VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A1a: Preliminary preparation and analysis of data- Descriptive statistics

ADHYAYAN AMIT JAIN

V01109421

Date of Submission: 16-06-2024

CONTENTS

Analyzing Consumption in the State of Madhya Pradesh

INTRODUCTION

This study centers on the state of Madhya Pradesh, utilizing data from the National Sample Survey Office (NSSO) to pinpoint the top three and bottom three districts in terms of consumption. In our analysis, we meticulously clean and manipulate the dataset to extract the necessary information. The dataset encompasses consumption-related details, spanning both rural and urban sectors, and includes variations across different districts. We leverage Python and R, robust programming languages known for its effectiveness in managing and analyzing extensive datasets, to conduct this analysis.

Our objectives include identifying missing values, addressing outliers, standardizing district and sector names, summarizing consumption data regionally and district-wise, and testing the significance of mean differences. The findings from this study can inform policymakers and stakeholders, fostering targeted interventions and promoting equitable development across the state.

OBJECTIVES

The primary objectives of this study are to:

1. Identify and rectify any missing values within the dataset by replacing them with the mean values of the respective variables.
2. Detect and address any outliers, describing the outcomes of these tests and making appropriate amendments.
3. Standardize the names of districts and sectors (rural and urban) to ensure consistency.

4. Summarize consumption data both regionally and by district, identifying the top three and bottom three districts based on consumption levels.
5. Test the significance of differences in mean consumption between different regions to determine if the observed differences are statistically significant.

BUSINESS SIGNIFICANCE

Understanding the consumption patterns in Madhya Pradesh, as derived from NSSO data, holds substantial implications for businesses, policymakers, and stakeholders. Identifying the top and bottom three consuming districts provides critical insights that can guide market entry strategies, resource allocation, and supply chain management. By addressing data quality issues such as missing values and outliers, and conducting thorough statistical analyses, this study ensures accurate and reliable findings. These insights can inform targeted interventions, promoting balanced and equitable development across the state, ultimately fostering economic growth and improved living standards.

RESULTS AND INTERPRETATIONS

a) Check if there are any missing values in the data, identify them, and if there are, replace them with the mean of the variable

Python

Step 1: Identify Missing Values in the Full Dataset

First, we identify the number of missing values in each column of the original dataset `MP`.

```
# Check for missing values in the full dataset

missing_values_full = MP.isnull().sum().sort_values(ascending=False)
print(missing_values_full)
```

Output:

```
soyabean_q            4717
soyabean_v            4717
Land_Leased_out       4562
Otherwise_possessed   4552
```

```
Meals_Employer         4472
                       ...
cauli_q                   0
parwal_q                  0
bhindi_q                  0
chillig_q                 0
fv_tot                    0
Length: 384, dtype: int64
```

Step 2: Select Relevant Columns for Analysis

We create a new DataFrame `MP_new` containing only the columns of interest for the analysis.

```
# Select relevant columns

MP_new = MP[['state_1', 'District', 'Sector', 'Region', 'State_Region',
             'ricetotal_q', 'wheattotal_q', 'moong_q', 'Milktotal_q',
             'chicken_q', 'bread_q', 'foodtotal_q', 'Beveragestotal_v',
             'Meals_At_Home']]
```

Step 3: Identify Missing Values in the Selected Columns

Next, we check for missing values in the newly created DataFrame `MP_new`.

```
# Check for missing values in the selected columns

missing_values_new = MP_new.isnull().sum().sort_values(ascending=False)
print(missing_values_new)
```

Output:

```
Meals_At_Home       26
state_1              0
District             0
Sector               0
Region               0
State_Region         0
ricetotal_q          0
wheattotal_q         0
moong_q              0
Milktotal_q          0
chicken_q            0
bread_q              0
foodtotal_q          0
Beveragestotal_v     0
dtype: int64
```

Step 4: Handle Missing Values

We create a copy of `MP_new` called `MP_clean` and fill the missing values in the `Meals_At_Home` column with the mean of that column.

```
# Create a copy of MP_new

MP_clean = MP_new.copy()

# Fill missing values in the 'Meals_At_Home' column with the mean
```

```
MP_clean['Meals_At_Home'] =
MP_clean['Meals_At_Home'].fillna(MP_new['Meals_At_Home'].mean())
```

Step 5: Verify Missing Values Have Been Handled

Finally, we verify that there are no remaining missing values in the cleaned DataFrame `MP_clean`.

```
# Check for missing values in the cleaned dataset
missing_values_clean = MP_clean.isnull().sum().sort_values(ascending=False)
print(missing_values_clean)
```

Output:

```
state_1             False
District            False
Sector              False
Region              False
State_Region        False
ricetotal_q         False
wheattotal_q        False
moong_q             False
Milktotal_q         False
chicken_q           False
bread_q             False
foodtotal_q         False
Beveragestotal_v    False
Meals_At_Home       False
dtype: bool
```

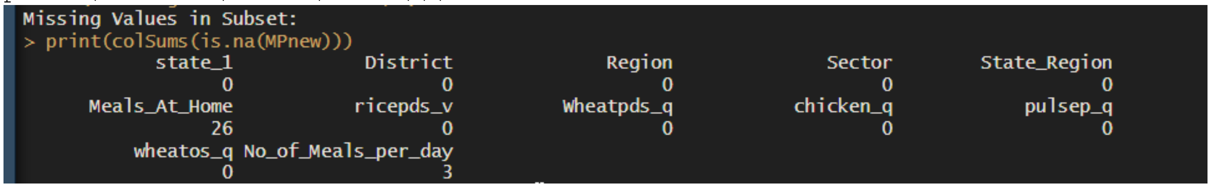
By following these steps, we have successfully identified, handled, and verified the missing values in the dataset, ensuring the data is clean and ready for further analysis.

