```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace

```
master = pd.merge(demographics, day_1_intake, how="right", on="SUBJID")
master = pd.merge(master, weight, how="right", on="SUBJID")
```

Create Age Groups

```
conditions = [
    master["AGE_YEARS_SCREENING"] < 18,
    (master["AGE_YEARS_SCREENING"] >= 18) & (master["AGE_YEARS_SCREENING"] <=
    ↵ 35),
    (master["AGE_YEARS_SCREENING"] > 35) & (master["AGE_YEARS_SCREENING"] <=
    ↵ 60),
    master["AGE_YEARS_SCREENING"] > 60
]

choices = [
    "Less than 18",
    "18-35",
    "36-60",
    "Above 60"
]

master["AGE_GROUP"] = np.select(conditions, choices, default= "NaN")

import os

master.to_csv(r'/home/pgr16/Documents/Data_Analysis/Obesity_Analysis_and_Prediction/Data/Master.c
    ↵
```

Exploratory Analysis

```
plt.figure()
sns.heatmap(master.isnull(), cbar=False, cmap="viridis")
plt.xlabel("Columns")
plt.ylabel("Rows")
plt.show()
```

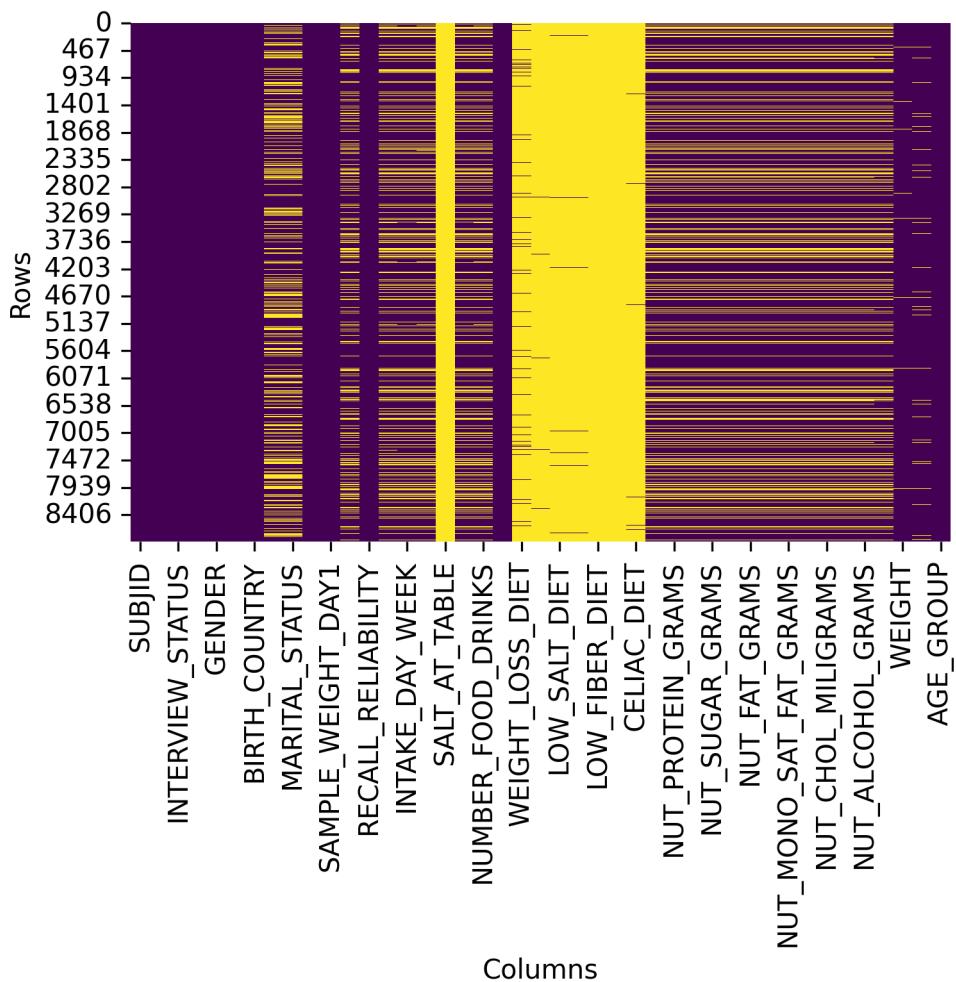
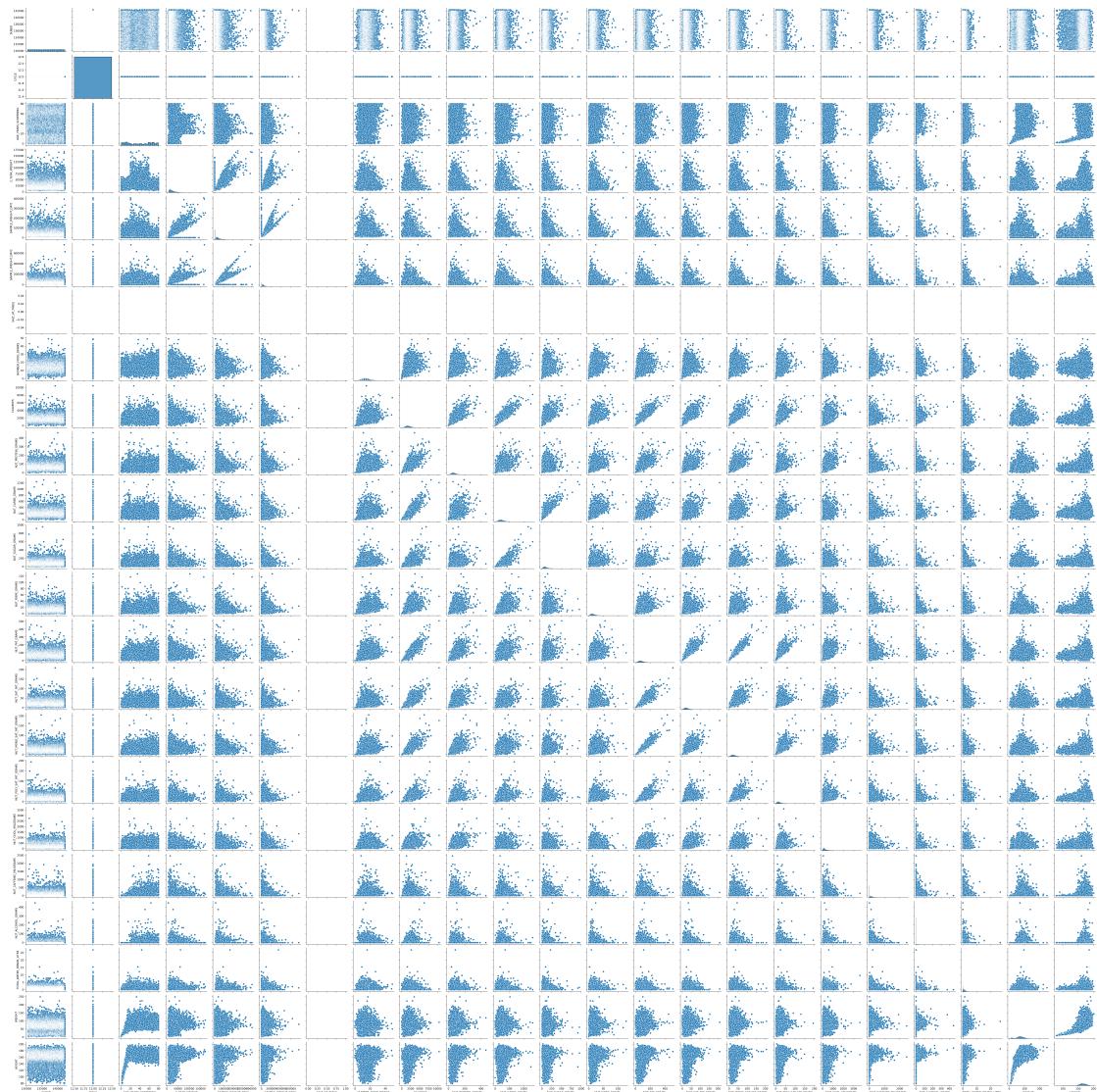


Figure 1: Missing Values Heatmap

