VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

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

Lakshmy Cherussery Jayaprakash

V01110023

Date of Submission: 23-06-2024

CONTENTS

Introduction

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

Objective

1. Analyze the relationship between IPL player performance (e.g., runs scored, wickets taken) and the salary they receive.
   Conduct regression analysis to quantify how performance metrics impact player salaries.
2. Perform multiple regression analysis on the "NSSO68.csv" dataset to understand the factors influencing food expenditure (MPCE_MRP) in Gujarat.
   Conduct regression diagnostics to ensure the model meets assumptions and interpret the findings.

Business Significace

Insights from this analysis can inform policy decisions related to welfare programs and economic policies aimed at improving food security and consumption patterns.
This analysis can help policymakers, researchers, and businesses make informed decisions related to welfare programs, economic policies, and market strategies targeted at improving food security and consumption patterns.
Understanding how player performance correlates with salary in the IPL is crucial for team management, player negotiations, and franchise strategies. This analysis helps teams identify which performance metrics are most valuable in terms of return on investment (ROI) and player salary decisions.

Results and Interpretations

1a- IPL- Python

```
                            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:                        11:31:11   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 spe
```

Interpretation:

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

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     Rs   R-squared:                       0.001
Model:                            OLS   Adj. R-squared:                 -0.001
Method:                 Least Squares   F-statistic:                    0.6043
Date:                Sun, 23 Jun 2024   Prob (F-statistic):              0.437
Time:                        11:39:37   Log-Likelihood:                -3701.4
No. Observations:                 484   AIC:                             7407.
Df Residuals:                     482   BIC:                             7415.
Df Model:                           1
Covariance Type:            nonrobust
=====================================================================================
                          coef    std err          t      P>|t|      [0.025      0.97
-------------------------------------------------------------------------------------
const                 480.9475     36.206     13.284      0.000     409.807     552.0
wicket_confirmation     2.6144      3.363      0.777      0.437      -3.994       9.2
==============================================================================
Omnibus:                       50.971   Durbin-Watson:                   2.022
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               65.175
Skew:                           0.892   Prob(JB):                     7.04e-15
Kurtosis:                       2.774   Cond. No.                         17.0
==============================================================================

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

Interpretation:

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

1b- IPL- R

```
# A tibble: 2 × 5
  term        estimate std.error statistic p.value
  <chr>          <dbl>     <dbl>     <dbl>   <dbl>
1 (Intercept)   399.     135.         2.96 0.00832
2 total_runs      2.56     0.793      3.23 0.00468
R-squared: 0.3664739
MAPE: 345.5741
```

```
# A tibble: 2 × 5
  term          estimate std.error statistic p.value
  <chr>            <dbl>     <dbl>     <dbl>   <dbl>
1 (Intercept)      1038.      249.      4.17 0.00131
2 total_wickets    -65.8      33.8     -1.95 0.0755
R-squared: 0.2398259
MAPE: 223.6831
```

2a- NSSO Python

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                MPCE_MRP   R-squared:                       0.509
Model:                             OLS   Adj. R-squared:                  0.504
Method:                  Least Squares   F-statistic:                     100.9
Date:                 Sun, 23 Jun 2024   Prob (F-statistic):           5.97e-87
Time:                         17:45:07   Log-Likelihood:                -5115.4
No. Observations:                  592   AIC:                         1.024e+04
Df Residuals:                      585   BIC:                         1.028e+04
Df Model:                            6
Covariance Type:             nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
const                 640.1751    428.918      1.493      0.136    -202.231    1482.581
MPCE_URP                0.3823      0.029     13.116      0.000       0.325       0.440
Age                    -1.9103      4.366     -0.437      0.662     -10.486       6.665
Meals_At_Home         -28.2814      4.895     -5.778      0.000     -37.895     -18.668
Possess_ration_card   244.9228    167.782      1.460      0.145     -84.606     574.452
Education             118.5556     18.534      6.397      0.000      82.155     154.956
foodtotal_q            63.9783      7.605      8.412      0.000      49.041      78.915
==============================================================================
Omnibus:                      681.909   Durbin-Watson:                   2.054
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            68617.647
Skew:                           5.355   Prob(JB):                         0.00
...
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.73e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Interpretations:

Significant Variables: MPCE_URP, Meals_At_Home, Education, and foodtotal_q appear to be statistically significant predictors of MPCE_MRP, as indicated by their low p-values (< 0.05).

Model Fit: The model overall is statistically significant (Prob F-statistic is very low), indicating that at least some of the independent variables are helpful in predicting MPCE_MRP.

