VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A2: Multiple Regression Analysis for NSSO68

Linear Regression Analysis for IPL performance and salary

Chandhini Kerachan Muraleedharan

V01107497

Date of Submission: 23-06-2024

CONTENTS

Sl. No.	**Title**	**Page No.**
1.	Introduction	**1**
2.	Results	**2**
3.	Interpretations	**2**
4.	Recommendations	**9**
5.	Codes	**10**
6.	References	

1. Introduction

(i) About the data :

NSSO-Consumption Dataset: Contains numerical data on consumption patterns of various commodities (e.g., grains, oils, fruits) across Indian states and union territories, along with basic demographic information.

Ball by Ball Dataset: Provides detailed information on every ball bowled in IPL matches from 2008 to 2022, with 816 unique match IDs and 17 variables (numeric and text), including team names, runs scored, and player performance.

IPL Matches Dataset: Contains text-based information on IPL matches from 2008 to 2022, with 16 variables per match ID, detailing dates, cities, teams, toss results, and player details.

IPL Salary Dataset: Includes yearly salary information for IPL players, with columns for salaries in dollars and without the "$" symbol, enabling analysis of salary trends across teams and years.

(ii) Objectives

NSSO Dataset: Multiple Regression Analysis

1. Perform multiple regression analysis on the NSSO68 dataset.
2. Conduct regression diagnostics to assess model assumptions and identify problems.
3. Interpret and explain the findings.
4. Address identified issues and re-evaluate the results.
5. Discuss significant changes observed after resolving issues.

IPL Data: Player Performance and Salary Analysis

1. Investigate the relationship between player performance and salary in the IPL through linear regression.
2. Conduct correlation analysis to explore the relationship between performance factors and pay.
3. Interpret the correlation results and discuss findings.
4. Provide insights into factors influencing player compensation, identify top performers and underperformers, and analyze statistical distributions for key players.

(iii) Business Significance

Multiple Regression Analysis and Regression Diagnostics provide valuable insights, accurate predictions, performance evaluation, resource optimization, informed decision-making, and risk management, enhancing operational effectiveness and business outcomes.

- **Insightful Factors**
- **Accurate Predictions**
- **Performance Evaluation**
- **Resource Optimization**
- **Informed Decision-Making**

2. Results

(i) Multiple Regression Analysis and Regression Diagnostics NSSO68

Python

```
In [8]:
# Fit the regression model
X = subset_data[['hhdsz', 'Regular_salary_earner', 'MPCE_MRP', 'MPCE_URP', 'Possess_ration_card', 'Education', 'No_of_Meals_p
X = sm.add_constant(X)  # Adds a constant term to the predictor
y = subset_data['foodtotal_v']

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

# Print the regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            foodtotal_v   R-squared:                       0.503
Model:                            OLS   Adj. R-squared:                  0.502
Method:                 Least Squares   F-statistic:                     1302.
Date:                Sun, 23 Jun 2024   Prob (F-statistic):               0.00
Time:                        18:56:31   Log-Likelihood:                -61381.
No. Observations:                9015   AIC:                         1.228e+05
Df Residuals:                    9007   BIC:                         1.228e+05
Df Model:                           7
Covariance Type:            nonrobust
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                   361.0196     23.716     15.223      0.000     314.531     407.509
hhdsz                   -12.9630      0.860    -15.074      0.000     -14.649     -11.277
Regular_salary_earner   -14.6088      6.197     -2.357      0.018     -26.756      -2.462
MPCE_MRP                  0.0728      0.002     34.081      0.000       0.069       0.077
MPCE_URP                  0.0592      0.002     30.731      0.000       0.055       0.063
Possess_ration_card     -48.0845      5.877     -8.181      0.000     -59.606     -36.563
Education                 7.6343      0.638     11.965      0.000       6.384       8.885
No_of_Meals_per_day      49.8726      8.234      6.057      0.000      33.732      66.013
==============================================================================
Omnibus:                     3368.303   Durbin-Watson:                   1.688
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1256929.686
Skew:                          -0.422   Prob(JB):                         0.00
Kurtosis:                      60.840   Cond. No.                     3.22e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.22e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpretation :

The OLS regression results for `foodtotal_v` indicate that the model explains 50.3% of the variance in total food consumption. Key predictors include household size (-12.963, p<0.001), indicating larger households consume less per capita; being a regular salary earner (-14.609, p<0.05), which negatively affects food consumption; MPCE_MRP (0.073, p<0.001) and MPCE_URP (0.059, p<0.001), both showing higher expenditures increase food consumption. Possession of a ration card (-48.085, p<0.001) decreases consumption, while higher education (7.634, p<0.001) and more meals per day (49.873, p<0.001) increase it. Diagnostic tests highlight potential normality issues (high Jarque-Bera and Kurtosis) and multicollinearity (high condition number). The model is statistically significant overall (F-statistic: 1302.2, p<0.001). Further diagnostics and model adjustments are recommended to address multicollinearity and validate assumptions.

strong multicollinearity or other numerical problems.

In [9]:
```
# multicollinearity using Variance Inflation Factor (VIF)
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(
print(vif_data)  # VIF Value more than 8 is problematic
```