```
plt.figure()
sns.pairplot(master)
plt.show()
```

<Figure size 1650x1050 with 0 Axes>

Pairplot of Data

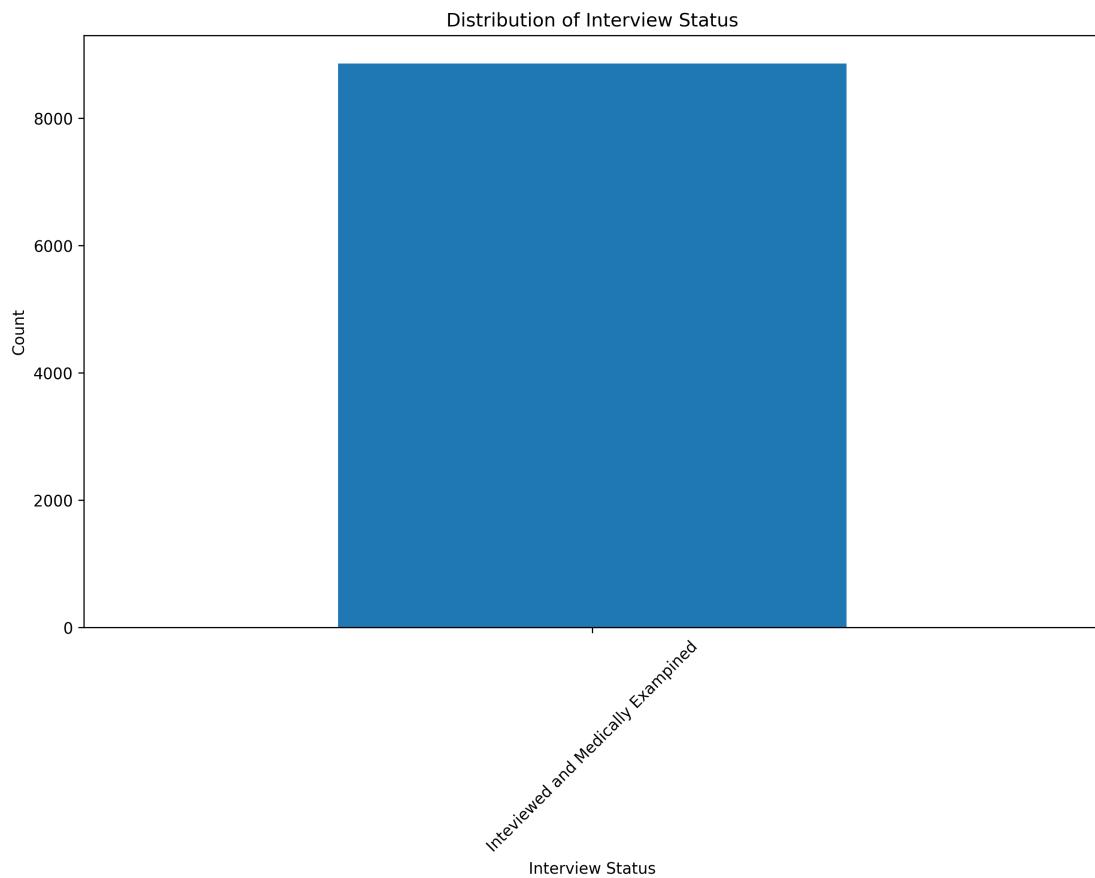


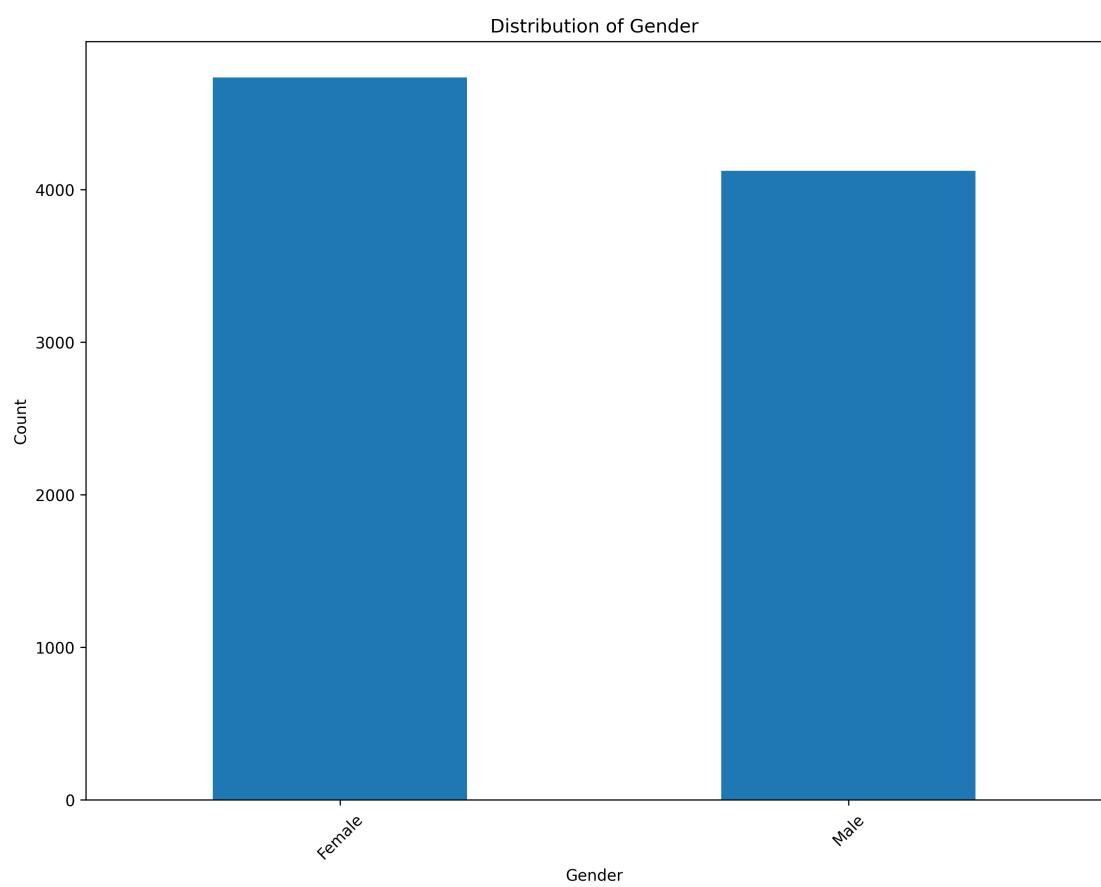
Visualisation of Categorical Variables

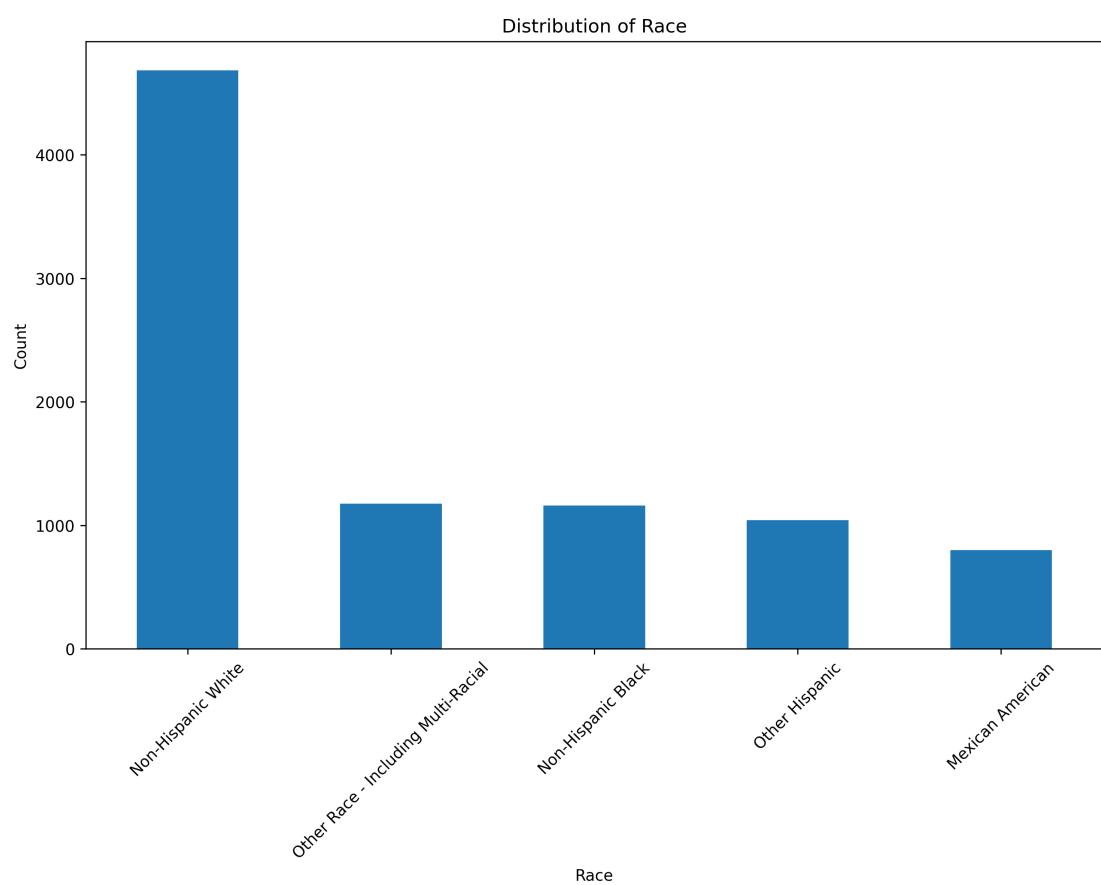
```
categorical_columns = [
    col for col in master.select_dtypes(include=["object", "category"]).columns
    if not col.endswith("_DIET")
]

