VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A2: Regression - Predictive Analytics

JESIN KANDATHY JOY

V01110163

Date of Submission: 23-06-2024

CONTENTS

INTRODUCTION

This report aims to address two distinct but related analytical problems using regression analysis. The first problem involves performing multiple regression analysis on household survey data to explore various socio-economic factors and their impact on household food expenditure. The second problem examines the relationship between IPL players' performance metrics and their salaries, focusing on the past three seasons.

OBJECTIVES

NSSO Data Analysis:

- Perform multiple regression analysis.
- Conduct regression diagnostics and address any identified issues.
- Revisit and explain the significant differences observed after corrections.

IPL Data Analysis:

- Establish the relationship between player performance metrics (runs scored and wickets taken) and their salaries.
- Analyze the relationship over the last three years.
- Provide insights and recommendations based on the findings.

BUSINESS SIGNIFICANCE

NSSO Data Analysis:

- Understanding the socio-economic factors affecting household food expenditure helps in policy formulation and targeting welfare programs.

IPL Data Analysis:

- Establishing a clear relationship between player performance and salaries ensures fair compensation, optimizes team budgets, and maintains competitive balance within the league.

R CODE AND RESULT

NSSO Data Analysis:

```r
# Subset data to state assigned
subset_data <- data %>%
  filter(state_1 == 'UP') %>%
  select(foodtotal_v, hhdsz, Regular_salary_earner, MPCE_MRP, MPCE_URP, Possess_ration_card, Education, No_of_Meals_per_day)
print(subset_data)

sum(is.na(subset_data$hhdsz))
sum(is.na(subset_data$Regular_salary_earner))
sum(is.na(subset_data$MPCE_MRP))
sum(is.na(subset_data$MPCE_URP))
sum(is.na(subset_data$Possess_ration_card))
sum(is.na(subset_data$Education))
sum(is.na(subset_data$No_of_Meals_per_day))

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("Possess_ration_card")
# Impute missing values with mean
data <- impute_with_mean(data, columns_to_impute)
sum(is.na(data$Possess_ration_card))

# Fit the regression model
model <- lm(foodtotal_v~ hhdsz+Regular_salary_earner+MPCE_MRP+MPCE_URP+Possess_ration_card+Education+No_of_Meals_per_day, data = subset_data)
# Print the regression results
print(summary(model))
```

```r
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)
```

IPL Data Analysis:

```r
# 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)
  }
}
```

```r
# 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 %>% dplyr::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)
```

```
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
```

```
# 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 %>% dplyr::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)
```

```
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
```

```
# 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))
```

```
> # 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"
```

PYTHON CODE AND RESULT

NSSO Data Analysis:

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

```
# 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:                Sun, 23 Jun 2024   Prob (F-statistic):               0.00
Time:                        13:17:15   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
==============================================================================
```

```
# Check for 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

```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
```

IPL Data Analysis:

```python
# Group and aggregate the performance metrics
grouped_data = df_ipl.groupby(['Season', 'Innings No', 'Striker', 'Bowler']).agg({
    'runs_scored': sum,
    'wicket_confirmation': sum
}).reset_index()
```

```python
# Calculate total runs and wickets each year
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()
```

```python
# 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
```

```python
# Matching names for runs
df_salary_runs = salary.copy()
df_runs = total_runs_each_year.copy()
df_salary_runs['Matched_Player'] = df_salary_runs['Player'].apply(lambda x: match_names(x, df_runs['Striker'].tolist()))

# Merge the DataFrames for runs
df_merged_runs = pd.merge(df_salary_runs, df_runs, left_on='Matched_Player', right_on='Striker')

# Subset data for the last three years
df_merged_runs = df_merged_runs.loc[df_merged_runs['Season'].isin(['2021', '2022', '2023'])]

# Perform regression analysis for runs
X = df_merged_runs[['runs_scored']]  # Independent variable(s)
y = df_merged_runs['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_runs = sm.OLS(y_train, X_train_sm).fit()

# Get the summary of the model
summary_runs = model_runs.summary()
print(summary_runs)
```

```
                            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:                Sun, 23 Jun 2024   Prob (F-statistic):           0.000100
Time:                        14:40:08   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.
==============================================================================
```

