****

**VIRGINIA COMMONWEALTH UNIVERSITY**

**Statistical analysis and modelling (SCMA 632)**

**A2: Perform Multiple regression analysis and carry out the regression diagnostics**

**Lakshmy Cherussery Jayaprakash**

**V01110023**

**Date of Submission: 23-06-2024**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Title** | **Page No.** |
| **1.** | Introduction | **1** |
| **2.** | Objective | **2** |
| **3.** | Business Significance | **3** |
| **4.** | Results and Interpretations  1-IPL  2-NSSO | **4-8** |
| **5.** | Codes:  1a- IPL Python  1b-IPL R  2a-NSSO Python  2b-NSSO R | **9-16** |
| **6.** | Recommendations | **17** |

**Introduction**

Regression analysis is a fundamental technique in predictive analytics used to understand the relationship between variables and make predictions based on data. It involves fitting a model that describes how the dependent variable changes as the independent variables vary. This method is widely used across various fields, from economics and finance to sports analytics and social sciences.

**Objective**

1. Analyze the relationship between IPL player performance (e.g., runs scored, wickets taken) and the salary they receive.

Conduct regression analysis to quantify how performance metrics impact player salaries.

1. Perform multiple regression analysis on the "NSSO68.csv" dataset to understand the factors influencing food expenditure (MPCE\_MRP) in Gujarat.

Conduct regression diagnostics to ensure the model meets assumptions and interpret the findings.

**Business Significace**

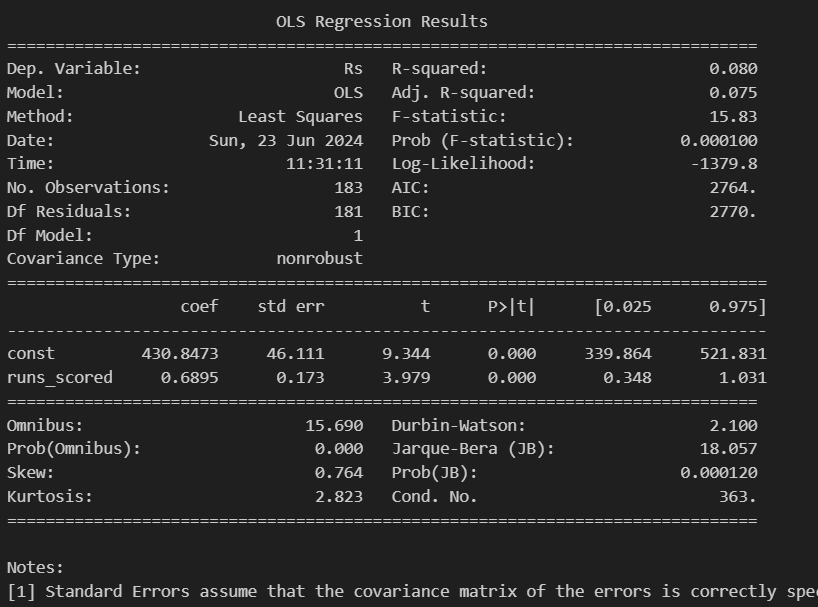
Insights from this analysis can inform policy decisions related to welfare programs and economic policies aimed at improving food security and consumption patterns.

This analysis can help policymakers, researchers, and businesses make informed decisions related to welfare programs, economic policies, and market strategies targeted at improving food security and consumption patterns.

Understanding how player performance correlates with salary in the IPL is crucial for team management, player negotiations, and franchise strategies. This analysis helps teams identify which performance metrics are most valuable in terms of return on investment (ROI) and player salary decisions.