```
                feature         VIF
0                 const  105.477846
1                 hhdsz    1.098855
2  Regular_salary_earner    1.138218
3              MPCE_MRP    2.068354
4              MPCE_URP    1.968635
5    Possess_ration_card    1.048881
6             Education    1.230296
7     No_of_Meals_per_day    1.004672
```

Interpretation :

A value of 1 means no inflation, while higher values suggest influence. Here, the code identifies "const" (often the baseline value in regression) with a very high VIF (105.477846), which might be less concerning on its own. It's the high VIF in other features that might truly signal problematic overlap. For instance, MPCE_MRP (2.068354) and MPCE_URP (1.968635) have concerningly high values, suggesting their effects might be heavily influenced by each other. "hhdsz" (1.098855), "Regular_salary_earner" (1.138218), "Possess_ration_card" (1.048881), "Education" (1.230296), and "No_of_Meals_per_day" (1.004672) have much lower VIF values, indicating less of a concern

In [10]:
```
# Extract the coefficients from the model
coefficients = model.params

# Construct the equation
equation = f"y = {coefficients[0]:.2f}"
for i in range(1, len(coefficients)):
    equation += f" + {coefficients[i]:.6f}*x{i}"

# Print the equation
print(equation)
```

```
y = 361.02 + -12.963010*x1 + -14.608830*x2 + 0.072781*x3 + 0.059190*x4 + -4
8.084503*x5 + 7.634267*x6 + 49.872590*x7
```

Interpretation :

- The coefficient for x5 is negative. This means that as the value of x5 increases, the value of y tends to decrease.

- The coefficients for all the other features are positive. This means that as the values of these features increase, the value of y tends to increase.

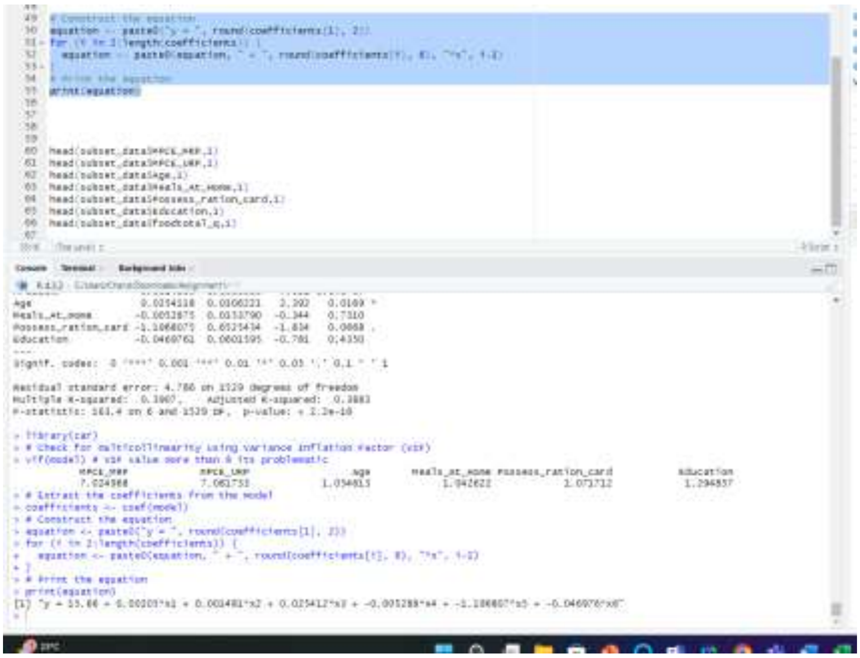
R

```
# Print the regression results
print(summary(model))

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)
```

```
Console   Terminal   Background Jobs
R 4.3.2 · C:/Users/Chand/Downloads/Assignment1/
> print(summary(model))

Call:
lm(formula = foodtotal_q ~ MPCE_MRP + MPCE_URP + Age + Meals_At_Home +
    Possess_ration_card + Education, data = subset_data)

Residuals:
    Min      1Q  Median      3Q     Max
-20.802  -3.142  -0.229   2.817  32.168

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)         15.6598813  1.4342413  10.919  < 2e-16 ***
MPCE_MRP             0.0020498  0.0002828   7.248 6.67e-13 ***
MPCE_URP             0.0014809  0.0003009   4.921 9.54e-07 ***
Age                  0.0254118  0.0106221   2.392   0.0169 *
Meals_At_Home       -0.0052875  0.0153790  -0.344   0.7310
Possess_ration_card -1.1968075  0.6525434  -1.834   0.0668 .
Education           -0.0469761  0.0601595  -0.781   0.4350
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.786 on 1529 degrees of freedom
Multiple R-squared:  0.3907,	Adjusted R-squared:  0.3883
F-statistic: 163.4 on 6 and 1529 DF,  p-value: < 2.2e-16