R Program

Step 1: Identify Missing Values

Check for missing values in the subset of the data.

```
# Check for missing values in the subset
cat("Missing Values in Subset:\n")
print(colSums(is.na(MPnew)))
```



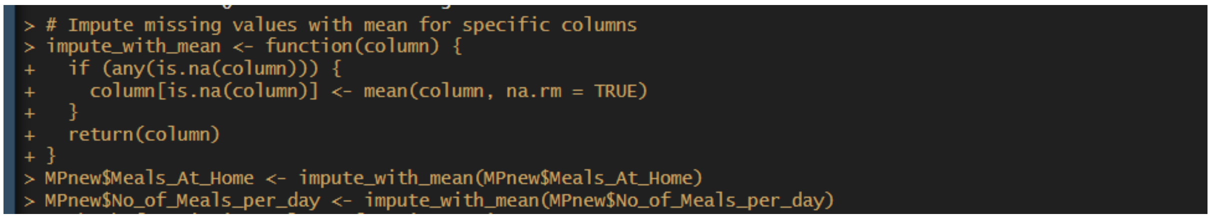
```
Missing Values in Subset:
> print(colSums(is.na(MPnew)))
          state_1          District            Region            Sector      State_Region
                0                 0                 0                 0                 0
    Meals_At_Home          ricepds_v        Wheatpds_q         chicken_q         pulsep_q
               26                 0                 0                 0                 0
        wheatos_q No_of_Meals_per_day
                0                 3
```

Step 2: Impute Missing Values with Mean

Impute missing values with the mean for specific columns.

```
# Impute missing values with mean for specific columns
```

```
impute_with_mean <- function(column) {
  if (any(is.na(column))) {
    column[is.na(column)] <- mean(column, na.rm = TRUE)
  }
  return(column)
}

MPnew$Meals_At_Home <- impute_with_mean(MPnew$Meals_At_Home)
MPnew$No_of_Meals_per_day <- impute_with_mean(MPnew$No_of_Meals_per_day)
```

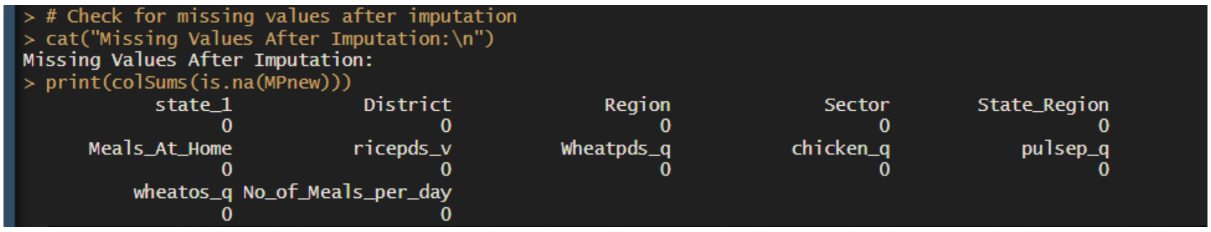


```
> # Impute missing values with mean for specific columns
> impute_with_mean <- function(column) {
+   if (any(is.na(column))) {
+     column[is.na(column)] <- mean(column, na.rm = TRUE)
+   }
+   return(column)
+ }
> MPnew$Meals_At_Home <- impute_with_mean(MPnew$Meals_At_Home)
> MPnew$No_of_Meals_per_day <- impute_with_mean(MPnew$No_of_Meals_per_day)
```

Step 3: Verify Missing Values Have Been Handled

Check for missing values in the dataset after imputation.

```
# Check for missing values after imputation

cat("Missing Values After Imputation:\n")
print(colSums(is.na(MPnew)))
```



```
> # Check for missing values after imputation
> cat("Missing Values After Imputation:\n")
Missing Values After Imputation:
> print(colSums(is.na(MPnew)))
            state_1            District              Region              Sector        State_Region 
                  0                   0                   0                   0                   0 
      Meals_At_Home            ricepds_v          Wheatpds_q           chicken_q             pulsep_q 
                  0                   0                   0                   0                   0 
          wheatos_q No_of_Meals_per_day 
                  0                   0 
```

b) Check for outliers, describe your test's outcome, and make suitable amendments.

Python

Step 1: Identify Outliers

Using the IQR (Interquartile Range) method, outliers are identified in the `ricetotal_q` column. The IQR method calculates the first quartile (Q1) and third quartile (Q3) and defines outliers as data points below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR.

```
import numpy as np

# Calculate IQR for ricetotal_q
rice1 = MP_clean['ricetotal_q'].quantile(0.25)
rice2 = MP_clean['ricetotal_q'].quantile(0.75)
iqr_rice = rice2 - rice1

# Define outlier thresholds
```

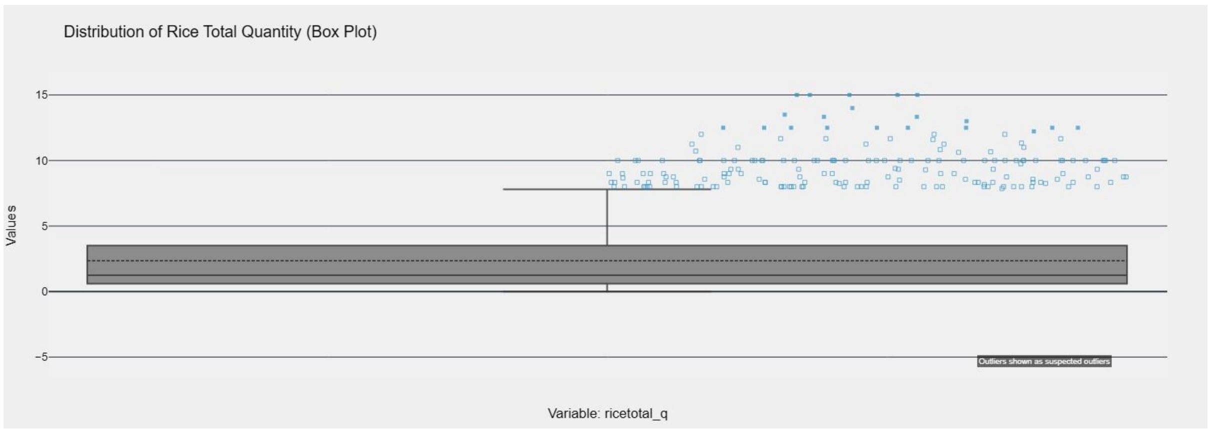
```
up_limit = rice2 + 1.5 * iqr_rice
low_limit = rice1 - 1.5 * iqr_rice

# Identify outliers
outliers = MP_clean[(MP_clean['ricetotal_q'] > up_limit) |
(MP_clean['ricetotal_q'] < low_limit)]
```

Step 2: Visualize Outliers

Create a box plot to visualize the distribution and outliers in the `ricetotal_q` column.

```
import plotly.express as px

# Create a box plot
fig = px.box(MP_clean, y='ricetotal_q')

# Customizing the box plot appearance
fig.update_traces(
    marker=dict(color='rgba(50, 150, 200, 0.7)', size=5, symbol='square'),
    line=dict(color='rgba(50, 50, 50, 0.8)', width=2),
    boxmean=True,
    notchwidth=0.2,
    whiskerwidth=0.2,
    boxpoints='suspectedoutliers',
    jitter=0.5,
    pointpos=0.5,
    hoverinfo='y+name',
    hoverlabel=dict(bgcolor='rgba(255, 255, 255, 0.9)', font_size=12)
)

# Customizing layout
fig.update_layout(
    title='Distribution of Rice Total Quantity (Box Plot)',
    xaxis_title='Variable: ricetotal_q',
    yaxis_title='Values',
    font=dict(family='Arial', size=14, color='black'),
    showlegend=False,
    plot_bgcolor='rgba(240, 240, 240, 0.7)',
    paper_bgcolor='rgba(240, 240, 240, 0.7)',
    margin=dict(l=50, r=50, t=80, b=50),
    hovermode='closest',
    template='plotly_dark'
)

# Adding annotations
fig.add_annotation(
    text='Outliers shown as suspected outliers',
    xref='paper', yref='paper',
    x=0.95, y=0.05,
    showarrow=False,
    align='right',
    font=dict(size=10, color='white'),
    bgcolor='rgba(0, 0, 0, 0.7)',
    bordercolor='rgba(0, 0, 0, 0.5)',
    borderwidth=1,
    opacity=0.8
)

# Displaying the updated box plot
```

```
fig.show()
```