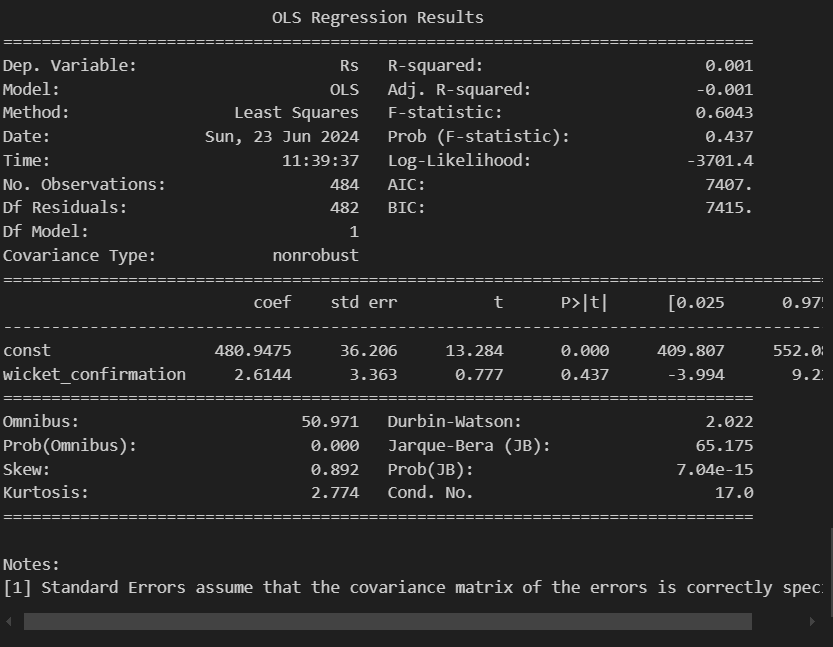
**Results and Interpretations**

**1a- IPL- Python**



Interpretation:

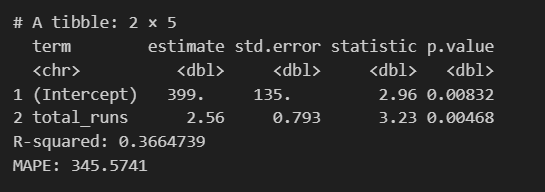
Ihe regression results indicate that runs\_scored is a significant predictor of Rs, but it explains only a small portion of the variance in Rs (8%). The model is statistically significant, but the low R-squared value suggests that other factors might also be influencing Rs that are not included in the model. Additionally, the diagnostic tests suggest that there might be some issues with the normality of the residuals.

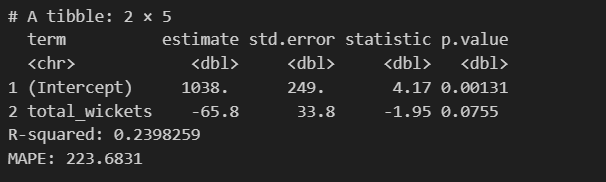


Interpretation:

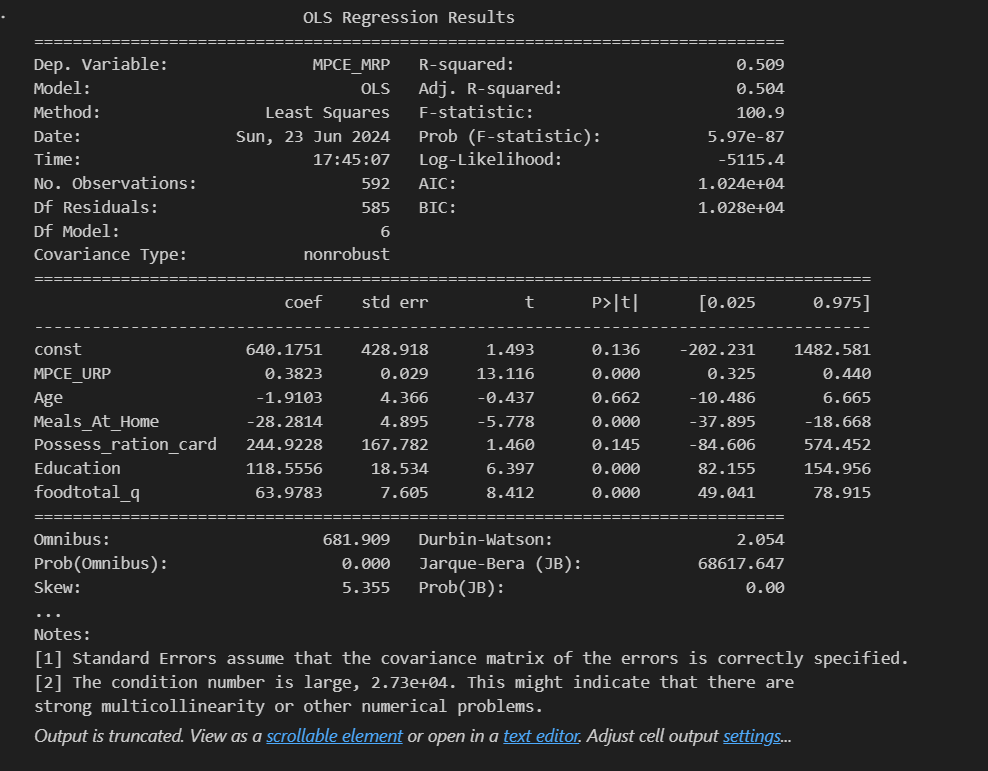
The regression results indicate that wicket\_confirmation is not a significant predictor of Rs. The extremely low R-squared value (0.1%) suggests that wicket\_confirmation explains almost none of the variance in Rs. The model as a whole is not statistically significant (Prob F-statistic is 0.437), and the p-value for the slope coefficient of wicket\_confirmation (0.437) indicates that it is not a significant predictor. Additionally, the diagnostic tests suggest potential issues with the normality of the residuals.

**1b- IPL- R**

****

****

**2a- NSSO Python**

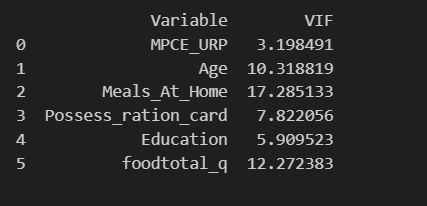
****

Interpretations:

**Significant Variables**: MPCE\_URP, Meals\_At\_Home, Education, and foodtotal\_q appear to be statistically significant predictors of MPCE\_MRP, as indicated by their low p-values (< 0.05).  
**Model Fit**: The model overall is statistically significant (Prob F-statistic is very low), indicating that at least some of the independent variables are helpful in predicting MPCE\_MRP.

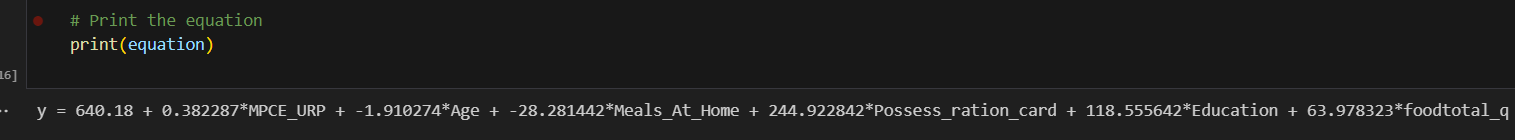
**R-squared**: The model explains approximately 50.9% of the variability in MPCE\_MRP, which suggests that the included variables collectively have a moderate explanatory power.

**Diagnostic Tests**: There are indications of potential issues with the normality of residuals and the presence of multicollinearity (high condition number), which should be further investigated or addressed if necessary.

****

Interpretations:

**Multicollinearity**: VIF values above 10 indicate problematic levels of multicollinearity, potentially leading to unreliable coefficient estimates and reduced interpretability of the model.