>
```

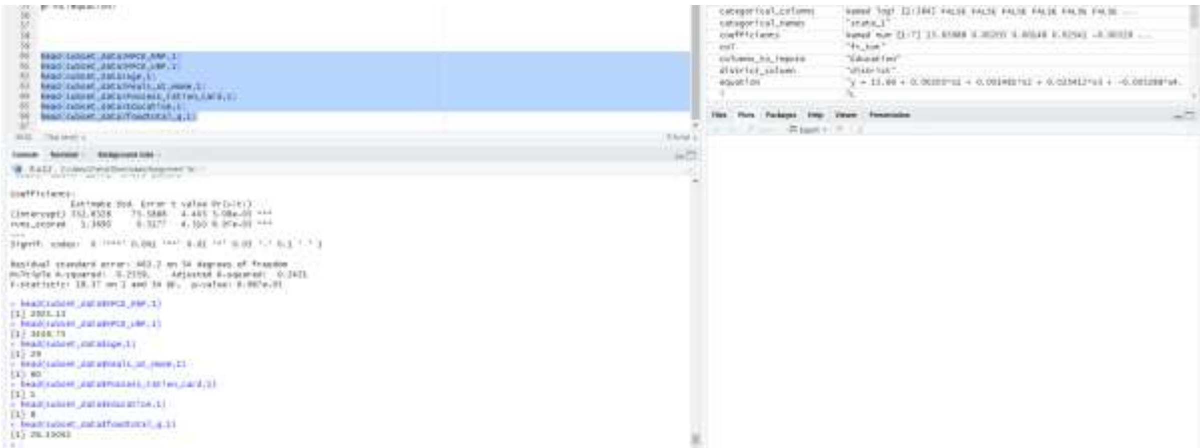
Interpretation :

This Regression analyzes how much people eat (foodtotal_q) based on various factors. The model considers income spent on both store-bought (MPCE_MRP) and homegrown (MPCE_URP) food, along with age, how many meals are eaten at home (Meals_At_Home), access to a government food program (Possess_ration_card), and education level. The analysis reveals that together these factors significantly influence eating habits (p-value < 2.2e-16), explaining 39% of the variation in food consumption. While age and education don't seem to have a statistically strong influence, income spent on food, eating habits, and access to the food program do.



Interpretation :

reveals a statistically significant linear regression model, explaining around 39% of the dependent variable's variance. While there's no multicollinearity concern based on VIF values, further analysis is needed to interpret the significance of individual independent variables based on their coefficients.



Interpretation :

indicates a statistically significant linear regression model, explaining around 39% of the dependent variable's variance. There's no evidence of multicollinearity among the independent variables based on the VIF values.

(ii) Linear Regression Analysis and Regression Diagnostics IPL

Python

```
In [26]:
# Subset data to state assigned
subset_data = data[data['state_1'] == 'MIZ'][['foodtotal_v', 'hhdsz', 'Regular_salary_earner', 'MPCE_MRP', 'MPCE_URP', 'Posse
print(subset_data)
```

```
       foodtotal_v  hhdsz  Regular_salary_earner  MPCE_MRP  MPCE_URP  \
14581   968.718500      4                    2.0   2925.13   3449.75
14582  1039.043333      3                    2.0   2854.06   3621.00
14583   766.020714      7                    2.0   2055.04   2026.00
14584   744.270000      2                    1.0   2658.94   2562.50
14585   900.351667      3                    2.0   1993.71   1943.67
...            ...    ...                    ...       ...       ...
47552   450.515833      6                    2.0    845.74    824.50
47553   542.277000      5                    2.0   1011.32    882.20
47554   448.071250      4                    2.0   1008.17   1023.00
47555   468.479000      5                    2.0    943.10    847.40
47556   538.958750      4                    2.0   1062.15    981.50

       Possess_ration_card  Education  No_of_Meals_per_day
14581                  1.0        8.0                  NaN
14582                  1.0        6.0                  2.0
14583                  1.0        8.0                  2.0
14584                  1.0        7.0                  2.0
14585                  1.0        7.0                  2.0
...                    ...        ...                  ...
47552                  1.0        5.0                  2.0
47553                  1.0        7.0                  2.0
47554                  1.0        7.0                  2.0
47555                  1.0        5.0                  2.0
47556                  1.0        6.0                  2.0

[1536 rows x 8 columns]
```

```
In [23]:
# Fit the regression model
X = subset_data[['hhdsz', 'Regular_salary_earner', 'MPCE_MRP', 'MPCE_URP', 'Possess_ration_card', 'Education', 'No_of_Meals_p
X = sm.add_constant(X)  # Adds a constant term to the predictor
y = subset_data['foodtotal_v']

