VIRGINIA COMMONWEALTH UNIVERSITY

STATISTICAL ANALYSIS AND MODELLING (SCMA 632)

A2: Multiple Regression Analysis for NSSO68

Linear Regression Analysis for IPL Performance and Salary

UJWAL P

V01107757

Date of Submission: 23-06-2024

CONTENTS

Introduction

Data:

NSSO-Consumption Dataset: Tracks consumption of various goods (grains, oils, fruits) across Indian states with demographics.

Ball by Ball Dataset: Details every ball bowled in IPL matches (2008-2022) with team, runs scored, and player performance.

IPL Matches Dataset: Provides text information on IPL matches (2008-2022) including dates, cities, teams, and player details.

IPL Salary Dataset: Includes yearly salary information for IPL players in dollars.

Objectives:

NSSO Dataset: Use multiple regression analysis to understand factors affecting consumption patterns in India. This includes checking the model's validity and addressing any issues for better results.

IPL Data: Analyze the relationship between player performance and salary in the IPL. This involves:

Using linear regression to see if salary can be predicted by performance.

Studying correlations between performance factors and salaries.

Identifying top performers, underperformers, and salary trends.

Analyzing salary data distributions for key players.

Business Importance:

Regression analysis helps businesses:

- Gain valuable insights into consumer behavior and player performance.
- Make accurate predictions about future outcomes.
- Evaluate performance and resource allocation.
- Make informed decisions based on data.
- Manage risks associated with business operations.

Overall, this project aims to leverage data analysis to gain insights that can improve business decision-making and outcomes.

Results: NSSO68 R

```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)

# 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
```

42:1 (Top Level) R Script

Console Background Jobs

R 4.4.1 · C:/Assignment1/

```
(Intercept)          11.1732494  1.2886441   8.671  < 2e-16 ***
MPCE_MRP              0.0047967  0.0001529  31.369  < 2e-16 ***
MPCE_URP              0.0004906  0.0001031   4.757 2.03e-06 ***
Age                   0.0660219  0.0116765   5.654 1.67e-08 ***
Meals_At_Home         0.1684741  0.0119247  14.128  < 2e-16 ***
Possess_ration_card -5.1474814  0.5656037  -9.101  < 2e-16 ***
Education            -0.1770245  0.0420508  -4.210 2.61e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 9.874 on 4118 degrees of freedom
  (3 observations deleted due to missingness)
Multiple R-squared:  0.4218,    Adjusted R-squared:  0.421
F-statistic: 500.7 on 6 and 4118 DF,  p-value: < 2.2e-16
```

Interpretation :

A relationship between a response variable and multiple predictors using linear regression. It checks for multicollinearity (correlated predictors) and extracts coefficients to understand how each predictor affects the response. The full model summary is needed for a detailed interpretation.

Results:

```
> # Print the equation
> print(equation)
[1] "y = 15.66 + 0.00205*x1 + 0.001481*x2 + 0.025412*x3 + -0.005288*x4 + -1.196807*x5 + -0.046976*x6"
>
```

analyzes a linear regression model. It examines the overall fit (summary), checks for correlated predictor variables (VIF), extracts how much each variable affects the outcome (coefficients), and builds a formula to predict the outcome based on those factors. By analyzing these aspects, the code helps understand the relationships between variables and how they influence the final result.

Results: NSSO68 Python

```
[11]: # Subset data to state assigned
subset_data = data[data['state_1'] == 'RJ'][['foodtotal_v', 'hhdsz', 'Regular_salary_earner', 'MPCE_MRP', 'MPCE_URP', 'Possess_ration_card', 'Education',
print(subset_data)
```

```
       foodtotal_v  hhdsz  Regular_salary_earner  MPCE_MRP  MPCE_URP  \
29288   777.287833      6                    2.0   2418.53   2042.00
29289   504.344250      4                    2.0   1968.38   1503.50
29290   548.630000     10                    2.0   1170.03   1023.30
29291   715.650500      2                    2.0   1702.70   1656.50
29292   434.551583     12                    1.0   1258.96   1167.50
...            ...    ...                    ...       ...       ...
94159   352.571333      6                    2.0    880.05    951.83
94160   369.088600      5                    2.0   3823.12  34518.80
94161   290.485125      8                    2.0    545.02    705.00
94162   285.053000      8                    2.0    633.28    532.75
94163   362.775833      6                    2.0    963.97   1351.83

       Possess_ration_card  Education  No_of_Meals_per_day
29288                  1.0       12.0                  2.0
29289                  1.0       12.0                  2.0
29290                  1.0        1.0                  2.0
29291                  1.0        7.0                  2.0
29292                  1.0        6.0                  2.0
...                    ...        ...                  ...
94159                  1.0        1.0                  2.0
94160                  2.0        1.0                  2.0
94161                  1.0        1.0                  2.0
94162                  2.0        1.0                  2.0
94163                  1.0        1.0                  2.0

[9015 rows x 8 columns]
```

```
[13]: # Fit the regression model
X = subset_data[['hhdsz', 'Regular_salary_earner', 'MPCE_MRP', 'MPCE_URP', 'Possess_ration_card', 'Education', 'No_of_Meals_per_day']]
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:                Mon, 24 Jun 2024   Prob (F-statistic):               0.00
Time:                        00:53:04   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
```