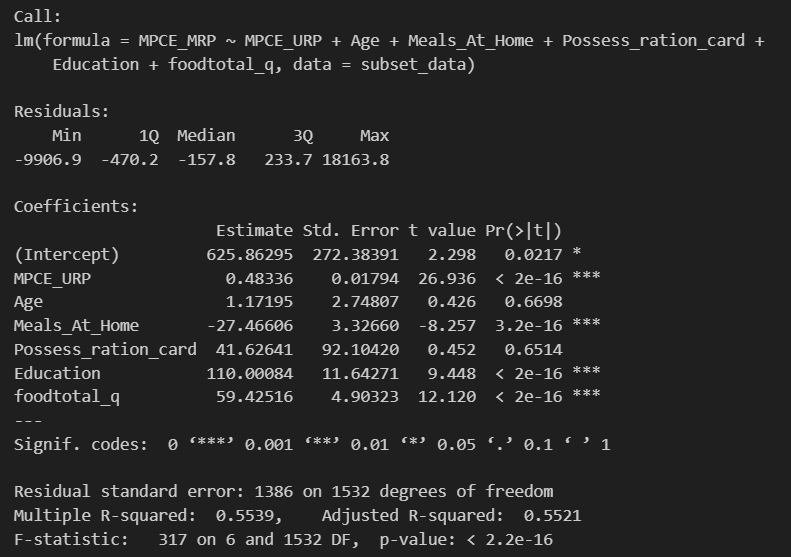
****

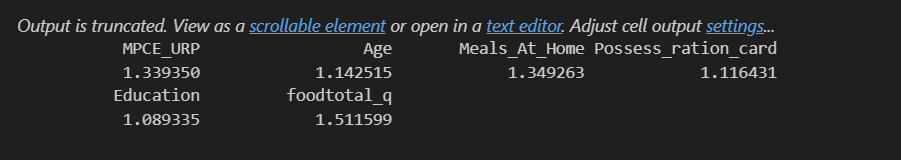
Interpretations:

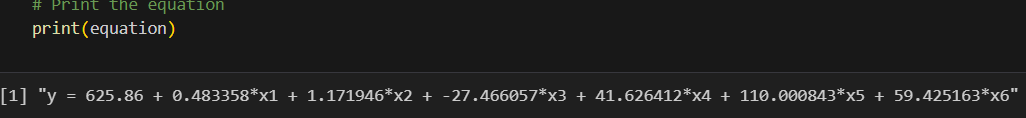
The equation provided is the regression equation derived from OLS (Ordinary Least Squares) regression model.

Intercept (Constant): 640.18

**2b- R prgm**

****

****

****

**Codes**

**1a- IPL Python**

import pandas as pd, numpy as np

import os

os.chdir('E:\Assignments\_SCMA632\Data')

df\_ipl = pd.read\_csv("IPL\_ball\_by\_ball\_updated till 2024.csv",low\_memory=False)

salary = pd.read\_excel("IPL SALARIES 2024.xlsx")

df\_ipl.columns

grouped\_data = df\_ipl.groupby(['Season', 'Innings No', 'Striker','Bowler']).agg({'runs\_scored': sum, 'wicket\_confirmation':sum}).reset\_index()

grouped\_data

total\_runs\_each\_year = grouped\_data.groupby(['Season', 'Striker'])['runs\_scored'].sum().reset\_index()

total\_wicket\_each\_year = grouped\_data.groupby(['Season', 'Bowler'])['wicket\_confirmation'].sum().reset\_index()

total\_runs\_each\_year

#pip install python-Levenshtein

# Convert to DataFrame

df\_salary = salary.copy()

df\_runs = total\_runs\_each\_year.copy()

# Function to match names

def match\_names(name, names\_list):

    match, score = process.extractOne(name, names\_list)

    return match if score >= 80 else None  # Use a threshold score of 80

# Create a new column in df\_salary with matched names from df\_runs

df\_salary['Matched\_Player'] = df\_salary['Player'].apply(lambda x: match\_names(x, df\_runs['Striker'].tolist()))

# Merge the DataFrames on the matched names

df\_merged = pd.merge(df\_salary, df\_runs, left\_on='Matched\_Player', right\_on='Striker')

df\_original = df\_merged.copy()