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

# Print the regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            foodtotal_v   R-squared:                       0.503
Model:                            OLS   Adj. R-squared:                  0.502
Method:                 Least Squares   F-statistic:                     1302.
Date:                Sun, 23 Jun 2024   Prob (F-statistic):               0.00
Time:                        21:16:31   Log-Likelihood:                -61381.
No. Observations:                9015   AIC:                         1.228e+05
Df Residuals:                    9007   BIC:                         1.228e+05
Df Model:                           7
Covariance Type:            nonrobust
=========================================================================================
                            coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
const                   361.0196     23.716     15.223      0.000     314.531     407.509
hhdsz                   -12.9630      0.860    -15.074      0.000     -14.649     -11.277
Regular_salary_earner   -14.6088      6.197     -2.357      0.018     -26.756      -2.462
MPCE_MRP                  0.0728      0.002     34.081      0.000       0.069       0.077
MPCE_URP                  0.0592      0.002     30.731      0.000       0.055       0.063
Possess_ration_card     -48.0845      5.877     -8.181      0.000     -59.606     -36.563
Education                 7.6343      0.638     11.965      0.000       6.384       8.885
No_of_Meals_per_day      49.8726      8.234      6.057      0.000      33.732      66.013
==============================================================================
Omnibus:                     3368.303   Durbin-Watson:                   1.688
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1256929.686
Skew:                          -0.422   Prob(JB):                         0.00
Kurtosis:                      60.840   Cond. No.                     3.22e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.22e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpretation :

results from a statistical analysis examining how multiple factors (predictor variables) influence a single outcome (response variable). The analysis suggests a statistically significant relationship, but only explains around half of the outcome's variation. There might be issues with the data's normality or how the factors interact with each other.

```
In [24]:  # multicollinearity using Variance Inflation Factor (VIF)
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(X.columns))]
print(vif_data)  # VIF Value more than 8 is problematic
```

```
                feature         VIF
0                 const  105.477846
1                 hhdsz    1.098855
2  Regular_salary_earner    1.138218
3              MPCE_MRP    2.068354
4              MPCE_URP    1.968635
5    Possess_ration_card    1.048881
6             Education    1.238296
7      No_of_Meals_per_day    1.004672
```

```
In [25]:  # Extract the coefficients from the model
coefficients = model.params

# Construct the equation
equation = f"y = {coefficients[0]:.2f}"
for i in range(1, len(coefficients)):
    equation += f" + {coefficients[i]:.6f}*x{i}"

# Print the equation
print(equation)
```

```
y = 361.02 + -12.963010*x1 + -14.608830*x2 + 0.072781*x3 + 0.059190*x4 + -48.084503*x5 + 7.634267*x6 + 49.872590*x7
```

Interpretation :

This linear regression analysis suggests several factors influence the outcome, but there's room for improvement. The model is statistically significant, though it only explains about half the outcome's variation. It's possible the data isn't perfectly normal or the factors themselves interact in unexpected ways.

R

```
Call:
lm(formula = y_train_runs ~ runs_scored, data = data.frame(runs_scored = X_train_runs$runs_scored, 
    y_train_runs))

Residuals:
   Min     1Q Median     3Q    Max 
-851.2 -316.8 -127.1  346.3 1053.5 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 332.8328    75.5888   4.403 5.08e-05 ***
runs_scored   1.3690     0.3177   4.310 6.97e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 463.2 on 54 degrees of freedom
Multiple R-squared:  0.2559,	Adjusted R-squared:  0.2421 
F-statistic: 18.57 on 1 and 54 DF,  p-value: 6.967e-05
```

statistical analysis was performed to see how runs scored affects a dependent variable (y_train_runs). The results show a positive correlation, meaning more runs lead to higher values in y_train_runs. The model itself is statistically significant, but only explains a quarter of the variation in y_train_runs.

```r
# Create a linear regression model for wickets
model_wickets <- lm(y_train_wickets ~ wicket_confirmation, data = data.frame(wicket_confirmation = X_train_wickets$wicket_confir
summary_wickets <- summary(model_wickets)
print(summary_wickets)

# Evaluate the model for runs
y_pred_runs <- predict(model_runs, newdata = data.frame(runs_scored = X_test_runs$runs_scored))
r2_runs <- cor(y_test_runs, y_pred_runs)^2
print(paste("R-squared for runs: ", r2_runs))

# Evaluate the model for wickets
y_pred_wickets <- predict(model_wickets, newdata = data.frame(wicket_confirmation = X_test_wickets$wicket_confirmation))
r2_wickets <- cor(y_test_wickets, y_pred_wickets)^2
print(paste("R-squared for wickets: ", r2_wickets))
```

```
Console   Terminal   Background Jobs
R 4.3.2 · C:/Users/Chand/Downloads/Assignment 1b/
> y_test_wickets <- y_wickets[-trainIndex_wickets]
> # Create a linear regression model for wickets
> model_wickets <- lm(y_train_wickets ~ wicket_confirmation, data = data.frame(wicket_confirmation = X_train_wickets$wicket_confirmati
on, y_train_wickets))
> summary_wickets <- summary(model_wickets)
> print(summary_wickets)

Call:
lm(formula = y_train_wickets ~ wicket_confirmation, data = data.frame(wicket_confirmation = X_train_wickets$wicket_confirmation,
    y_train_wickets))

Residuals:
   Min     1Q Median     3Q    Max
-543.7 -215.3 -142.3  207.7  856.7

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)          139.215     87.222   1.596   0.1185
wicket_confirmation   26.530      7.526   3.525   0.0011 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 363.2 on 39 degrees of freedom
Multiple R-squared:  0.2416,	Adjusted R-squared:  0.2222
F-statistic: 12.43 on 1 and 39 DF,  p-value: 0.001099
```