for col in categorical_columns:

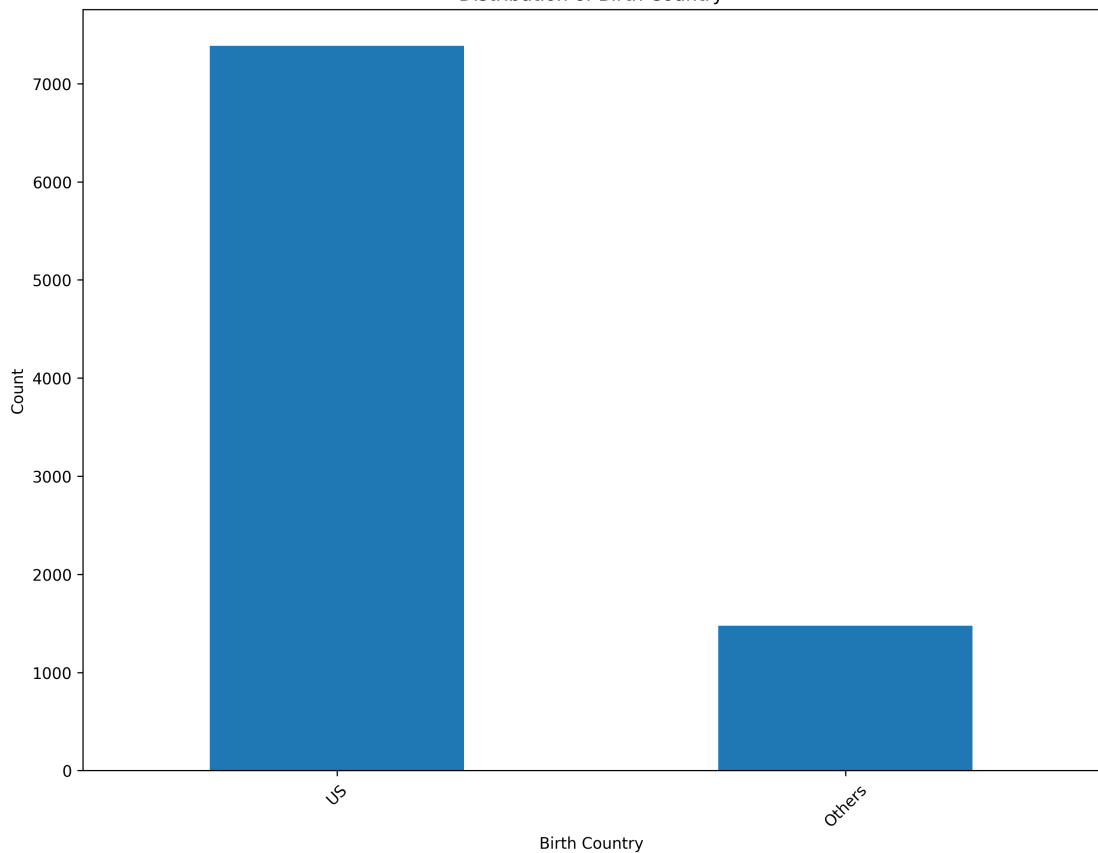
    plt.figure(figsize=(10, 8))
    master[col].value_counts(dropna=False).plot(kind='bar')
    plt.title(f"Distribution of {col.replace('_', ' ').title()}")
    plt.ylabel("Count")
    plt.xlabel(f"{col.replace('_', ' ').title()}")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