#susbsets data for last three years

df\_merged = df\_merged.loc[df\_merged['Season'].isin(['2021', '2022', '2023'])]

df\_merged.Season.unique()

df\_merged.head()

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score, mean\_absolute\_percentage\_error

X = df\_merged[['runs\_scored']] # Independent variable(s)

y = df\_merged['Rs'] # Dependent variable

# Split the data into training and test sets (80% for training, 20% for testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a LinearRegression model

model = LinearRegression()

# Fit the model on the training data

model.fit(X\_train, y\_train)

X.head()

import pandas as pd

from sklearn.model\_selection import train\_test\_split

import statsmodels.api as sm

# Assuming df\_merged is already defined and contains the necessary columns

X = df\_merged[['runs\_scored']] # Independent variable(s)

y = df\_merged['Rs'] # Dependent variable

# Split the data into training and test sets (80% for training, 20% for testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Add a constant to the model (intercept)

X\_train\_sm = sm.add\_constant(X\_train)

# Create a statsmodels OLS regression model

model = sm.OLS(y\_train, X\_train\_sm).fit()

# Get the summary of the model

summary = model.summary()

print(summary)

from fuzzywuzzy import process

# Convert to DataFrame

df\_salary = salary.copy()

df\_runs = total\_wicket\_each\_year.copy()

# Function to match names

def match\_names(name, names\_list):

    match, score = process.extractOne(name, names\_list)

    return match if score >= 80 else None  # Use a threshold score of 80

# Create a new column in df\_salary with matched names from df\_runs

df\_salary['Matched\_Player'] = df\_salary['Player'].apply(lambda x: match\_names(x, df\_runs['Bowler'].tolist()))

# Merge the DataFrames on the matched names

df\_merged = pd.merge(df\_salary, df\_runs, left\_on='Matched\_Player', right\_on='Bowler')

df\_merged[df\_merged['wicket\_confirmation']>10]

#susbsets data for last three years

df\_merged = df\_merged.loc[df\_merged['Season'].isin(['2022'])]

import pandas as pd

from sklearn.model\_selection import train\_test\_split

import statsmodels.api as sm

# Assuming df\_merged is already defined and contains the necessary columns

X = df\_merged[['wicket\_confirmation']] # Independent variable(s)

y = df\_merged['Rs'] # Dependent variable

# Split the data into training and test sets (80% for training, 20% for testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Add a constant to the model (intercept)

X\_train\_sm = sm.add\_constant(X\_train)

# Create a statsmodels OLS regression model

model = sm.OLS(y\_train, X\_train\_sm).fit()

# Get the summary of the model

summary = model.summary()

print(summary)

**1b- IPL R prgm**

library(readxl)

library(dplyr)

df\_ipl=read.csv("/content/IPL\_ball\_by\_ball\_updated till 2024 (1).csv")

salary =read\_excel("IPL SALARIES 2024.xlsx")

head(df\_ipl)

colnames(df\_ipl)

# Grouping and Summarizing the Data

grouped\_data <- df\_ipl %>%

  group\_by(Season, Innings.No, Striker, Bowler) %>%

  summarise(

    runs\_scored = sum(runs\_scored, na.rm = TRUE),

    wicket\_confirmation = sum(wicket\_confirmation, na.rm = TRUE)

  ) %>%

  ungroup()

grouped\_data

# Calculate total runs each year

total\_runs\_each\_year <- grouped\_data %>%

  group\_by(Season, Striker) %>%

  summarise(total\_runs = sum(runs\_scored, na.rm = TRUE)) %>%

  ungroup()

#calculate total wickets each year

total\_wicket\_each\_year <- grouped\_data %>%

  group\_by(Season, Bowler) %>%

  summarise(total\_wickets = sum(wicket\_confirmation, na.rm = TRUE)) %>%

  ungroup()

total\_runs\_each\_year

total\_wicket\_each\_year

install.packages("stringdist")