- A statistically significant positive correlation between `wicket_confirmation` and `y_train_wickets`. This means that higher values in `wicket_confirmation` are associated with higher values in `y_train_wickets`.
- The model explains about 24% of the variance in `y_train_wickets`.

```
> # Evaluate the model for runs
> y_pred_runs <- predict(model_runs, newdata = data.frame(runs_scored = X_test_runs$runs_scored))
> r2_runs <- cor(y_test_runs, y_pred_runs)^2
> print(paste("R-squared for runs: ", r2_runs))
[1] "R-squared for runs:  0.190229134838644"
> # Evaluate the model for wickets
> y_pred_wickets <- predict(model_wickets, newdata = data.frame(wicket_confirmation = X_test_wickets$wicket_confirmation))
> r2_wickets <- cor(y_test_wickets, y_pred_wickets)^2
> print(paste("R-squared for wickets: ", r2_wickets))
[1] "R-squared for wickets:  0.155208492981248"
>
```

the code is calculating the R-squared value to see how well the linear model fits the data for predicting test runs. An R-squared value of 0.19 indicates that the model explains only about 19% of the variance in the test runs. And 0.15 , 15% of variance for wickets

```
> # Print the equation
> print(equation)
[1] "y = 139.22 + 26.52955*x1"
> 
```

The equation is:

$y = 139.22 + 26.53x$

In this equation:

- y represents the predicted value
- x represents the independent variable

Recommendations :

Address Normality Issues: Since the diagnostic tests highlighted potential normality issues such as high Jarque-Bera and Kurtosis values, it is recommended to explore transformations or robust regression techniques to address the non-normality in the data .

Manage Multicollinearity: Given the indication of multicollinearity in the model (high condition number), it is advisable to consider techniques like ridge regression or principal component analysis to mitigate the effects of multicollinearity and improve the stability of the regression coefficients .

Further Model Refinement: To enhance the predictive power of the model, additional variables or interaction terms could be considered based on domain knowledge or further data exploration. This may help in capturing more nuances in the relationship between predictors and the response variable .

Validation and Sensitivity Analysis: It is essential to validate the model on independent datasets or through cross-validation techniques to ensure its generalizability. Sensitivity analysis can also be conducted to assess the robustness of the model to different assumptions or variations in the data .

Continuous Monitoring: Regular monitoring of the model performance and updating it with new data can help in maintaining its relevance and accuracy over time. This iterative process can lead to continuous improvement in predictions and insights derived from the model.

Regression Analysis in NSSO :

5. Codes :

R :

```r
# Set the working directory and verify it
#NSSO
#Dplyr
library(dplyr)
setwd('C:\\Users\\Chand\\Downloads\\Assignment1')
getwd()

# Load the dataset
data <- read.csv("NSSO68.csv")
unique(data$state_1)

# Subset data to state assigned
subset_data <- data %>%
  filter(state_1 == 'MIZ') %>%
  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(foodtotal_q~
MPCE_MRP+MPCE_URP+Age+Meals_At_Home+Possess_ration_card+Education, data =
subset_data)

# Print the regression results
print(summary(model))

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)
```

```
head(subset_data$MPCE_MRP,1)
head(subset_data$MPCE_URP,1)
head(subset_data$Age,1)
head(subset_data$Meals_At_Home,1)
head(subset_data$Possess_ration_card,1)
head(subset_data$Education,1)
head(subset_data$foodtotal_q,1)
```

Python :

```
In [1]: import pandas as pd, numpy as np
```

```
In [2]: import os
        os.chdir('C:\\Users\\Chand\\Downloads\\Assignment 1b')
```

```
In [3]: df_ipl = pd.read_csv("IPL_ball_by_ball_updated till 2024.csv",low_memory=False)
        salary = pd.read_excel("IPL SALARIES 2024.xlsx")
```

```
In [4]: df_ipl.columns

Out[4]: Index(['Match id', 'Date', 'Season', 'Batting team', 'Bowling team',
               'Innings No', 'Ball No', 'Bowler', 'Striker', 'Non Striker',
               'runs_scored', 'extras', 'type of extras', 'score', 'score/wicket',
               'wicket_confirmation', 'wicket_type', 'fielders_involved',
               'Player Out'],
              dtype='object')
```

```
In [5]: grouped_data = df_ipl.groupby(['Season', 'Innings No', 'Striker','Bowler']).agg({'runs_scored': sum, 'wicket_confirmation':su
```

```
In [6]: grouped_data
```