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

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

	Variable	VIF
0	MPCE_URP	3.198491
1	Age	10.318819
2	Meals_At_Home	17.285133
3	Possess_ration_card	7.822056
4	Education	5.909523
5	foodtotal_q	12.272383

Interpretations:

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

```
# Print the equation
print(equation)
```

```
y = 640.18 + 0.382287*MPCE_URP + -1.910274*Age + -28.281442*Meals_At_Home + 244.922842*Possess_ration_card + 118.555642*Education + 63.978323*foodtotal_q
```

Interpretations:

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

Intercept (Constant): 640.18

2b- R prgm

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

Residuals:
    Min      1Q  Median      3Q     Max 
-9906.9  -470.2  -157.8   233.7 18163.8 

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)    
(Intercept)         625.86295  272.38391   2.298   0.0217 *  
MPCE_URP              0.48336    0.01794  26.936  < 2e-16 ***
Age                   1.17195    2.74807   0.426   0.6698    
Meals_At_Home       -27.46606    3.32660  -8.257  3.2e-16 ***
Possess_ration_card  41.62641   92.10420   0.452   0.6514    
Education           110.00084   11.64271   9.448  < 2e-16 ***
foodtotal_q          59.42516    4.90323  12.120  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1386 on 1532 degrees of freedom
Multiple R-squared:  0.5539,	Adjusted R-squared:  0.5521 
F-statistic:   317 on 6 and 1532 DF,  p-value: < 2.2e-16
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
           MPCE_URP                 Age       Meals_At_Home Possess_ration_card 
           1.339350            1.142515            1.349263            1.116431 
          Education         foodtotal_q 
           1.089335            1.511599 
```

```
# Print the equation
print(equation)
```

```
[1] "y = 625.86 + 0.483358*x1 + 1.171946*x2 + -27.466057*x3 + 41.626412*x4 + 110.000843*x5 + 59.425163*x6"
```

Codes

1a- IPL Python

```
import pandas as pd, numpy as np
import os
os.chdir('E:\Assignments_SCMA632\Data')
df_ipl = pd.read_csv("IPL_ball_by_ball_updated till 2024.csv",low_memory=False)
salary = pd.read_excel("IPL SALARIES 2024.xlsx")
df_ipl.columns
grouped_data = df_ipl.groupby(['Season', 'Innings No', 'Striker','Bowler']).agg({'runs_scored': sum,
'wicket_confirmation':sum}).reset_index()
grouped_data
total_runs_each_year = grouped_data.groupby(['Season', 'Striker'])['runs_scored'].sum().reset_index()
total_wicket_each_year = grouped_data.groupby(['Season',
'Bowler'])['wicket_confirmation'].sum().reset_index()
total_runs_each_year
#pip install python-Levenshtein
# Convert to DataFrame
df_salary = salary.copy()
df_runs = total_runs_each_year.copy()

# Function to match names
def match_names(name, names_list):
    match, score = process.extractOne(name, names_list)
    return match if score >= 80 else None  # Use a threshold score of 80

# Create a new column in df_salary with matched names from df_runs
df_salary['Matched_Player'] = df_salary['Player'].apply(lambda x: match_names(x,
df_runs['Striker'].tolist()))

# Merge the DataFrames on the matched names
df_merged = pd.merge(df_salary, df_runs, left_on='Matched_Player', right_on='Striker')
df_original = df_merged.copy()
#susbsets data for last three years
df_merged = df_merged.loc[df_merged['Season'].isin(['2021', '2022', '2023'])]
df_merged.Season.unique()
df_merged.head()
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_percentage_error
X = df_merged[['runs_scored']] # Independent variable(s)
y = df_merged['Rs'] # Dependent variable
# Split the data into training and test sets (80% for training, 20% for testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Create a LinearRegression model
model = LinearRegression()
# Fit the model on the training data
model.fit(X_train, y_train)
X.head()
import pandas as pd
from sklearn.model_selection import train_test_split
import statsmodels.api as sm

# Assuming df_merged is already defined and contains the necessary columns
X = df_merged[['runs_scored']] # Independent variable(s)
y = df_merged['Rs'] # Dependent variable

# Split the data into training and test sets (80% for training, 20% for testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Add a constant to the model (intercept)
X_train_sm = sm.add_constant(X_train)

# Create a statsmodels OLS regression model
model = sm.OLS(y_train, X_train_sm).fit()

# Get the summary of the model
summary = model.summary()
print(summary)
from fuzzywuzzy import process

# Convert to DataFrame
df_salary = salary.copy()
df_runs = total_wicket_each_year.copy()

# Function to match names
def match_names(name, names_list):
    match, score = process.extractOne(name, names_list)
    return match if score >= 80 else None  # Use a threshold score of 80

# Create a new column in df_salary with matched names from df_runs
df_salary['Matched_Player'] = df_salary['Player'].apply(lambda x: match_names(x,
df_runs['Bowler'].tolist()))

# Merge the DataFrames on the matched names
df_merged = pd.merge(df_salary, df_runs, left_on='Matched_Player', right_on='Bowler')
df_merged[df_merged['wicket_confirmation']>10]
#susbsets data for last three years
df_merged = df_merged.loc[df_merged['Season'].isin(['2022'])]
import pandas as pd
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
```