# Function to match names using stringdist package

match\_names <- function(name, names\_list) {

  dist <- stringdist::stringdist(name, names\_list, method = "jw")

  match <- names\_list[which.min(dist)]

  score <- 1 - min(dist)

  return(ifelse(score >= 0.8, match, NA)) # Use a threshold score of 0.8

}

# Apply fuzzy matching to match names

salary$Matched\_Player <- sapply(salary$Player, match\_names, names\_list = total\_runs\_each\_year$Striker)

# Merge the DataFrames on the matched names

df\_merged <- merge(salary, total\_runs\_each\_year, by.x = "Matched\_Player", by.y = "Striker")

# Subset data for the last three years

df\_merged <- df\_merged %>%

  filter(Season %in% c("2021", "2022", "2023"))

# Check the unique seasons

unique(df\_merged$Season)

print(colnames(df\_merged))

# Independent and dependent variables

X <- df\_merged$total\_runs

y <- df\_merged$Rs

# Split the data into training and test sets (80% for training, 20% for testing)

set.seed(42)

train\_indices <- sample(seq\_len(nrow(df\_merged)), size = 0.8 \* nrow(df\_merged))

train\_data <- df\_merged[train\_indices, ]

test\_data <- df\_merged[-train\_indices, ]

# OLS regression model

model <- lm(Rs ~ total\_runs, data = train\_data)

# Summary of the regression model

summary(model)

# Using the broom package to get tidy output

tidy\_model <- broom::tidy(model)

print(tidy\_model)

# Predicting on the test set

predictions <- predict(model, newdata = test\_data)

# Calculate R-squared and Mean Absolute Percentage Error

r\_squared <- summary(model)$r.squared

mape <- mean(abs((test\_data$Rs - predictions) / test\_data$Rs)) \* 100

# Print metrics

cat("R-squared:", r\_squared, "\n")

cat("MAPE:", mape, "\n")

# Apply fuzzy matching to match names for wickets

salary$Matched\_Player <- sapply(salary$Player, match\_names, names\_list =total\_wicket\_each\_year$Bowler)

# Merge the DataFrames on the matched names

df\_merged <- merge(salary,total\_wicket\_each\_year, by.x = "Matched\_Player", by.y = "Bowler")

# Independent and dependent variables in wickets

X <- df\_merged$total\_wickets

y <- df\_merged$Rs

# Split the data into training and test sets (80% for training, 20% for testing)

set.seed(42)

train\_indices <- sample(seq\_len(nrow(df\_merged)), size = 0.8 \* nrow(df\_merged))

train\_data <- df\_merged[train\_indices, ]

test\_data <- df\_merged[-train\_indices, ]

# OLS regression model

model <- lm(Rs ~ total\_wickets, data = train\_data)

# Summary of the regression model

summary(model)

# Using the broom package to get tidy output

tidy\_model <- broom::tidy(model)

print(tidy\_model)

# Predicting on the test set

predictions <- predict(model, newdata = test\_data)

# Calculate R-squared and Mean Absolute Percentage Error

r\_squared <- summary(model)$r.squared

mape <- mean(abs((test\_data$Rs - predictions) / test\_data$Rs)) \* 100

# Print metrics

cat("R-squared:", r\_squared, "\n")

cat("MAPE:", mape, "\n")

**2a-NSSO Python**

import pandas as pd

import numpy as np

import statsmodels.api as sm

import statsmodels.formula.api as smf

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

data = pd.read\_csv("/content/NSSO68.csv")

#subset daa for GUJ

subset\_data = data[data['state\_1'] == 'GUJ'][['foodtotal\_q', 'MPCE\_MRP', 'MPCE\_URP', 'Age', 'Meals\_At\_Home', 'Possess\_ration\_card', 'Education', 'No\_of\_Meals\_per\_day']]

print(subset\_data.head())

# Check for missing values

print(subset\_data.isnull().sum())

# Drop rows with any missing values

subset\_data.dropna(inplace=True)