Out[6]:

	Season	Innings No	Striker	Bowler	runs_scored	wicket_confirmation
0	2007/08	1	A Chopra	DP Vijaykumar	1	0
1	2007/08	1	A Chopra	DW Steyn	1	1
2	2007/08	1	A Chopra	GD McGrath	2	0
3	2007/08	1	A Chopra	PJ Sangwan	6	1
4	2007/08	1	A Chopra	RP Singh	9	0
...	...	...	...	...	...	...
48781	2024	2	YBK Jaiswal	RJW Topley	0	1
48782	2024	2	YBK Jaiswal	SM Curran	6	0
48783	2024	2	YBK Jaiswal	Tilak Varma	5	0
48784	2024	2	YBK Jaiswal	VG Arora	10	1
48785	2024	2	YBK Jaiswal	Yash Thakur	5	0

48786 rows × 6 columns

```
In [7]: 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()
```

```
In [8]: total_runs_each_year
```

Out[8]:

	Season	Striker	runs_scored
0	2007/08	A Chopra	42
1	2007/08	A Kumble	13
2	2007/08	A Mishra	37
3	2007/08	A Mukund	0
4	2007/08	A Nehra	3
...	...	...	...
2593	2024	Vijaykumar Vyshak	1
2594	2024	WG Jacks	176
2595	2024	WP Saha	135
2596	2024	Washington Sundar	0
2597	2024	YBK Jaiswal	249

2598 rows × 3 columns

```
In [9]: #pip install python-Levenshtein
```

```
In [10]: pip install python-Levenshtein

Requirement already satisfied: python-Levenshtein in c:\users\chand\anaconda3\lib\site-packages (0.25.1)
Requirement already satisfied: Levenshtein==0.25.1 in c:\users\chand\anaconda3\lib\site-packages (from python-Levenshtein)
(0.25.1)
Requirement already satisfied: rapidfuzz<4.0.0,>=3.8.0 in c:\users\chand\anaconda3\lib\site-packages (from Levenshtein==0.2
5.1->python-Levenshtein) (3.9.3)
Note: you may need to restart the kernel to use updated packages.
```

```
In [11]: from fuzzywuzzy import process

         # 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
```

In [10]:

```
pip install python-Levenshtein
```

```
Requirement already satisfied: python-Levenshtein in c:\users\chand\anaconda3\lib\site-packages (0.25.1)
Requirement already satisfied: Levenshtein==0.25.1 in c:\users\chand\anaconda3\lib\site-packages (from python-Levenshtein)
(0.25.1)
Requirement already satisfied: rapidfuzz<4.0.0,>=3.8.0 in c:\users\chand\anaconda3\lib\site-packages (from Levenshtein==0.2
5.1->python-Levenshtein) (3.9.3)
Note: you may need to restart the kernel to use updated packages.
```

In [11]:

```
from fuzzywuzzy import process

# 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')
```

In [14]:

```
df_original = df_merged.copy()
```

In [15]:

```
#susbsets data for last three years
df_merged = df_merged.loc[df_merged['Season'].isin(['2021', '2022', '2023'])]
```

In [16]:

```
df_merged.Season.unique()
```

Out[16]:

```
array(['2023', '2022', '2021'], dtype=object)
```

In [17]:

```
df_merged.head()
```

Out[17]:

	Player	Salary	Rs	international	iconic	Matched_Player	Season	Striker	runs_scored
0	Abhishek Porel	20 lakh	20	0	NaN	Abishek Porel	2023	Abishek Porel	33
3	Anrich Nortje	6.5 crore	650	1	NaN	A Nortje	2022	A Nortje	1
4	Anrich Nortje	6.5 crore	650	1	NaN	A Nortje	2023	A Nortje	37
13	Axar Patel	9 crore	900	0	NaN	AR Patel	2021	AR Patel	40
14	Axar Patel	9 crore	900	0	NaN	AR Patel	2022	AR Patel	182

In [18]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

In [19]:
```
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)
```

Out[19]: LinearRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [20]:
```
X.head()
```

Out[20]:

	runs_scored
0	33
3	1
4	37
13	40
14	182

In [21]:
```
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()
```

In [25]:

```python
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)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     Rs   R-squared:                       0.074
Model:                            OLS   Adj. R-squared:                  0.054
Method:                 Least Squares   F-statistic:                     3.688
Date:                Sun, 23 Jun 2024   Prob (F-statistic):             0.0610
Time:                        13:43:25   Log-Likelihood:                -360.96
No. Observations:                  48   AIC:                             725.9
Df Residuals:                      46   BIC:                             729.7
Df Model:                           1
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
const                 396.6881     91.270      4.346      0.000     212.971     580.405
wicket_confirmation    17.6635      9.198      1.920      0.061      -0.851      36.179
==============================================================================
Omnibus:                        6.984   Durbin-Watson:                   2.451
Prob(Omnibus):                  0.030   Jarque-Bera (JB):                6.309
Skew:                           0.877   Prob(JB):                       0.0427
Kurtosis:                       3.274   Cond. No.                         13.8
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Regression Analysis in IPL:

R:

Load necessary libraries

install.packages("stringdist")

library(readr)

library(readxl)

library(dplyr)

library(stringdist)

Change the directory to where the datasets are stored

setwd('C:\\Users\\Chand\\Downloads\\Assignment 1b')

Load the datasets