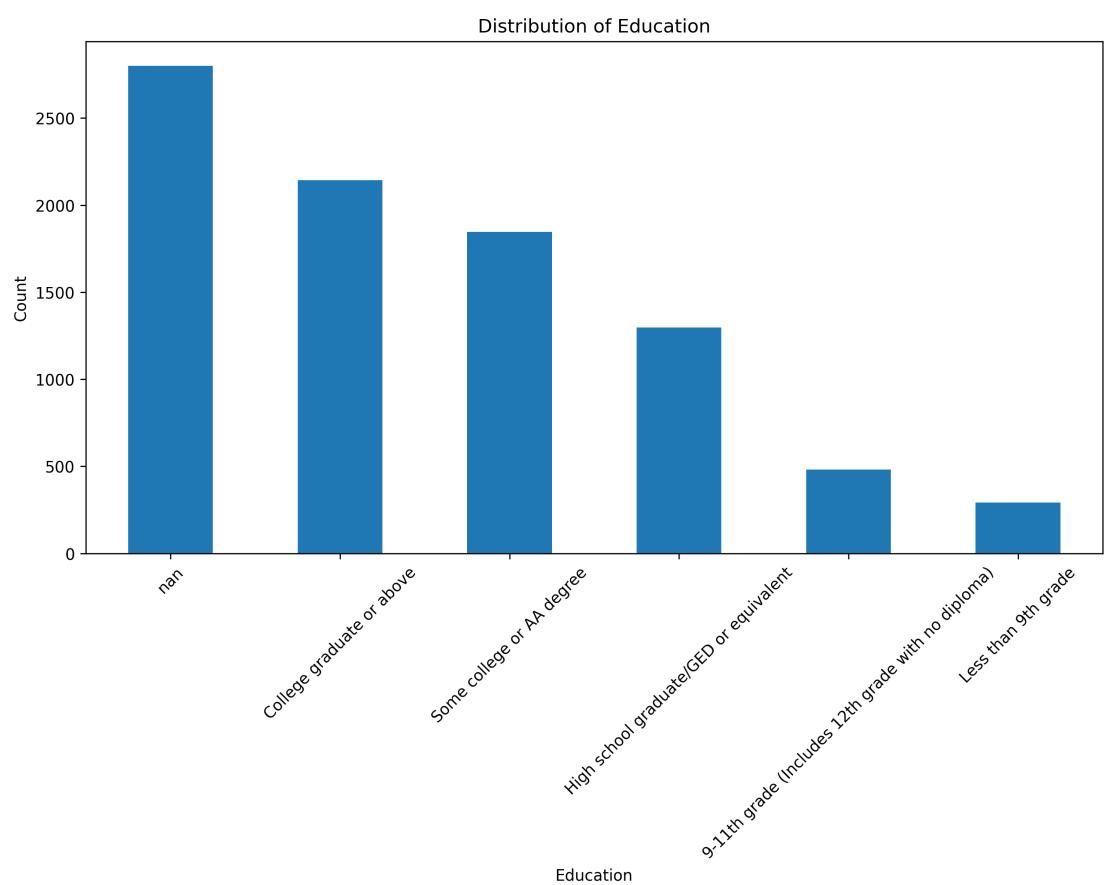


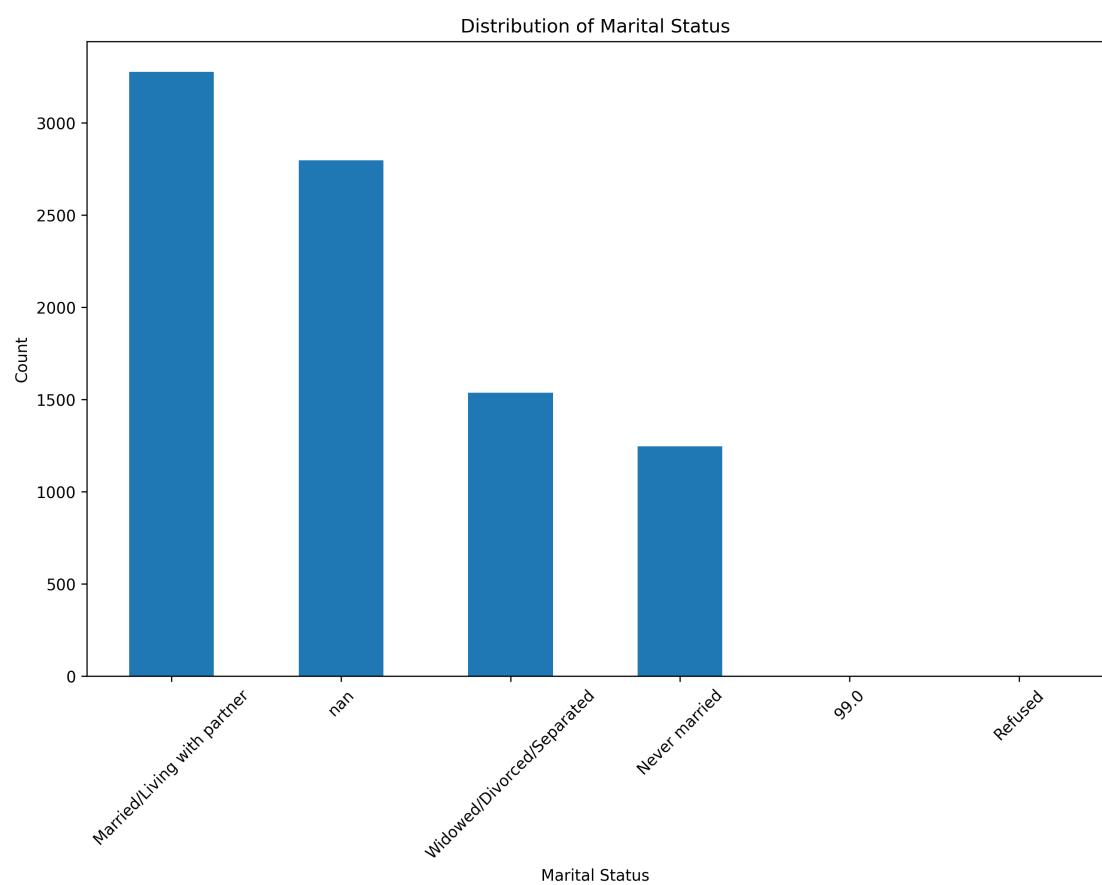


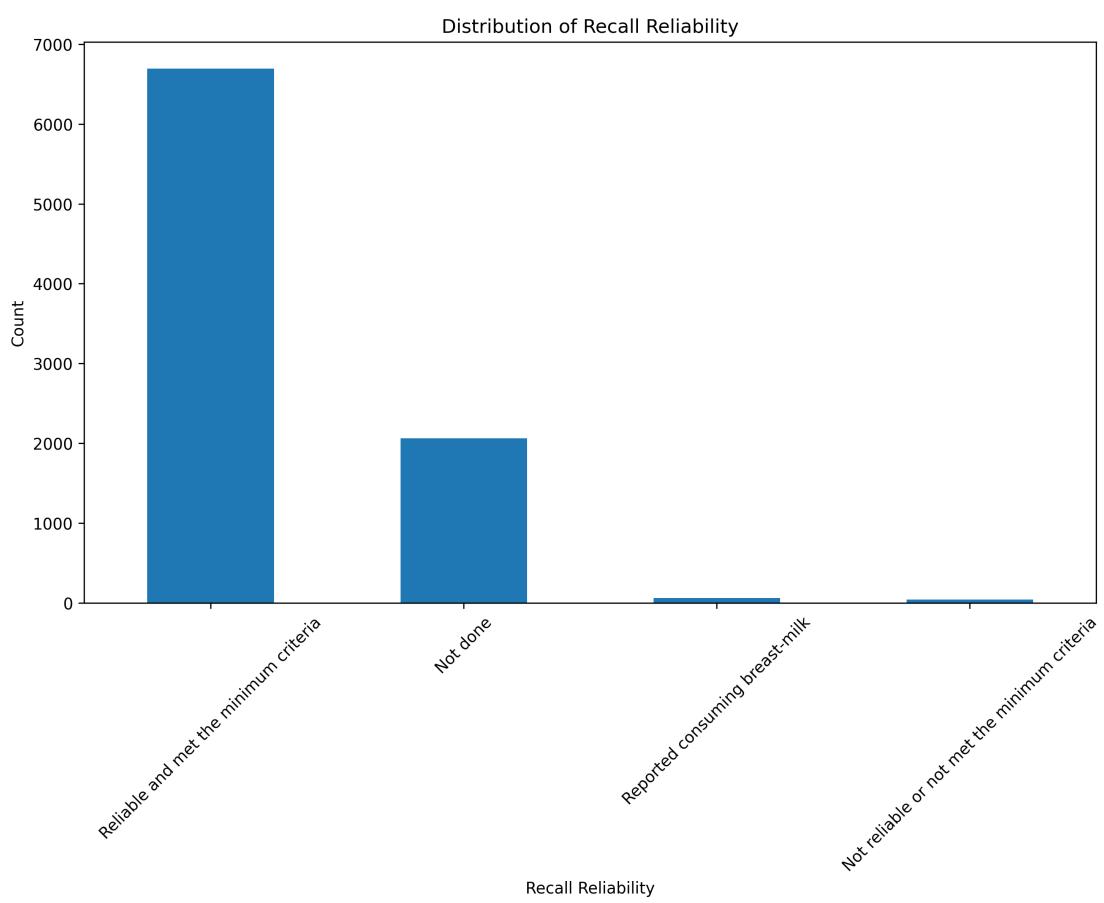


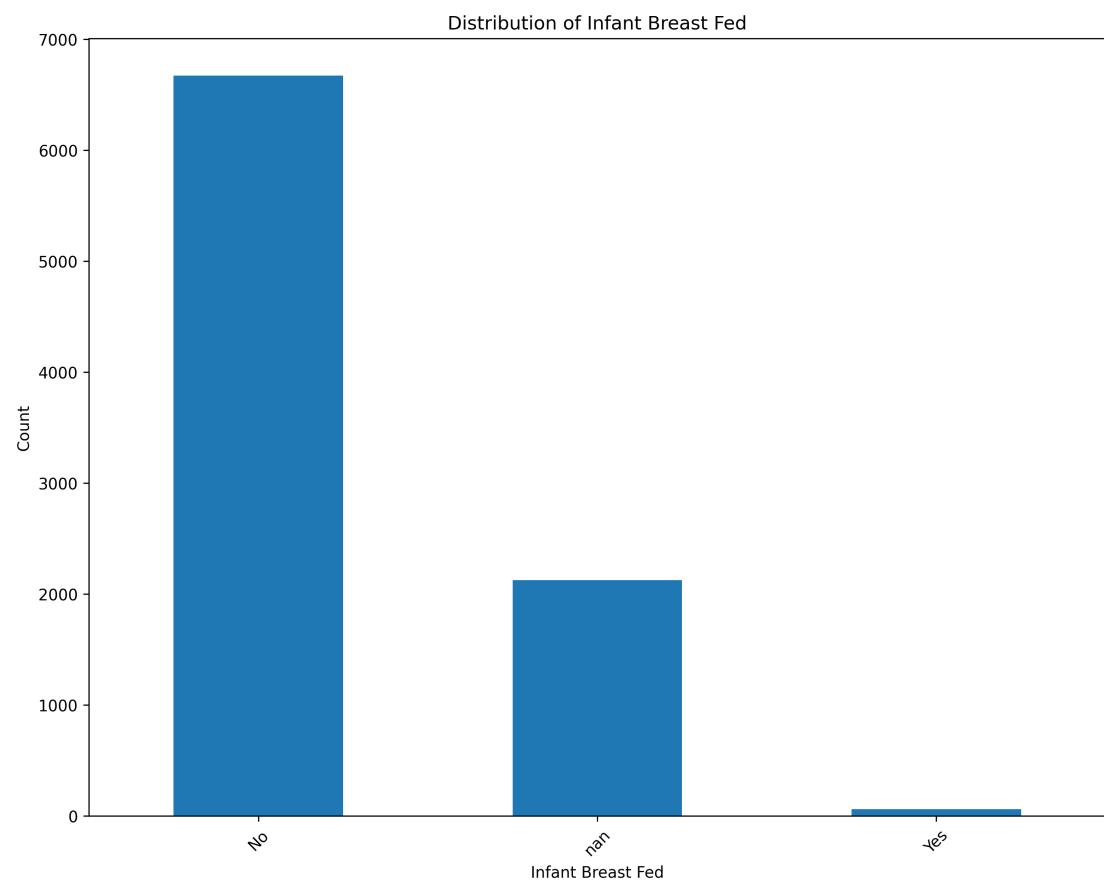
Distribution of Birth Country

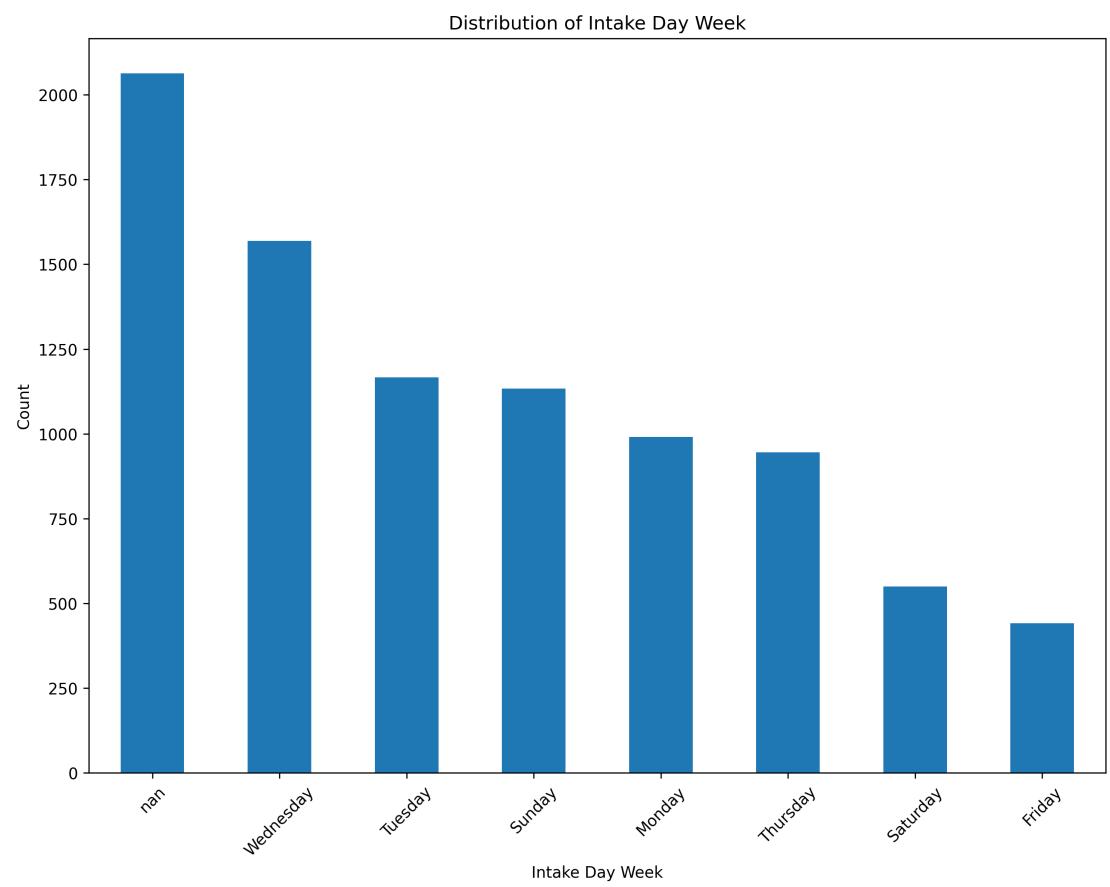


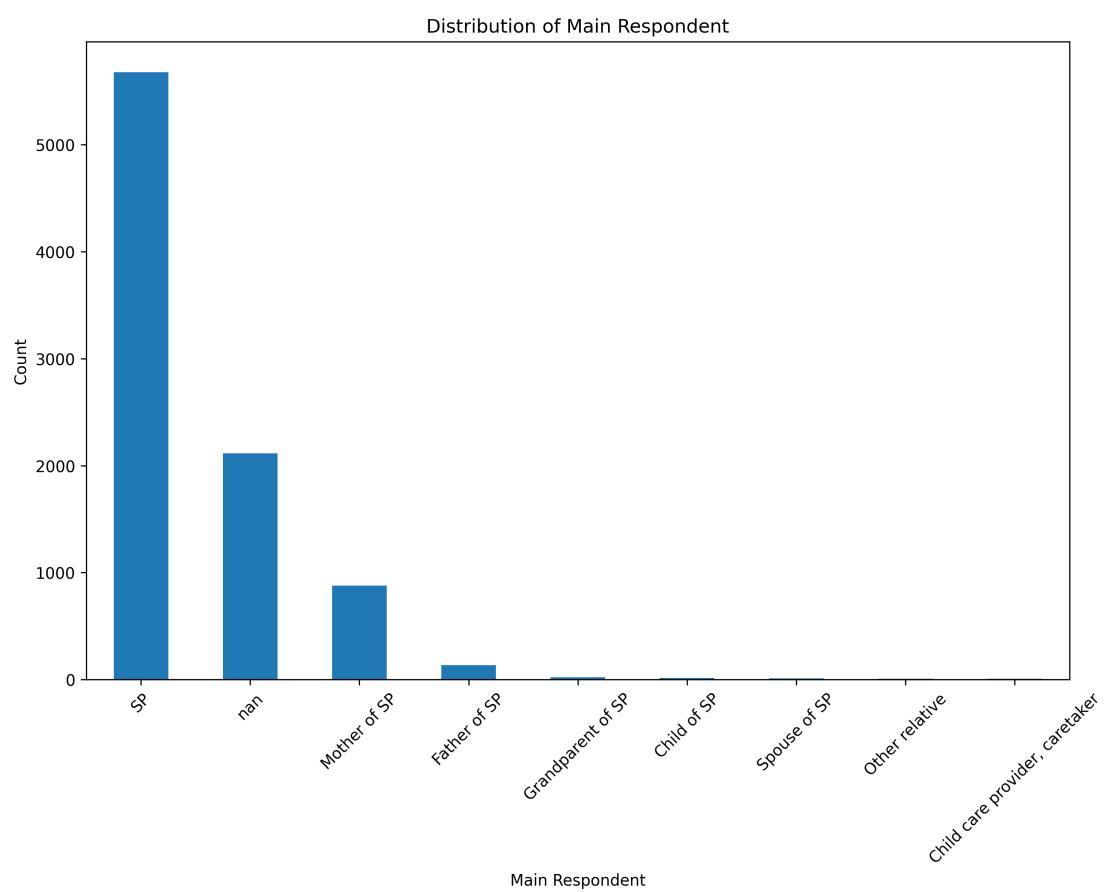




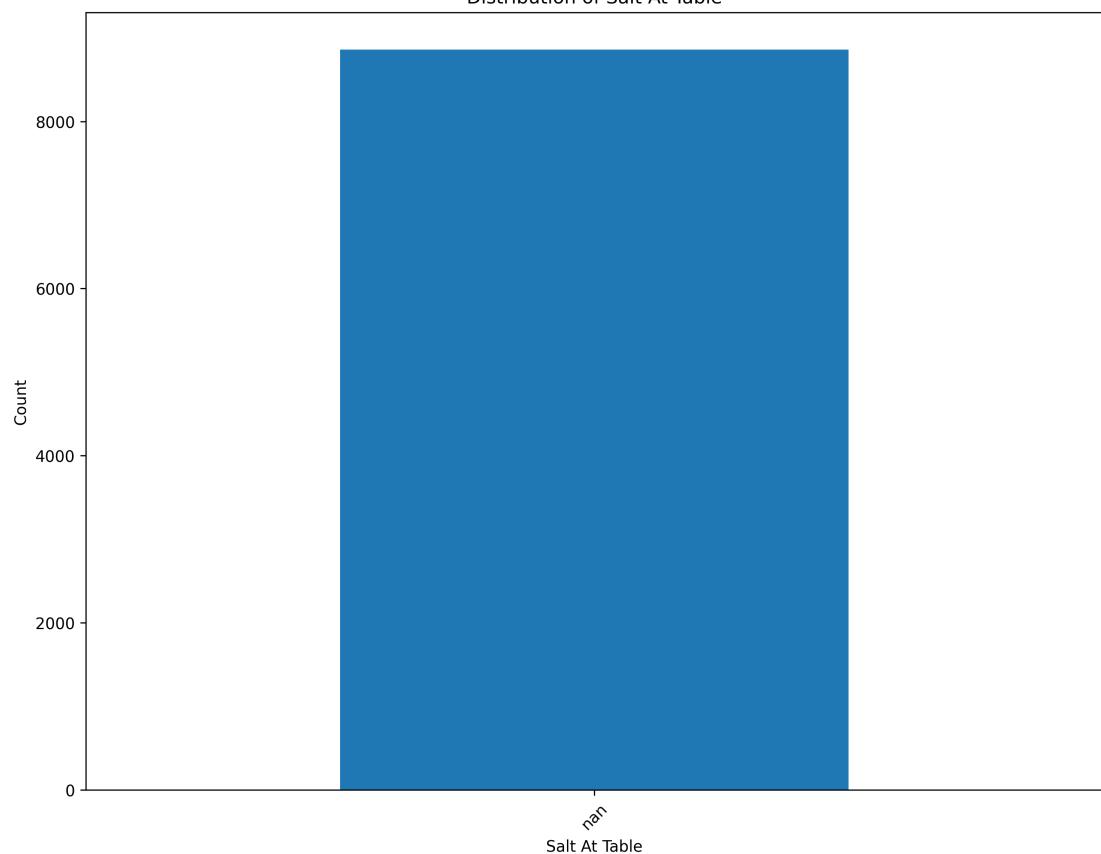