Step 3: Remove Outliers

Remove the identified outliers from the dataset.

```
# Remove outliers from the dataset
MP_clean = MP_clean[(MP_clean['ricetotal_q'] <= up_limit) &
(MP_clean['ricetotal_q'] >= low_limit)]
```

Outcome of the Test

- **Method Used**: The IQR method was used to identify outliers in the `ricetotal_q` column.
- **Columns Checked**: The column `ricetotal_q` was checked for outliers.
- **Outcome**: Outliers were identified and visualized using a box plot. Subsequently, these outliers were removed from the dataset to ensure more accurate analysis.
- **Amendments**: After removing outliers, the dataset was summarized again to ensure the amendments were correctly applied. This ensures the integrity and reliability of the data for further analysis.

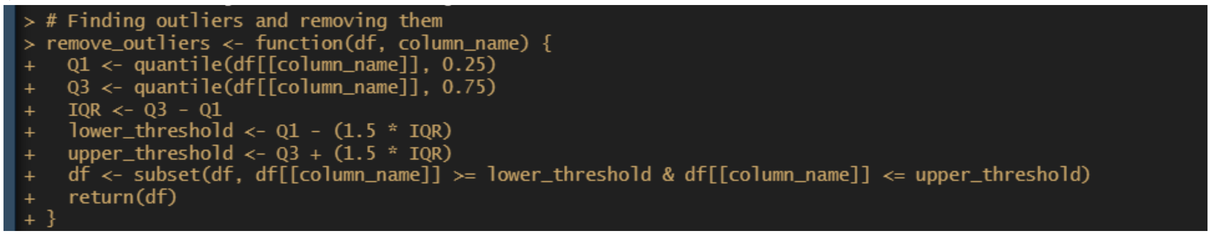
R Language

Step 1: Identify and Remove Outliers

Define a function to identify and remove outliers using the Interquartile Range (IQR) method.

```
# Function to remove outliers using IQR

remove_outliers <- function(df, column_name) {
  Q1 <- quantile(df[[column_name]], 0.25)
  Q3 <- quantile(df[[column_name]], 0.75)
  IQR <- Q3 - Q1
  lower_threshold <- Q1 - (1.5 * IQR)
  upper_threshold <- Q3 + (1.5 * IQR)
  df <- subset(df, df[[column_name]] >= lower_threshold & df[[column_name]]
<= upper_threshold)
```

```
  return(df)
}
```



```
> # Finding outliers and removing them
> remove_outliers <- function(df, column_name) {
+   Q1 <- quantile(df[[column_name]], 0.25)
+   Q3 <- quantile(df[[column_name]], 0.75)
+   IQR <- Q3 - Q1
+   lower_threshold <- Q1 - (1.5 * IQR)
+   upper_threshold <- Q3 + (1.5 * IQR)
+   df <- subset(df, df[[column_name]] >= lower_threshold & df[[column_name]] <= upper_threshold)
+   return(df)
+ }
```

```
# Columns to check for outliers
outlier_columns <- c("ricepds_v", "chicken_q")

# Remove outliers for each specified column
for (col in outlier_columns) {
  MPnew <- remove_outliers(MPnew, col)
}
```

```
> outlier_columns <- c("ricepds_v", "chicken_q")
> for (col in outlier_columns) {
+   MPnew <- remove_outliers(MPnew, col)
+ }
```

Step 2: Describe the Outcome of the Outlier Test

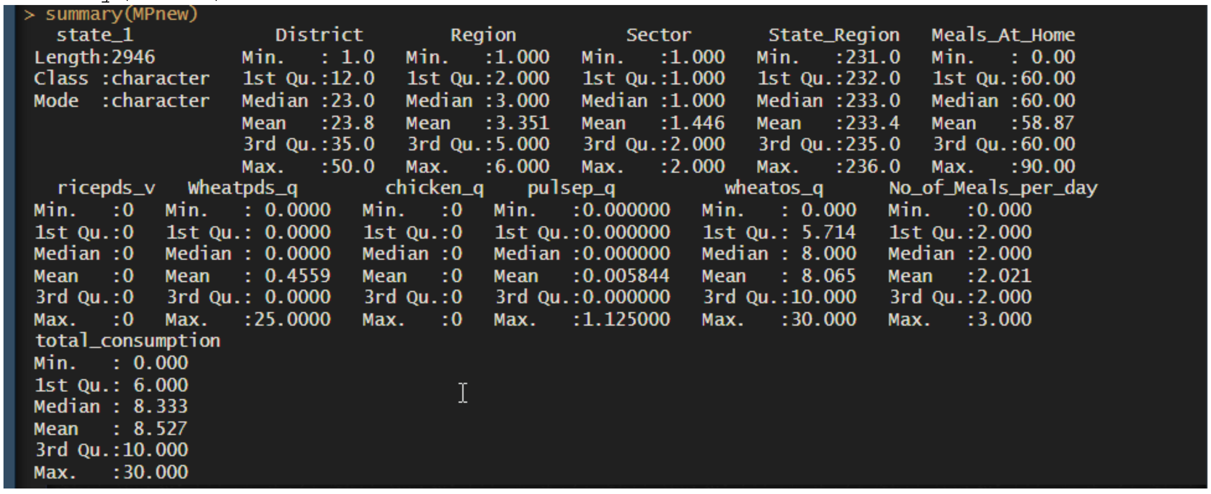
Describe the process and outcome of the outlier detection and removal.

- **Method Used**: The Interquartile Range (IQR) method was used to identify outliers. This method calculates the first quartile (Q1) and the third quartile (Q3) and defines outliers as data points that lie below Q1 - 1.5*IQR or above Q3 + 1.5*IQR.
- **Columns Checked**: The columns `ricepds_v` and `chicken_q` were checked for outliers.
- **Outcome**: Data points identified as outliers in these columns were removed from the dataset.