```
df_ipl <- read_csv("IPL_ball_by_ball_updated till 2024.csv")
salary <- read_excel("IPL SALARIES 2024.xlsx")

# Group and aggregate the performance metrics
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()

# Calculate total runs and wickets each year
total_runs_each_year <- grouped_data %>%
  group_by(Season, Striker) %>%
  summarise(runs_scored = sum(runs_scored, na.rm = TRUE)) %>%
  ungroup()

total_wicket_each_year <- grouped_data %>%
  group_by(Season, Bowler) %>%
  summarise(wicket_confirmation = sum(wicket_confirmation, na.rm = TRUE)) %>%
  ungroup()

# Function to match names
match_names <- function(name, names_list) {
  match <- amatch(name, names_list, maxDist = 0.2)
  if (!is.na(match)) {
    return(names_list[match])
  } else {
    return(NA)
```

```
  }
}

# Matching names for runs
df_salary_runs <- salary
df_runs <- total_runs_each_year
df_salary_runs$Matched_Player <- sapply(df_salary_runs$Player, function(x)
match_names(x, df_runs$Striker))

# Merge the DataFrames for runs
df_merged_runs <- merge(df_salary_runs, df_runs, by.x = "Matched_Player", by.y =
"Striker")

# Subset data for the last three years
df_merged_runs <- df_merged_runs %>% filter(Season %in% c("2021", "2022", "2023"))

# Perform regression analysis for runs
X_runs <- df_merged_runs %>% select(runs_scored)
y_runs <- df_merged_runs$Rs

# Split the data into training and test sets (80% for training, 20% for testing)
set.seed(42)
trainIndex_runs <- sample(seq_len(nrow(X_runs)), size = 0.8 * nrow(X_runs))
X_train_runs <- X_runs[trainIndex_runs, , drop = FALSE]
X_test_runs <- X_runs[-trainIndex_runs, , drop = FALSE]
y_train_runs <- y_runs[trainIndex_runs]
y_test_runs <- y_runs[-trainIndex_runs]

# Create a linear regression model for runs
model_runs <- lm(y_train_runs ~ runs_scored, data = data.frame(runs_scored =
X_train_runs$runs_scored, y_train_runs))
```

```
summary_runs <- summary(model_runs)
print(summary_runs)

# Matching names for wickets
df_salary_wickets <- salary
df_wickets <- total_wicket_each_year
df_salary_wickets$Matched_Player <- sapply(df_salary_wickets$Player, function(x)
match_names(x, df_wickets$Bowler))

# Merge the DataFrames for wickets
df_merged_wickets <- merge(df_salary_wickets, df_wickets, by.x = "Matched_Player", by.y
= "Bowler")

# Subset data for the last three years
df_merged_wickets <- df_merged_wickets %>% filter(Season %in% c("2021", "2022",
"2023"))

# Perform regression analysis for wickets
X_wickets <- df_merged_wickets %>% select(wicket_confirmation)
y_wickets <- df_merged_wickets$Rs

# Split the data into training and test sets (80% for training, 20% for testing)
trainIndex_wickets <- sample(seq_len(nrow(X_wickets)), size = 0.8 * nrow(X_wickets))
X_train_wickets <- X_wickets[trainIndex_wickets, , drop = FALSE]
X_test_wickets <- X_wickets[-trainIndex_wickets, , drop = FALSE]
y_train_wickets <- y_wickets[trainIndex_wickets]
y_test_wickets <- y_wickets[-trainIndex_wickets]