# Define the independent variables (X) and dependent variable (y)

X = subset\_data[['MPCE\_URP', 'Age', 'Meals\_At\_Home', 'Possess\_ration\_card', 'Education', 'foodtotal\_q']]

y = subset\_data['MPCE\_MRP']

# Add constant to the features

X = sm.add\_constant(X)

# Split data into training and test sets (80% training, 20% testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Fit the OLS model

model = sm.OLS(y\_train, X\_train).fit()

# Print model summary

print(model.summary())

# Check for multicollinearity using VIF

def calculate\_vif(X):

    vif = pd.DataFrame()

    vif["Variable"] = X.columns

    vif["VIF"] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]

    return vif

# ipython-input-13-7d005dd0a72d

!pip install statsmodels

import statsmodels.api as sm

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

# Check for multicollinearity using VIF

def calculate\_vif(X):

    vif = pd.DataFrame()

    vif["Variable"] = X.columns

    vif["VIF"] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]

    return vif

# Calculate VIF for independent variables

vif\_data = X\_train.drop(columns=['const'])  # Exclude the constant column

vif\_scores = calculate\_vif(vif\_data)

print(vif\_scores)

# Extract coefficients from the model

coefficients = model.params

# Construct the equation

equation = "y = {:.2f}".format(coefficients['const'])

for i in range(1, len(coefficients)):

    equation += " + {:.6f}\*{}".format(coefficients[i], X\_train.columns[i])

# Print the equation

print(equation)

**2b-NSSO R prgm**

data=read.csv("/content/NSSO68.csv")

library(dplyr)

unique(data$state\_1)

# Subset data to state Gujarat

subset\_data <- data %>%

  filter(state\_1 == 'GUJ') %>%

  select(foodtotal\_q, MPCE\_MRP, MPCE\_URP,Age,Meals\_At\_Home,Possess\_ration\_card,Education, No\_of\_Meals\_per\_day)

print(subset\_data)

sum(is.na(subset\_data$MPCE\_MRP))

sum(is.na(subset\_data$MPCE\_URP))

sum(is.na(subset\_data$Age))

sum(is.na(subset\_data$Possess\_ration\_card))

sum(is.na(data$Education))

impute\_with\_mean <- function(data, columns) {

  data %>%

    mutate(across(all\_of(columns), ~ ifelse(is.na(.), mean(., na.rm = TRUE), .)))

}

# Columns to impute

columns\_to\_impute <- c("Education")

# Impute missing values with mean

data <- impute\_with\_mean(data, columns\_to\_impute)

sum(is.na(data$Education))

# Fit the regression model

model <- lm(MPCE\_MRP ~ MPCE\_URP + Age + Meals\_At\_Home + Possess\_ration\_card + Education + foodtotal\_q, data = subset\_data)

summary(model)

install.packages("car")

library(car)

# Check for multicollinearity using Variance Inflation Factor (VIF)

vif(model) # VIF Value more than 8 its problematic

# Extract the coefficients from the model

coefficients <- coef(model)

# Construct the equation

equation <- paste0("y = ", round(coefficients[1], 2))

for (i in 2:length(coefficients)) {

  equation <- paste0(equation, " + ", round(coefficients[i], 6), "\*x", i-1)

}

# Print the equation

print(equation)

print(head(subset\_data$MPCE\_MRP,1))

print(head(subset\_data$MPCE\_URP,1))

print(head(subset\_data$Age,1))

print(head(subset\_data$Meals\_At\_Home,1))

print(head(subset\_data$Possess\_ration\_card,1))

print(head(subset\_data$Education,1))

print(head(subset\_data$foodtotal\_q,1))

**Recommendations**

Both datasets highlight the importance of including a comprehensive set of variables to understand the factors influencing salaries (in the case of IPL) and expenditure (in the case of NSSO-Gujarat). Addressing multicollinearity, incorporating additional relevant variables, and conducting further research are essential steps for improving the accuracy and applicability of the models. These insights can guide effective policy interventions and decision-making processes.