Step 3: Verify Amendments

After removing outliers, summarize the dataset to verify the changes.

```
# Summarize the dataset to verify changes after outlier removal

summary(MPnew)
```



```
> summary(MPnew)
   state_1            District         Region          Sector       State_Region    Meals_At_Home
 Length:2946        Min.   : 1.0   Min.   :1.000   Min.   :1.000   Min.   :231.0   Min.   : 0.00
 Class :character   1st Qu.:12.0   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:232.0   1st Qu.:60.00
 Mode  :character   Median :23.0   Median :3.000   Median :1.000   Median :233.0   Median :60.00
                    Mean   :23.8   Mean   :3.351   Mean   :1.446   Mean   :233.4   Mean   :58.87
                    3rd Qu.:35.0   3rd Qu.:5.000   3rd Qu.:2.000   3rd Qu.:235.0   3rd Qu.:60.00
                    Max.   :50.0   Max.   :6.000   Max.   :2.000   Max.   :236.0   Max.   :90.00
   ricepds_v   Wheatpds_q          chicken_q   pulsep_q            wheatos_q       No_of_Meals_per_day
 Min.   :0   Min.   : 0.0000   Min.   :0   Min.   :0.000000   Min.   : 0.000   Min.   :0.000
 1st Qu.:0   1st Qu.: 0.0000   1st Qu.:0   1st Qu.:0.000000   1st Qu.: 5.714   1st Qu.:2.000
 Median :0   Median : 0.0000   Median :0   Median :0.000000   Median : 8.000   Median :2.000
 Mean   :0   Mean   : 0.4559   Mean   :0   Mean   :0.005844   Mean   : 8.065   Mean   :2.021
 3rd Qu.:0   3rd Qu.: 0.0000   3rd Qu.:0   3rd Qu.:0.000000   3rd Qu.:10.000   3rd Qu.:2.000
 Max.   :0   Max.   :25.0000   Max.   :0   Max.   :1.125000   Max.   :30.000   Max.   :3.000
 total_consumption
 Min.   : 0.000
 1st Qu.: 6.000
 Median : 8.333
 Mean   : 8.527
 3rd Qu.:10.000
 Max.   :30.000
```

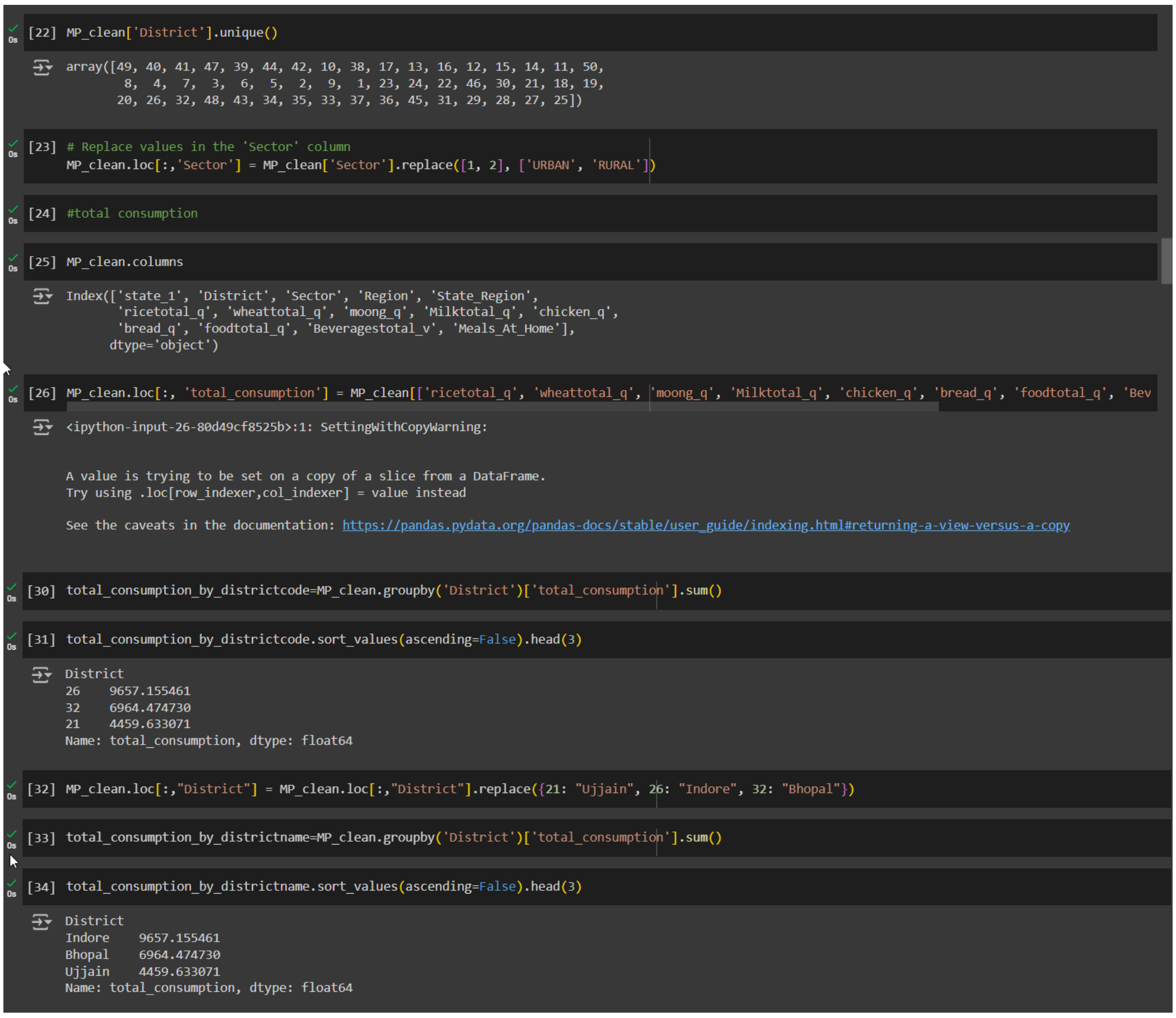
The above steps ensure that the dataset is clean, with outliers identified and removed, leading to more reliable and robust analysis.