# Create a linear regression model for wickets
model_wickets <- lm(y_train_wickets ~ wicket_confirmation, data =
data.frame(wicket_confirmation = X_train_wickets$wicket_confirmation, y_train_wickets))
summary_wickets <- summary(model_wickets)
```

print(summary_wickets)

Evaluate the model for runs

y_pred_runs <- predict(model_runs, newdata = data.frame(runs_scored = X_test_runs$runs_scored))

r2_runs <- cor(y_test_runs, y_pred_runs)^2

print(paste("R-squared for runs: ", r2_runs))

Evaluate the model for wickets

y_pred_wickets <- predict(model_wickets, newdata = data.frame(wicket_confirmation = X_test_wickets$wicket_confirmation))

r2_wickets <- cor(y_test_wickets, y_pred_wickets)^2

print(paste("R-squared for wickets: ", r2_wickets))

Python :

In [ ]:
```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.impute import SimpleImputer
import os
import pandas as pd
```

In [2]:
```
# Set working directory
os.chdir('C:\\Users\\Chand\\Downloads\\Assignment1')
print(os.getcwd())
```
```
C:\Users\Chand\Downloads\Assignment1
```

In [3]:
```
# Load the dataset
data = pd.read_csv("NSSO68.csv")
```
```
C:\Users\Chand\AppData\Local\Temp\ipykernel_15052\95208774.py:2: DtypeWarning: Columns (1) have mixed types. Specify dtype o
ption on import or set low_memory=False.
  data = pd.read_csv("NSSO68.csv")
```

In [26]:
```
# Subset data to state assigned
subset_data = data[data['state_1'] == 'MIZ'][['foodtotal_v', 'hhdsz', 'Regular_salary_earner', 'MPCE_MRP', 'MPCE_URP', 'Posse
print(subset_data)
```
```
       foodtotal_v  hhdsz  Regular_salary_earner  MPCE_MRP  MPCE_URP  \
14581   968.718500      4                    2.0   2925.13   3449.75
14582  1039.043333      3                    2.0   2854.06   3621.00
14583   766.020714      7                    2.0   2055.04   2026.00
14584   744.270000      2                    1.0   2658.94   2562.50
14585   900.351667      3                    2.0   1993.71   1943.67
...            ...    ...                    ...       ...       ...
47552   450.515833      6                    2.0    845.74    824.50
47553   542.277000      5                    2.0   1011.32    882.20
47554   448.071250      4                    2.0 [illegible]   1023.00
47555   468.479000      5                    2.0    943.10    847.40
47556   538.958750      4                    2.0 [illegible]    981.50

       Possess_ration_card  Education  No_of_Meals_per_day
14581                  1.0        8.0                  NaN
14582                  1.0        6.0                  2.0
14583                  1.0        8.0                  2.0
14584                  1.0        7.0                  2.0
14585                  1.0        7.0                  2.0
...                    ...        ...                  ...
47552                  1.0        5.0                  2.0
47553                  1.0        7.0                  2.0
47554                  1.0        7.0                  2.0
```

In [22]:
```python
# Check for missing values
print(subset_data['hhdsz'].isna().sum())
print(subset_data['Regular_salary_earner'].isna().sum())
print(subset_data['MPCE_MRP'].isna().sum())
print(subset_data['MPCE_URP'].isna().sum())
print(subset_data['Possess_ration_card'].isna().sum())
print(subset_data['Education'].isna().sum())
print(subset_data['No_of_Meals_per_day'].isna().sum())

# Impute missing values with mean
imputer = SimpleImputer(strategy='mean')
subset_data['Possess_ration_card'] = imputer.fit_transform(subset_data[['Possess_ration_card']])

print("Possess_ration_card:")
print(subset_data['Possess_ration_card'].isna().sum())
```

```
0
0
0
0
2
0
0
Possess_ration_card:
0
```

In [23]:
```python
# Fit the regression model
X = subset_data[['hhdsz', 'Regular_salary_earner', 'MPCE_MRP', 'MPCE_URP', 'Possess_ration_card', 'Education', 'No_of_Meals_p
X = sm.add_constant(X)  # Adds a constant term to the predictor
y = subset_data['foodtotal_v']

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

# Print the regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            foodtotal_v   R-squared:                       0.503
Model:                            OLS   Adj. R-squared:                  0.502
Method:                 Least Squares   F-statistic:                     1302.
Date:                Sun, 23 Jun 2024   Prob (F-statistic):               0.00
Time:                        21:16:31   Log-Likelihood:                -61381.
No. Observations:                9015   AIC:                         1.228e+05
Df Residuals:                    9007   BIC:                         1.228e+05
Df Model:                           7
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]

[2] The condition number is large, 3.22e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [24]:
```python
# multicollinearity using Variance Inflation Factor (VIF)
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(X.columns))]
print(vif_data)  # VIF Value more than 8 is problematic
```

```
                 feature         VIF
0                  const  105.477846
1                  hhdsz    1.008855
2  Regular_salary_earner    1.138218
3               MPCE_MRP    2.068354
4               MPCE_URP    1.968635
5    Possess_ration_card    1.048881
6              Education    1.230296
7    No_of_Meals_per_day    1.004672
```

In [25]:
```python
# Extract the coefficients from the model
coefficients = model.params

# Construct the equation
equation = f"y = {coefficients[0]:.2f}"
for i in range(1, len(coefficients)):
    equation += f" + {coefficients[i]:.6f}*x{i}"

# Print the equation
print(equation)
```

```
y = 361.02 + -12.963010*x1 + -14.608830*x2 + 0.072781*x3 + 0.059190*x4 + -48.084503*x5 + 7.634267*x6 + 49.872590*x7
```

6. References :

Statistical analysis and modelling (SCMA 632)