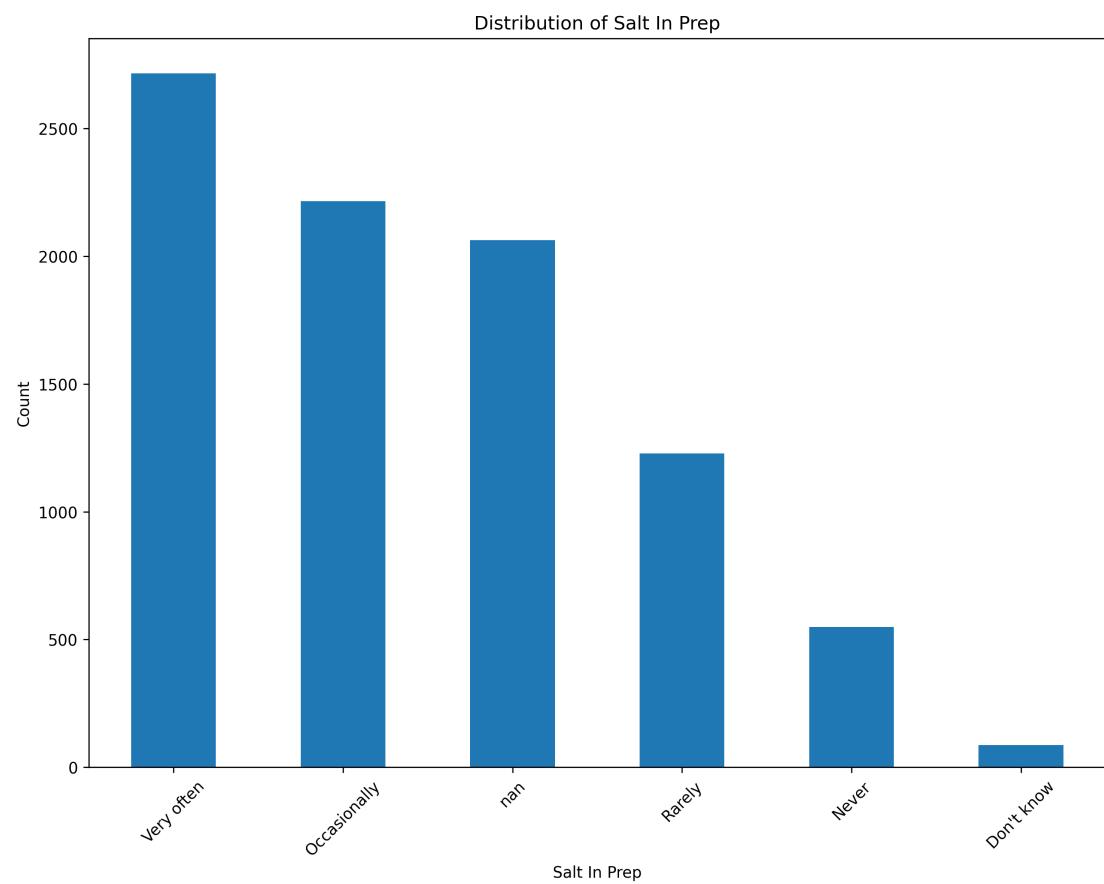


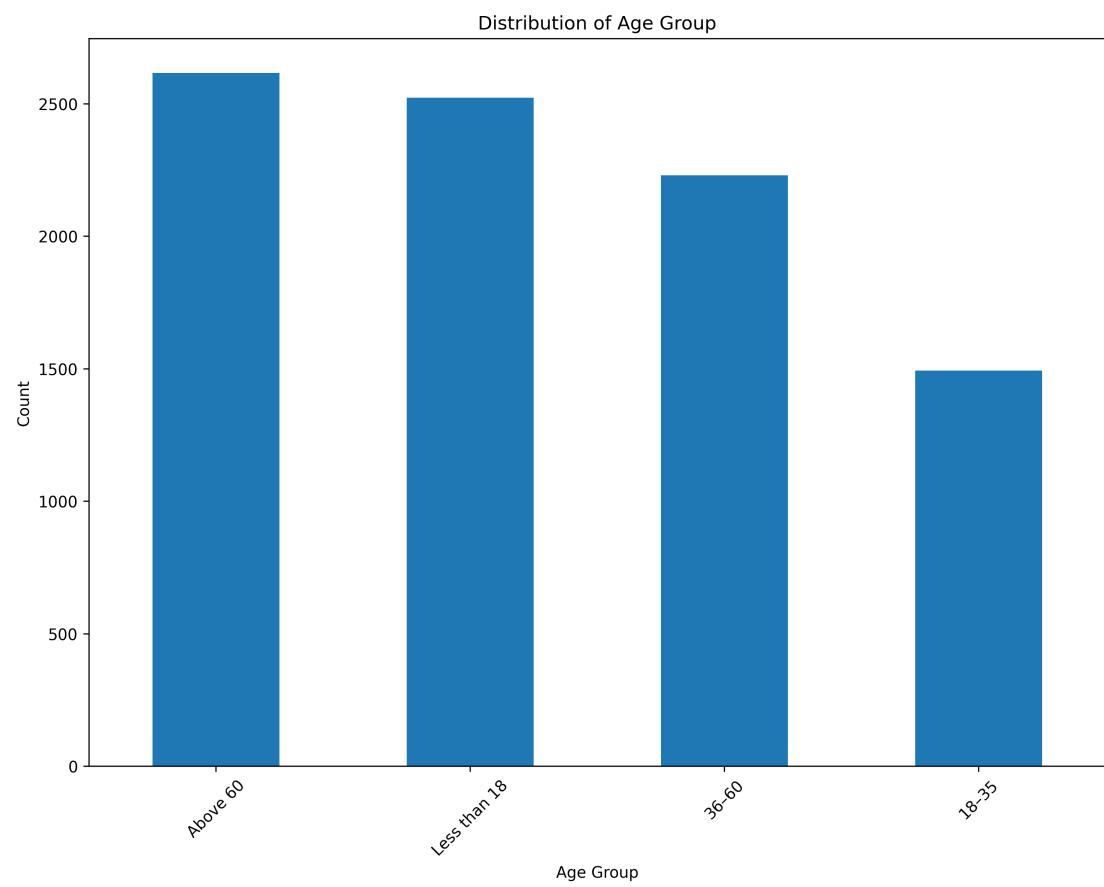




Distribution of Salt At Table







Data Analysis

Calorie Intake per Ethnicity

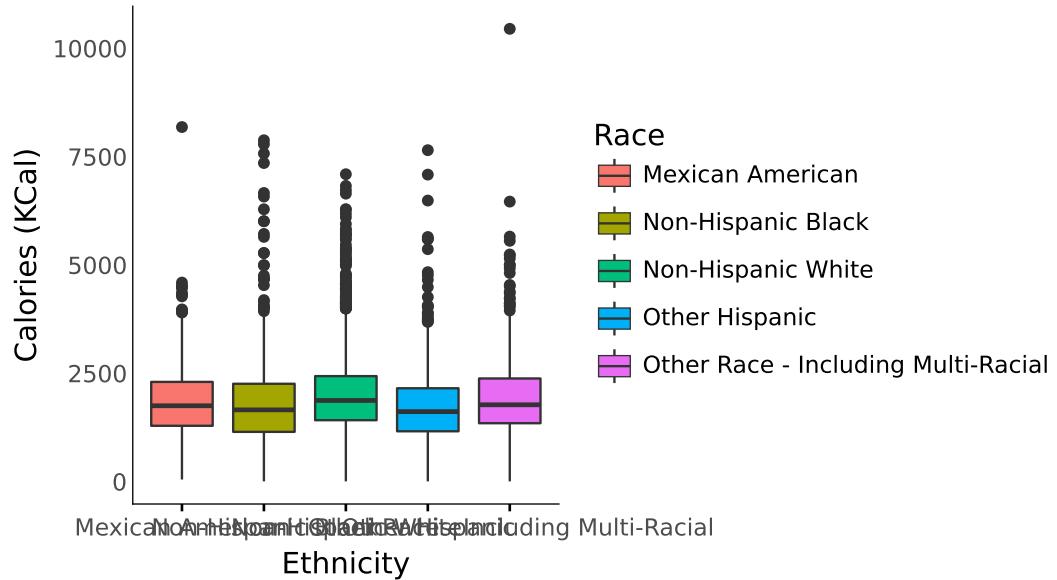
```
from plotnine import *

calorie = master[master["RECALL_RELIABILITY"]== "Reliable and met the minimum
↪ criteria"]
calorie = calorie[calorie["INFANT_BREAST_FED"]== "No"]

title = "Boxplot of Calories per ethnic group"
p1 = (
    ggplot(calorie) +
    geom_boxplot(aes(x = "RACE", y = "CALORIES", fill = "RACE")) +
    theme_minimal() +
    labs(title = title,
        y = "Calories (KCal)",
        x = "Ethnicity",
        fill = "Race",
        caption = "Population comes from only the surveys classified as 'Reliable
↪ and met the minimum criteria'" +
    theme(
        axis_line_y=element_line(color="black", size=0.5),
        axis_line_x=element_line(color="black", size=0.5),
        axis_ticks_x=element_text(),
        panel_grid_major=element_blank(),
        panel_grid_minor=element_blank()
    )
)

p1
```