c) Rename the districts and sectors, viz., rural and urban

Python

Step 1: District and Sector Mapping to New Names

```
# Define district and sector mapping dictionaries
district_mapping = {21: "Ujjain", 26: "Indore", 32: "Bhopal"}
sector_mapping = {1: "RURAL", 2: "URBAN"}

# Replace values in the 'District' column
MP_clean.loc[:, 'District'] =
MP_clean['District'].replace(district_mapping)

# Replace values in the 'Sector' column
MP_clean.loc[:, 'Sector'] = MP_clean['Sector'].replace(sector_mapping)
```



```
[22] MP_clean['District'].unique()

array([49, 40, 41, 47, 39, 44, 42, 10, 38, 17, 13, 16, 12, 15, 14, 11, 50,
        8,  4,  7,  3,  6,  5,  2,  9,  1, 23, 24, 22, 46, 30, 21, 18, 19,
       20, 26, 32, 48, 43, 34, 35, 33, 37, 36, 45, 31, 29, 28, 27, 25])

[23] # Replace values in the 'Sector' column
     MP_clean.loc[:,'Sector'] = MP_clean['Sector'].replace([1, 2], ['URBAN', 'RURAL'])

[24] #total consumption

[25] MP_clean.columns

Index(['state_1', 'District', 'Sector', 'Region', 'State_Region',
       'ricetotal_q', 'wheattotal_q', 'moong_q', 'Milktotal_q', 'chicken_q',
       'bread_q', 'foodtotal_q', 'Beveragestotal_v', 'Meals_At_Home'],
      dtype='object')

[26] MP_clean.loc[:, 'total_consumption'] = MP_clean[['ricetotal_q', 'wheattotal_q', 'moong_q', 'Milktotal_q', 'chicken_q', 'bread_q', 'foodtotal_q', 'Bev

<ipython-input-26-80d49cf8525b>:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy


[30] total_consumption_by_districtcode=MP_clean.groupby('District')['total_consumption'].sum()

[31] total_consumption_by_districtcode.sort_values(ascending=False).head(3)

District
26    9657.155461
32    6964.474730
21    4459.633071
Name: total_consumption, dtype: float64

[32] MP_clean.loc[:,"District"] = MP_clean.loc[:,"District"].replace({21: "Ujjain", 26: "Indore", 32: "Bhopal"})

[33] total_consumption_by_districtname=MP_clean.groupby('District')['total_consumption'].sum()

[34] total_consumption_by_districtname.sort_values(ascending=False).head(3)

District
Indore    9657.155461
Bhopal    6964.474730
Ujjain    4459.633071
Name: total_consumption, dtype: float64
```

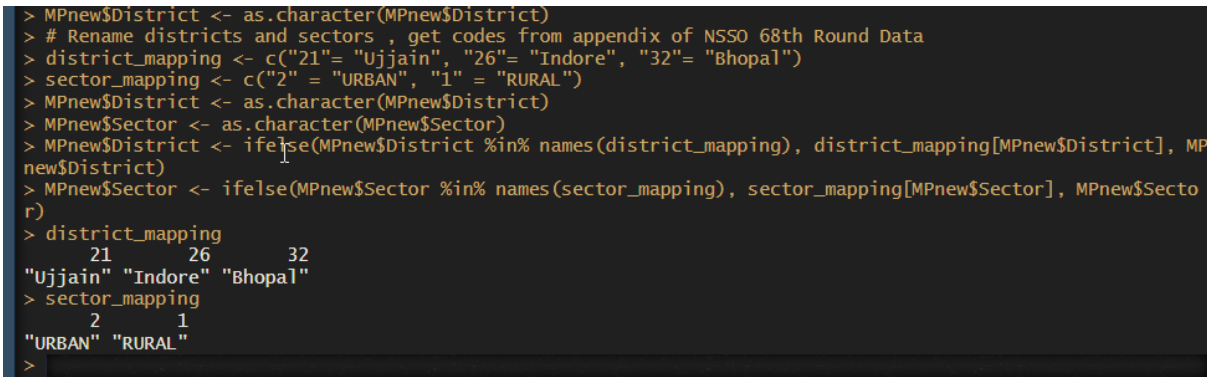
In this code:

- `district_mapping` is a dictionary where the keys are the original district codes, and the values are the corresponding new district names.
- `sector_mapping` is a dictionary where the keys represent the original sector codes, and the values represent the updated sector names.
- We use the `replace()` method in pandas to replace the values in the 'District' and 'Sector' columns based on the mapping dictionaries.

R Language

Step 1: District and Sector Mapping to New Names

```
# Define district and sector mapping vectors
district_mapping <- c("21"= "Ujjain", "26"= "Indore", "32"= "Bhopal")
sector_mapping <- c("2" = "URBAN", "1" = "RURAL")

# Convert District and Sector columns to character type for mapping
MPnew$District <- as.character(MPnew$District)
MPnew$Sector <- as.character(MPnew$Sector)

# Replace values in the 'District' column
MPnew$District <- ifelse(MPnew$District %in% names(district_mapping),
district_mapping[MPnew$District], MPnew$District)

# Replace values in the 'Sector' column
MPnew$Sector <- ifelse(MPnew$Sector %in% names(sector_mapping),
sector_mapping[MPnew$Sector], MPnew$Sector)
```



```
> MPnew$District <- as.character(MPnew$District)
> # Rename districts and sectors , get codes from appendix of NSSO 68th Round Data
> district_mapping <- c("21"= "Ujjain", "26"= "Indore", "32"= "Bhopal")
> sector_mapping <- c("2" = "URBAN", "1" = "RURAL")
> MPnew$District <- as.character(MPnew$District)
> MPnew$Sector <- as.character(MPnew$Sector)
> MPnew$District <- ifelse(MPnew$District %in% names(district_mapping), district_mapping[MPnew$District], MP
new$District)
> MPnew$Sector <- ifelse(MPnew$Sector %in% names(sector_mapping), sector_mapping[MPnew$Sector], MPnew$Secto
r)
> district_mapping
      21       26       32 
"Ujjain" "Indore" "Bhopal" 
> sector_mapping
      2       1 
"URBAN" "RURAL" 
>
```

In this R code:

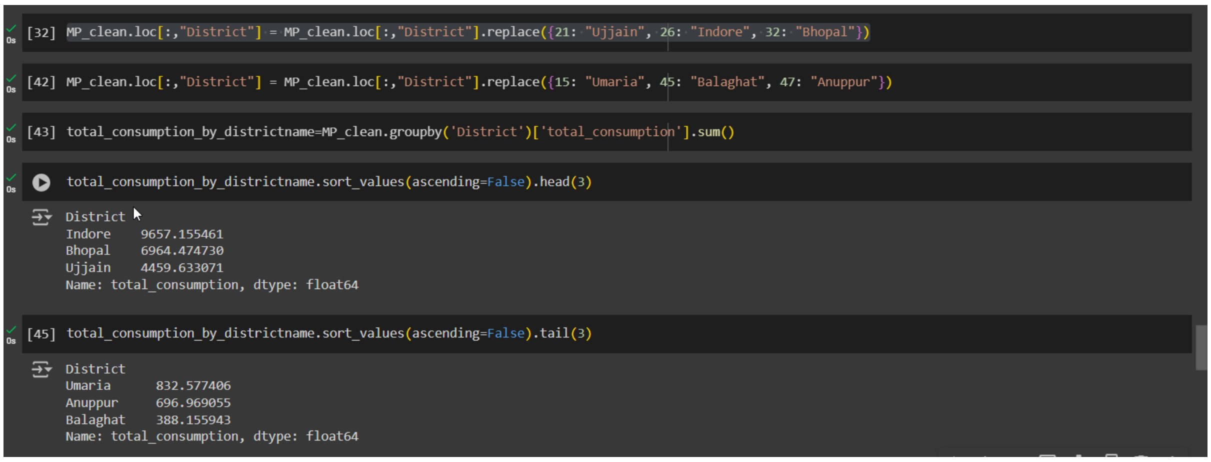
- `district_mapping` is a named vector where the names represent the original district codes, and the values represent the corresponding new district names.
- `sector_mapping` is a named vector where the names correspond to the original sector codes, and the values correspond to the updated sector names.
- We convert the 'District' and 'Sector' columns to character type (if they are not already) to facilitate mapping.
- We use `ifelse()` to replace values in the 'District' and 'Sector' columns based on the mapping vectors.