```python
# Matching names for wickets
df_salary_wickets = salary.copy()
df_wickets = total_wicket_each_year.copy()
df_salary_wickets['Matched_Player'] = df_salary_wickets['Player'].apply(lambda x: match_names(x, df_wickets['Bowler'].tolist()))

# Merge the DataFrames for wickets
df_merged_wickets = pd.merge(df_salary_wickets, df_wickets, left_on='Matched_Player', right_on='Bowler')

# Subset data for the last three years
df_merged_wickets = df_merged_wickets.loc[df_merged_wickets['Season'].isin(['2021', '2022', '2023'])]

# Perform regression analysis for wickets
X = df_merged_wickets[['wicket_confirmation']]  # Independent variable(s)
y = df_merged_wickets['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_wickets = sm.OLS(y_train, X_train_sm).fit()

# Get the summary of the model
summary_wickets = model_wickets.summary()
print(summary_wickets)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     Rs   R-squared:                       0.070
Model:                            OLS   Adj. R-squared:                  0.064
Method:                 Least Squares   F-statistic:                     10.84
Date:                Sun, 23 Jun 2024   Prob (F-statistic):            0.00125
Time:                        14:40:27   Log-Likelihood:                -1094.8
No. Observations:                 145   AIC:                             2194.
Df Residuals:                     143   BIC:                             2200.
Df Model:                           1
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
const                 373.6944     55.698      6.709      0.000     263.597     483.792
wicket_confirmation    17.5404      5.327      3.293      0.001       7.011      28.070
==============================================================================
Omnibus:                       23.226   Durbin-Watson:                   1.989
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               28.667
Skew:                           1.038   Prob(JB):                     5.96e-07
Kurtosis:                       3.658   Cond. No.                         15.2
==============================================================================
```

```python
# Perform regression analysis for runs
X = df_merged_runs[['runs_scored']]  # Independent variable(s)
y = df_merged_runs['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 linear regression model
model_runs = LinearRegression()

# Fit the model on the training data
model_runs.fit(X_train, y_train)

# Make predictions
y_pred = model_runs.predict(X_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)
print(f"R-squared: {r2}")
```

```
R-squared: 0.12271550126860675
```

```python
# Perform regression analysis for wickets
X = df_merged_wickets[['wicket_confirmation']]  # Independent variable(s)
y = df_merged_wickets['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 linear regression model
model_wickets = LinearRegression()

# Fit the model on the training data
model_wickets.fit(X_train, y_train)

# Make predictions
y_pred = model_wickets.predict(X_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)
print(f"R-squared: {r2}")
```

```
R-squared: 0.014105556424401033
```

INTERPRETATIONS

NSSO Data Analysis:

The regression model analyzes the relationship between household food expenditure (foodtotal_v) and several predictors: household size (hhdsz), presence of a regular salary earner (Regular_salary_earner), monthly per capita expenditure at market prices (MPCE_MRP) and uniform recall period (MPCE_URP), possession of a ration card (Possess_ration_card), education level (Education), and number of meals per day (No_of_Meals_per_day).

Interpretations

- Larger households and those with regular salary earners spend less on food per capita.
- Higher overall spending correlates with higher food spending.
- Possession of a ration card leads to lower food expenditure.
- Higher education levels and more meals per day increase food expenditure.

Recommendations

- Target subsidies for larger households and those without regular salary earners.
- Include additional variables and interactions to improve model accuracy.
- Promote educational initiatives to enhance nutrition and well-being.

IPL Data Analysis:

Interpretations

Runs Scored vs. Salary

- The positive coefficient for runs scored (1.3690) indicates that for each additional run scored, a player's salary increases by approximately 1.37 units.
- The model explains about 25.59% of the variance in player salaries, which suggests a moderate relationship between runs scored and salary.
- The residuals show a wide range, indicating some variability in salaries that is not explained by the runs scored alone.

Wickets Taken vs. Salary

- The positive coefficient for wickets taken (26.530) suggests that for each wicket taken, a player's salary increases by approximately 26.53 units.
- The model explains about 24.16% of the variance in player salaries, indicating a moderate relationship between wickets taken and salary.
- The residuals also show considerable variability, suggesting that factors other than wickets taken affect player salaries.

Recommendations

- Include additional performance metrics and contextual factors to improve the model's explanatory power.
- Review and potentially adjust salary structures to better reflect player contributions in various performance areas.
- Players should focus on both scoring runs and taking wickets to maximize their salary potential.