Boxplot of Calories per ethnic group



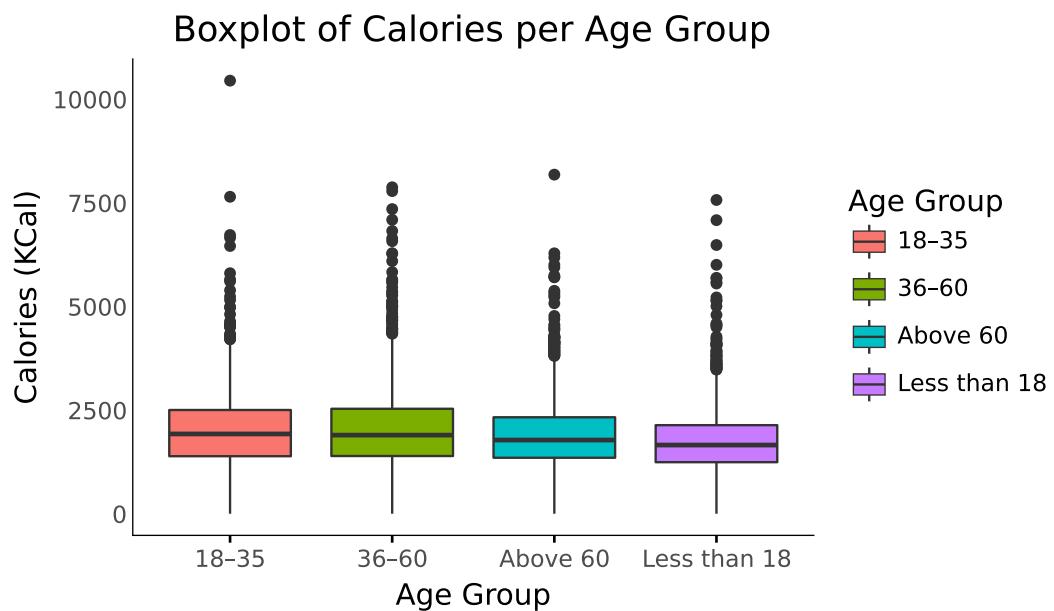
Population comes from only the surveys classified as 'Reliable and met the minimum criteria'

Box Plot of Calorie Intake per Age Group

```
title = "Boxplot of Calories per Age Group"
p2 = (
  ggplot(calorie) +
  geom_boxplot(aes(x = "AGE_GROUP", y = "CALORIES", fill = "AGE_GROUP")) +
  theme_minimal() +
  labs(title = title,
       y = "Calories (KCal)",
       x = "Age Group",
       fill = "Age Group",
       caption = "\n Population comes from only the surveys classified as 'Reliable
       ↵ and met the minimum criteria'" +
  theme(
    axis_line_y=element_line(color="black", size=0.5),
    axis_line_x=element_line(color="black", size=0.5),
    axis_ticks_x=element_text(),
    panel_grid_major=element_blank(),
    panel_grid_minor=element_blank()
  )
)
```

)

p2



only the surveys classified as 'Reliable and met the minimum criteria'

Weighted Analysis

```
from IPython.display import Markdown
from tabulate import tabulate

def weighted_mean_versus_normal_mean(data, value_col, weight_col,
                                      characteristic=None):
    weighted = (data[value_col] * data[weight_col]).sum() /
    data[weight_col].sum()
    normal = data[value_col].mean()
    thing = value_col.replace("NUT_", "")
    thing = thing.replace("_", " ").title()

    if thing.endswith("Grams"):
        thing = thing.replace("Grams", "(g)")
    elif thing.endswith("Miligrams"):
        thing = thing.replace("Miligrams", "(mg)")

    return {
        "Characteristic": thing,
        "Unweighted Mean": round(normal, 2),
        "Weighted Mean": round(weighted, 2),
    }

nutrient_cols = [
    col for col in master if col.startswith(("NUT_")) or col ==
    "SAMPLE_WEIGHT_DAY1"
]

nutrient_cols

table = []

for col in nutrient_cols:
    if col != "SAMPLE_WEIGHT_DAY1": # skip weight column itself
        result = weighted_mean_versus_normal_mean(
            master, col, "SAMPLE_WEIGHT_DAY1", col
        )
        table.append(result)

table

[{'Characteristic': 'Protein (g)',  
 'Unweighted Mean': np.float64(70.66),
```

```
'Weighted Mean': np.float64(72.98)},  
{'Characteristic': 'Carbs (g)',  
 'Unweighted Mean': np.float64(223.91),  
 'Weighted Mean': np.float64(227.62)},  
{'Characteristic': 'Sugar (g)',  
 'Unweighted Mean': np.float64(97.06),  
 'Weighted Mean': np.float64(97.13)},  
{'Characteristic': 'Fibre (g)',  
 'Unweighted Mean': np.float64(15.52),  
 'Weighted Mean': np.float64(15.66)},  
{'Characteristic': 'Fat (g)',  
 'Unweighted Mean': np.float64(79.95),  
 'Weighted Mean': np.float64(82.08)},  
{'Characteristic': 'Sat Fat (g)',  
 'Unweighted Mean': np.float64(26.03),  
 'Weighted Mean': np.float64(26.75)},  
{'Characteristic': 'Mono Sat Fat (g)',  
 'Unweighted Mean': np.float64(26.89),  
 'Weighted Mean': np.float64(27.57)},  
{'Characteristic': 'Poly Sat Fat (g)',  
 'Unweighted Mean': np.float64(18.38),  
 'Weighted Mean': np.float64(18.86)},  
{'Characteristic': 'Chol (mg)',  
 'Unweighted Mean': np.float64(285.87),  
 'Weighted Mean': np.float64(296.91)},  
{'Characteristic': 'Caffeine (mg)',  
 'Unweighted Mean': np.float64(114.79),  
 'Weighted Mean': np.float64(116.0)},  
{'Characteristic': 'Alcohol (g)',  
 'Unweighted Mean': np.float64(5.5),  
 'Weighted Mean': np.float64(6.18)}]
```