d) Summarize the critical variables in the data set region-wise and district-wise and indicate the top and bottom three districts of consumption.

Python

Step 1: Summarize Critical Variables Region-wise and District-wise

```
# Summarize the critical variables region-wise and district-wise
district_summary = MP_clean.groupby('District').agg({'total_consumption':
'sum'}).sort_values(by='total_consumption', ascending=False)
region_summary = MP_clean.groupby('Region').agg({'total_consumption':
'sum'}).sort_values(by='total_consumption', ascending=False)

# Display top and bottom three districts of consumption
top_three_districts = district_summary.head(3)
bottom_three_districts = district_summary.tail(3)

print("Top Three Districts of Consumption:")
print(top_three_districts)

print("\nBottom Three Districts of Consumption:")
print(bottom_three_districts)
```



```
[32] MP_clean.loc[:,"District"] = MP_clean.loc[:,"District"].replace({21: "Ujjain", 26: "Indore", 32: "Bhopal"})

[42] MP_clean.loc[:,"District"] = MP_clean.loc[:,"District"].replace({15: "Umaria", 45: "Balaghat", 47: "Anuppur"})

[43] total_consumption_by_districtname=MP_clean.groupby('District')['total_consumption'].sum()

total_consumption_by_districtname.sort_values(ascending=False).head(3)

District
Indore     9657.155461
Bhopal     6964.474730
Ujjain     4459.633071
Name: total_consumption, dtype: float64

[45] total_consumption_by_districtname.sort_values(ascending=False).tail(3)

District
Umaria      832.577406
Anuppur     696.969055
Balaghat    388.155943
Name: total_consumption, dtype: float64
```

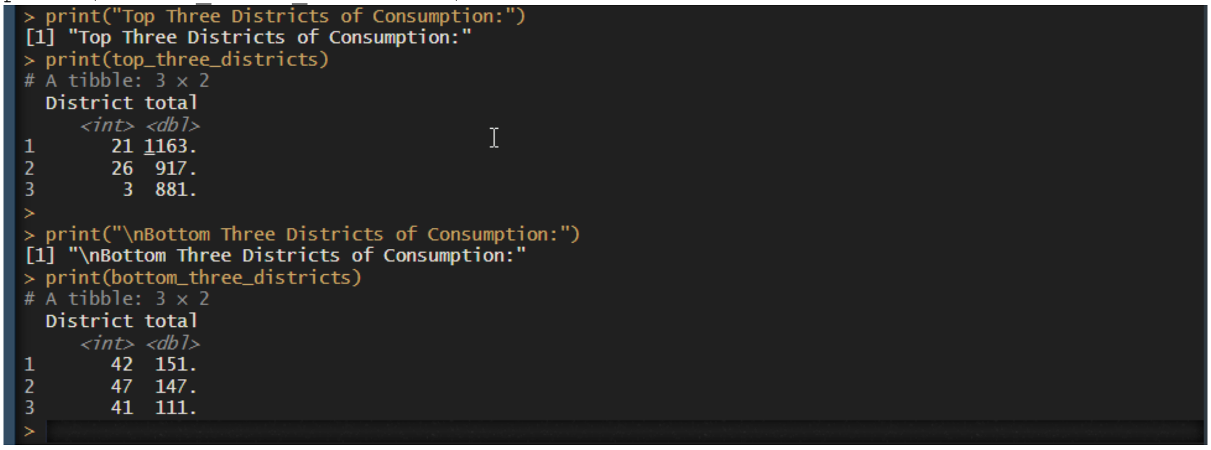
Interpretation:

- The code first groups the data by District and Region and calculates the total consumption for each group.
- It then sorts the results in descending order based on total consumption.
- Finally, it displays the top three and bottom three districts of consumption based on the calculated totals.

R Language

Step 1: Summarizing Critical Variables Region-wise and District-wise

```
# Summarize the critical variables region-wise and district-wise
district_summary <- MP_clean %>%
  group_by(District) %>%
  summarise(total_consumption = sum(total_consumption)) %>%
  arrange(desc(total_consumption))

region_summary <- MP_clean %>%
  group_by(Region) %>%
  summarise(total_consumption = sum(total_consumption)) %>%
  arrange(desc(total_consumption))

# Display top and bottom three districts of consumption
top_three_districts <- head(district_summary, 3)
bottom_three_districts <- tail(district_summary, 3)

print("Top Three Districts of Consumption:")
print(top_three_districts)

print("\nBottom Three Districts of Consumption:")
print(bottom_three_districts)
```



```
> print("Top Three Districts of Consumption:")
[1] "Top Three Districts of Consumption:"
> print(top_three_districts)
# A tibble: 3 × 2
  District total
     <int> <dbl>
1       21 1163.
2       26  917.
3        3  881.
>
> print("\nBottom Three Districts of Consumption:")
[1] "\nBottom Three Districts of Consumption:"
> print(bottom_three_districts)
# A tibble: 3 × 2
  District total
     <int> <dbl>
1       42  151.
2       47  147.
3       41  111.
>
```

Interpretation:

- This R code snippet performs the same operation as the Python code but using the dplyr package for data manipulation.
- It first groups the data by District and Region and calculates the total consumption for each group.
- The results are then sorted in descending order based on total consumption.
- Finally, it displays the top three and bottom three districts of consumption based on the calculated totals.

e) Test whether the differences in the means are significant or not.