Interpretation :

performs a linear regression analysis to see how well certain factors (predictors) influence an outcome (response variable). Here's a breakdown:

The code checks the overall model fit (summary) and variable correlations (VIF) to assess model validity.

It extracts coefficients to determine how each predictor variable affects the response variable.

Finally, it builds a formula to predict the outcome based on those factors and coefficients.

In essence, this code helps analyze the relationships between variables and how they influence the final result through linear regression.

```
[14]: # 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    1.230296
7     No_of_Meals_per_day    1.004672
```

```
[15]: # 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 :

it appears to be performing the following analysis:

It checks the assumptions of the linear regression model, including multicollinearity (correlated variables) using VIF.

It extracts the coefficients, which indicate how much each input variable affects the outcome variable.

It builds a formula to predict the outcome variable based on the input variables and their coefficients.

Overall, this code helps analyse the relationship between a response variable and several predictor variables through linear regression.

Results: IPL R

```r
# 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_tra
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$B

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

75:20 (Top Level) R Script

Console Background Jobs

R 4.4.1 · C:/Assignment1/

```
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

>
```

Interpretation :

It extracts the coefficients, which indicate how much each input variable (runs_scored) affects the outcome variable (total wicket count). For every run scored, the expected total wicket count increases by 1.369.

It checks the overall model fit (summary) to see how well the model explains the data (R-squared is 0.2559).

It assesses the significance of the coefficients (p-value), where a small p-value (here, both are less than 0.00005) indicates a strong relationship between the predictor and response variable.

suggests a statistically significant positive relationship between runs scored and total wicket count in cricket matches, though the model only explains a quarter of the variation in the data.

```r
# 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_cc
r2_wickets <- cor(y_test_wickets, y_pred_wickets)^2
print(paste("R-squared for wickets: ", r2_wickets))
```

```
140:52   (Top Level)                                                    R Script
Console   Background Jobs
R 4.4.1 · C:/Assignment1/
> r2_wickets <- cor(y_test_wickets, y_pred_wickets)^2
> print(paste("R-squared for wickets: ", r2_wickets))
[1] "R-squared for wickets:  0.0985013558096498"
> # 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_confir
mation))
> r2_wickets <- cor(y_test_wickets, y_pred_wickets)^2
> print(paste("R-squared for wickets: ", r2_wickets))
[1] "R-squared for wickets:  0.0985013558096498"
>
```

Interpretation :

It predicts runs and wickets using the respective models and stores the results in y_pred_runs and y_pred_wickets.

It calculates the R-squared value between the predicted values and the actual values (y_test_runs and y_test_wickets) and stores the results in r2_runs and r2_wickets.

the R-squared values for runs and wickets.

Results: IPL Python

```
[20]: df_merged.Season.unique()

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

[21]: df_merged.head()
```

[21]:

	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

```python
[25]: 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)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     Rs   R-squared:                       0.080
Model:                            OLS   Adj. R-squared:                  0.075
Method:                 Least Squares   F-statistic:                     15.83
Date:                Mon, 24 Jun 2024   Prob (F-statistic):           0.000100
Time:                        00:15:06   Log-Likelihood:                -1379.8
No. Observations:                 183   AIC:                              2764.
Df Residuals:                     181   BIC:                              2770.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     Rs   R-squared:                       0.080
Model:                            OLS   Adj. R-squared:                  0.075
Method:                 Least Squares   F-statistic:                     15.83
Date:                Mon, 24 Jun 2024   Prob (F-statistic):           0.000100
Time:                        00:15:06   Log-Likelihood:                -1379.8
No. Observations:                 183   AIC:                              2764.
Df Residuals:                     181   BIC:                              2770.
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const         430.8473     46.111      9.344      0.000     339.864     521.831
runs_scored     0.6895      0.173      3.979      0.000       0.348       1.031
==============================================================================
Omnibus:                       15.690   Durbin-Watson:                   2.100
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               18.057
Skew:                           0.764   Prob(JB):                     0.000120
Kurtosis:                       2.823   Cond. No.                         363.
==============================================================================

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

Interpretation :

This Python code performs linear regression to predict a target value (Rs) based on a feature (runs_scored). It splits data into training and testing sets, fits a model, and provides a summary to assess how well the model explains the relationship between runs scored and the target variable.

R-squared: 0.08, which is a low value, indicating the model doesn't strongly explain the relationship between runs scored and Rs.

P-value: 0.0001, which is statistically significant, meaning there is a relationship between runs scored and Rs, but the model may not be the best way to capture it.

Coefficients: These show how much Rs is expected to change on average with a one-unit change in runs scored. For example, the coefficient for "runs_scored" is 0.6895, which means that for every one-run increase in runs scored, Rs is expected to increase by 0.6895 on average.

```
                            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:                Mon, 24 Jun 2024   Prob (F-statistic):             0.0610
Time:                        00:19:36   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.
```

Interpretation :

Coefficients:

The coefficient for "const" is 396.69. This represents the intercept of the regression line, which is the predicted value of "Rs" when "wicket_confirmation" is zero.

The coefficient for "wicket_confirmation" is 17.66. This means that for every one-unit increase in "wicket_confirmation", the predicted value of "Rs" is expected to increase by 17.66 on average.

R-squared: This is 0.054, which is a relatively low value. It indicates that the model doesn't explain a strong proportion of the variance in "Rs".

P-value: This is 0.061, which is marginally significant. It suggests there might be a weak relationship between "wicket_confirmation" and "Rs", but more data or a different model might be needed to confirm this.

Overall, the analysis suggests a weak positive relationship between "wicket_confirmation" and "Rs". However, the R-squared value is low, indicating that the model doesn't capture a strong explanatory power for the data.

Recommendations :

- **Consider alternative models:** Depending on the nature of your data and the problem you're trying to solve, exploring different types of models like decision trees, random forests, or support vector machines might be beneficial.
- · **Model evaluation:** Evaluate the model performance on unseen data (a held-out test set) to assess its generalizability beyond the training data.

Codes :

Regression Analysis in NSSO68 R:

```
# Set the working directory and verify it

#NSSO

install.packages("car")

#Dplyr

library(dplyr)

setwd('C:\\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 == 'RJ') %>%

  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)
```

Regression Analysis in NSSO68 Python:

```
[1]: 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

[8]: # Set working directory
     os.chdir('C:\\Assignment1')
     print(os.getcwd())

     C:\Assignment1

[9]: # Load the dataset
     data = pd.read_csv("NSSO68.csv")

[12]: # 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())

[13]: # Fit the regression model
      X = subset_data[['hhdsz', 'Regular_salary_earner', 'MPCE_MRP', 'MPCE_URP', 'Possess_ration_card', 'Education', 'No_of_Meals_per_day']]
      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())
```

```python
[14]: # 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
```

```python
[15]: # 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)
```

Regression Analysis in IPL R:

```r
# Load necessary libraries
install.packages("stringdist")
install.packages("dplyr")
install.packages("readr")
install.packages("readxl")

library(readr)
library(readxl)
library(dplyr)
library(stringdist)

# Change the directory to where the datasets are stored
setwd("C:\\A2")

# 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)
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))

Regression Analysis in IPL Python:

```
[7]: import pandas as pd, numpy as np

[10]: import os
      os.chdir('C:\\A2')

[11]: df_ipl = pd.read_csv('IPL_ball_by_ball_updated till 2024.csv',low_memory=False)
      salary = pd.read_excel('IPL SALARIES 2024.xlsx')

[12]: df_ipl.columns

[16]: #pip install python-Levenshtein

[17]: 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')

[18]: df_original = df_merged.copy()

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

[20]: df_merged.Season.unique()

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

[21]: df_merged.head()

[22]: from sklearn.linear_model import LinearRegression
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import mean_squared_error

[23]: 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)

[23]: ▾ LinearRegression
      LinearRegression()

[24]: X.head()
```

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

```
[29]: 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)
```