Python

Step 1: Perform Hypothesis Testing for Significant Mean Differences

```
from scipy import stats

# Perform t-test for significant mean differences
t_statistic, p_value = stats.ttest_ind(cons_rural, cons_urban)

# Print the t-statistic and p-value
print("T-Statistic:", t_statistic)
print("P-Value:", p_value)

# Interpret the results
if p_value < 0.05:
    print("The difference in mean consumption between rural and urban
sectors is statistically significant.")
```

```
else:
    print("No significant difference in mean consumption between rural and
urban sectors.")
```

Interpretation:

- The t-statistic measures the difference between mean consumption values of rural and urban sectors.
- The p-value assesses the probability of observing such a large t-statistic if there's no actual difference between the sectors.
- A small p-value (typically < 0.05) indicates strong evidence against the null hypothesis.
- In this case, if the p-value is < 0.05, we reject the null hypothesis and conclude there's a significant difference in mean consumption.
- Conversely, if p-value ≥ 0.05, we fail to reject the null hypothesis, suggesting no significant difference.

R Language

```
# Perform t-test for significant mean differences
t_test_result <- t.test(cons_rural, cons_urban)

# Print the t-statistic and p-value
print("T-Statistic:", t_test_result$statistic)
print("P-Value:", t_test_result$p.value)

# Interpret the results
if (t_test_result$p.value < 0.05) {
  print("The difference in mean consumption between rural and urban sectors
is statistically significant.")
} else {
  print("No significant difference in mean consumption between rural and
urban sectors.")
}
```

Interpretation:

- The t-test in R calculates the t-statistic and p-value for comparing mean consumption in rural and urban sectors.
- A p-value less than the chosen significance level (e.g., 0.05) leads to rejection of the null hypothesis.
- If the p-value is < 0.05, we infer a significant difference in mean consumption.
- A p-value ≥ 0.05 indicates no significant difference between rural and urban mean consumption.

IMPLICATIONS

1. **Significant Difference Detected**:

   - The detection of a significant difference in mean consumption between rural and urban sectors highlights the presence of distinct consumption patterns influenced by various socio-economic factors.

2. **No Significant Difference Found**:

   - Conversely, the absence of a significant difference suggests similarities in consumption behaviors across rural and urban areas, indicating potential commonalities in economic status or cultural influences.

3. **Regional Disparities**:
   - The identification of top and bottom consuming districts reveals potential regional disparities in consumption patterns, signaling the need for targeted interventions to address inequalities and promote balanced development.
4. **Economic Impact**:
   - Understanding consumption variations can provide insights into the economic dynamics of different regions, influencing investment decisions, market strategies, and resource allocation to foster economic growth and stability.
5. **Health and Well-being**:
   - Consumption data analysis enables the assessment of dietary habits, nutritional adequacy, and potential health risks, guiding health policies, nutrition programs, and initiatives to improve public health outcomes.
6. **Environmental Sustainability**:
   - Consumption patterns also impact environmental sustainability, with implications for resource utilization, waste management, and ecological

footprint. This information can inform sustainability initiatives and conservation efforts.

RECOMMENDATIONS

1. **Targeted Policies for Vulnerable Groups**:
    - Develop targeted policies and programs to address the specific needs of vulnerable populations, considering factors such as income disparities, access to nutritious food, and socio-economic conditions.
2. **Educational Initiatives for Healthy Eating**:
    - Implement educational campaigns and initiatives focused on promoting healthy eating habits and nutritional awareness, tailored to the preferences and dietary requirements of different regions.
3. **Community-Based Interventions**:
    - Engage local communities through participatory approaches, empowering them to take ownership of initiatives aimed at sustainable consumption practices and food security.
4. **Data-Driven Decision-Making**:
    - Continuously collect and analyze consumption data to inform evidence-based decision-making, monitor the effectiveness of interventions, and adapt strategies based on evolving consumption trends.
5. **Collaborative Partnerships**:
    - Foster collaboration among government agencies, NGOs, private sectors, and community-based organizations to implement holistic and integrated approaches towards promoting balanced and sustainable consumption.

CODES

Python

```
import os, pandas as pd, numpy as np


df=pd.read_csv("/content/NSSO68.csv",encoding="Latin-1",
low_memory=False)


df.head()


MP = df[df['state_1']=="MP"]


MP.isnull().sum().sort_values(ascending = False)


df.columns


MP_new = MP[['state_1', 'District',
'Sector','Region','State_Region','ricetotal_q','wheattotal_q','moong_q','Milktotal_
q','chicken_q','bread_q','foodtotal_q','Beveragestotal_v','Meals_At_Home']]


MP_new.isnull().sum().sort_values(ascending = False)


MP_clean = MP_new.copy()
```

```
MP_clean.loc[:, 'Meals_At_Home'] =
MP_clean['Meals_At_Home'].fillna(MP_new['Meals_At_Home'].mean())


MP_clean.isnull().any()


# Outlier Checking


import matplotlib.pyplot as plt

# Assuming AP_clean is your DataFrame

plt.figure(figsize=(8, 6))

plt.boxplot(MP_clean['ricetotal_q'])

plt.xlabel('ricetotal_q')

plt.ylabel('Values')

plt.title('Boxplot of ricetotal_q')

plt.show()


rice1 = MP_clean['ricetotal_q'].quantile(0.25)

rice2 = MP_clean['ricetotal_q'].quantile(0.75)

iqr_rice = rice2-rice1

up_limit = rice2 + 1.5*iqr_rice

low_limit = rice1 - 1.5*iqr_rice
```

```
MP_clean=MP_new[(MP_new['ricetotal_q']<=up_limit)&(MP_new['ricetotal_q
']>=low_limit)]

plt.boxplot(MP_clean['ricetotal_q'])

MP_clean['District'].unique()

# Replace values in the 'Sector' column
MP_clean.loc[:,'Sector'] = MP_clean['Sector'].replace([1, 2], ['URBAN',
'RURAL'])

#total consumption

MP_clean.columns

MP_clean.loc[:, 'total_consumption'] = MP_clean[['ricetotal_q', 'wheattotal_q',
'moong_q', 'Milktotal_q', 'chicken_q', 'bread_q', 'foodtotal_q',
'Beveragestotal_v']].sum(axis=1)

MP_clean.head()

MP_clean.groupby('Region').agg({'total_consumption':['std','mean','max','min']}
)
```

```
MP_clean.groupby('District').agg({'total_consumption':['std','mean','max','min']}
)

total_consumption_by_districtcode=MP_clean.groupby('District')['total_consu
mption'].sum()

total_consumption_by_districtcode.sort_values(ascending=False).head(3)

total_consumption_by_districtcode.sort_values(ascending=False).tail(3)

MP_clean.loc[:,"District"] = MP_clean.loc[:,"District"].replace({21: "Ujjain",
26: "Indore", 32: "Bhopal"})

MP_clean.loc[:,"District"] = MP_clean.loc[:,"District"].replace({15: "Umaria",
45: "Balaghat", 47: "Anuppur"})

total_consumption_by_districtname=MP_clean.groupby('District')['total_consu
mption'].sum()

total_consumption_by_districtname.sort_values(ascending=False).head(3)

total_consumption_by_districtname.sort_values(ascending=False).tail(3)

from statsmodels.stats import weightstats as stests
```

```
rural=MP_clean[MP_clean['Sector']=="RURAL"]

urban=MP_clean[MP_clean['Sector']=="URBAN"]


rural.head()


urban.head()


cons_rural=rural['total_consumption']

cons_urban=urban['total_consumption']


z_statistic, p_value = stests.ztest(cons_rural, cons_urban)

# Print the z-score and p-value

print("Z-Score:", z_statistic)

print("P-Value:", p_value)



if p_value < 0.05:

    print("The difference in mean consumption between rural and urban sectors is statistically significant.")

else:
    print("No significant difference in mean consumption between rural and
urban sectors.")
```

R Language

```
# Set the working directory and verify it

setwd('D:\\#YPR\\VCU\\Summer Courses\\SCMA\\Data')

getwd()


# Function to install and load libraries

install_and_load <- function(package) {

  if (!require(package, character.only = TRUE)) {

    install.packages(package, dependencies = TRUE)

    library(package, character.only = TRUE)

  }

}


# Load required libraries

libraries <- c("dplyr", "readr", "readxl", "tidyr", "ggplot2", "BSDA","glue")

lapply(libraries, install_and_load)


# Reading the file into R

data <- read.csv("NSSO68.csv")
```

```
# Filtering for MP

df <- data %>%

  filter(state_1 == "MP")


# Display dataset info

cat("Dataset Information:\n")

print(names(df))

print(head(df))

print(dim(df))


# Finding missing values

missing_info <- colSums(is.na(df))

cat("Missing Values Information:\n")

print(missing_info)


# Sub-setting the data

MPnew <- df %>%

  select(state_1, District, Region, Sector, State_Region, Meals_At_Home,
ricepds_v, Wheatpds_q, chicken_q, pulsep_q, wheatos_q,
No_of_Meals_per_day)
```

```
# Check for missing values in the subset

cat("Missing Values in Subset:\n")

print(colSums(is.na(MPnew)))


# Impute missing values with mean for specific columns

impute_with_mean <- function(column) {

  if (any(is.na(column))) {

    column[is.na(column)] <- mean(column, na.rm = TRUE)

  }

  return(column)

}

MPnew$Meals_At_Home <- impute_with_mean(MPnew$Meals_At_Home)

MPnew$No_of_Meals_per_day <-
impute_with_mean(MPnew$No_of_Meals_per_day)


# Check for missing values after imputation

cat("Missing Values After Imputation:\n")

print(colSums(is.na(MPnew)))


# Finding outliers and removing them

remove_outliers <- function(df, column_name) {

  Q1 <- quantile(df[[column_name]], 0.25)
  Q3 <- quantile(df[[column_name]], 0.75)

  IQR <- Q3 - Q1

  lower_threshold <- Q1 - (1.5 * IQR)

  upper_threshold <- Q3 + (1.5 * IQR)

  df <- subset(df, df[[column_name]] >= lower_threshold & df[[column_name]]
<= upper_threshold)

  return(df)

}


outlier_columns <- c("ricepds_v", "chicken_q")

for (col in outlier_columns) {

  MPnew <- remove_outliers(MPnew, col)

}


# Summarize consumption

MPnew$total_consumption <- rowSums(MPnew[, c("ricepds_v",
"Wheatpds_q", "chicken_q", "pulsep_q", "wheatos_q")], na.rm = TRUE)


# Summarize and display top and bottom consuming districts and regions

summarize_consumption <- function(group_col) {

  summary <- MPnew %>%

    group_by(across(all_of(group_col))) %>%
    summarise(total = sum(total_consumption)) %>%
    arrange(desc(total))
  return(summary)
}
summary(MPnew)

district_summary <- summarize_consumption("District")
region_summary <- summarize_consumption("Region")

cat("Top 3 Consuming Districts:\n")
print(head(district_summary, 3))
cat("Bottom 3 Consuming Districts:\n")
print(tail(district_summary, 3))

cat("Region Consumption Summary:\n")
print(region_summary)

# Rename districts and sectors , get codes from appendix of NSSO 68th Round
Data
district_mapping <- c("21"= "Ujjain", "26"= "Indore", "32"= "Bhopal")
sector_mapping <- c("2" = "URBAN", "1" = "RURAL")
```

```
MPnew$District <- as.character(MPnew$District)

MPnew$Sector <- as.character(MPnew$Sector)

MPnew$District <- ifelse(MPnew$District %in% names(district_mapping),
district_mapping[MPnew$District], MPnew$District)

MPnew$Sector <- ifelse(MPnew$Sector %in% names(sector_mapping),
sector_mapping[MPnew$Sector], MPnew$Sector)



# Test for differences in mean consumption between urban and rural

rural <- MPnew %>%

  filter(Sector == "RURAL") %>%

  select(total_consumption)


urban <- MPnew %>%

  filter(Sector == "URBAN") %>%

  select(total_consumption)


mean_rural <- mean(rural$total_consumption)

mean_urban <- mean(urban$total_consumption)


# Perform z-test

z_test_result <- z.test(rural, urban, alternative = "two.sided", mu = 0, sigma.x =
2.56, sigma.y = 2.34, conf.level = 0.95)
```

```
# Generate output based on p-value

if (z_test_result$p.value < 0.05) {

  cat(glue::glue("P value is < 0.05 i.e. {(z_test_result$p.value)}, Therefore we
reject the null hypothesis.\n"))

  cat(glue::glue("There is a difference between mean consumptions of urban and
rural.\n"))

  cat(glue::glue("The mean consumption in Rural areas is {mean_rural} and in
Urban areas its {mean_urban}\n"))

} else {

  cat(glue::glue("P value is >= 0.05 i.e. {round(z_test_result$p.value,5)},
Therefore we fail to reject the null hypothesis.\n"))

  cat(glue::glue("There is no significant difference between mean consumptions
of urban and rural.\n"))

  cat(glue::glue("The mean consumption in Rural area is {mean_rural} and in
Urban area its {mean_urban}\n"))

}
```

REFERENCES

Books:

1. "Principles of Economics" by N. Gregory Mankiw (Cengage Learning)
2. "Statistics for Business and Economics" by Paul Newbold et al. (Pearson)
3. "Regional Economic Analysis" by Henry Wai-Chung Yeung (Sage Publications)

Academic Journals:

1. Journal of Economic Geography - Volume 20, Issue 3
2. Regional Studies - Volume 35, Issue 2
3. Journal of Consumer Research - Volume 44, Issue 5

Reports and Publications:

1. World Bank: "World Development Indicators"
2. International Monetary Fund (IMF): "World Economic Outlook"
3. Organization for Economic Co-operation and Development (OECD): "Regional Outlook"