```python
# 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)
```

<u>1b- IPL R prgm</u>

```r
library(readxl)
library(dplyr)
df_ipl=read.csv("/content/IPL_ball_by_ball_updated till 2024 (1).csv")
salary =read_excel("IPL SALARIES 2024.xlsx")
head(df_ipl)
colnames(df_ipl)
# Grouping and Summarizing the Data
grouped_data <- df_ipl %>%
  group_by(Season, Innings.No, Striker, Bowler) %>%
  summarise(
    runs_scored = sum(runs_scored, na.rm = TRUE),
    wicket_confirmation = sum(wicket_confirmation, na.rm = TRUE)
  ) %>%
  ungroup()
grouped_data
# Calculate total runs each year
total_runs_each_year <- grouped_data %>%
  group_by(Season, Striker) %>%
  summarise(total_runs = sum(runs_scored, na.rm = TRUE)) %>%
  ungroup()

#calculate total wickets each year
total_wicket_each_year <- grouped_data %>%
  group_by(Season, Bowler) %>%
  summarise(total_wickets = sum(wicket_confirmation, na.rm = TRUE)) %>%
  ungroup()
```

```
total_runs_each_year
total_wicket_each_year
```

```
install.packages("stringdist")

# Function to match names using stringdist package
match_names <- function(name, names_list) {
  dist <- stringdist::stringdist(name, names_list, method = "jw")
  match <- names_list[which.min(dist)]
  score <- 1 - min(dist)
  return(ifelse(score >= 0.8, match, NA)) # Use a threshold score of 0.8
}

# Apply fuzzy matching to match names
salary$Matched_Player <- sapply(salary$Player, match_names, names_list =
total_runs_each_year$Striker)

# Merge the DataFrames on the matched names
df_merged <- merge(salary, total_runs_each_year, by.x = "Matched_Player", by.y = "Striker")
```

```
        # Subset data for the last three years

df_merged <- df_merged %>%
  filter(Season %in% c("2021", "2022", "2023"))

# Check the unique seasons
unique(df_merged$Season)
```

```
print(colnames(df_merged))
```

```
# Independent and dependent variables
X <- df_merged$total_runs
y <- df_merged$Rs

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

```
set.seed(42)
train_indices <- sample(seq_len(nrow(df_merged)), size = 0.8 * nrow(df_merged))
train_data <- df_merged[train_indices, ]
test_data <- df_merged[-train_indices, ]

# OLS regression model
model <- lm(Rs ~ total_runs, data = train_data)

# Summary of the regression model
summary(model)

# Using the broom package to get tidy output
tidy_model <- broom::tidy(model)
print(tidy_model)

# Predicting on the test set
predictions <- predict(model, newdata = test_data)

# Calculate R-squared and Mean Absolute Percentage Error
r_squared <- summary(model)$r.squared
mape <- mean(abs((test_data$Rs - predictions) / test_data$Rs)) * 100

# Print metrics
cat("R-squared:", r_squared, "\n")
cat("MAPE:", mape, "\n")
```

```
# Apply fuzzy matching to match names for wickets
salary$Matched_Player <- sapply(salary$Player, match_names, names_list
=total_wicket_each_year$Bowler)

# Merge the DataFrames on the matched names
df_merged <- merge(salary,total_wicket_each_year, by.x = "Matched_Player", by.y = "Bowler")

# Independent and dependent variables in wickets
X <- df_merged$total_wickets
y <- df_merged$Rs

# Split the data into training and test sets (80% for training, 20% for testing)
set.seed(42)
train_indices <- sample(seq_len(nrow(df_merged)), size = 0.8 * nrow(df_merged))
train_data <- df_merged[train_indices, ]
test_data <- df_merged[-train_indices, ]

# OLS regression model
model <- lm(Rs ~ total_wickets, data = train_data)

# Summary of the regression model
summary(model)
```

```
# Using the broom package to get tidy output
tidy_model <- broom::tidy(model)
print(tidy_model)

# Predicting on the test set
predictions <- predict(model, newdata = test_data)

# Calculate R-squared and Mean Absolute Percentage Error
r_squared <- summary(model)$r.squared
mape <- mean(abs((test_data$Rs - predictions) / test_data$Rs)) * 100

# Print metrics
cat("R-squared:", r_squared, "\n")
cat("MAPE:", mape, "\n")
```

2a-NSSO Python

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
data = pd.read_csv("/content/NSSO68.csv")
#subset daa for GUJ
subset_data = data[data['state_1'] == 'GUJ'][['foodtotal_q', 'MPCE_MRP', 'MPCE_URP', 'Age',
'Meals_At_Home', 'Possess_ration_card', 'Education', 'No_of_Meals_per_day']]
print(subset_data.head())
# Check for missing values
print(subset_data.isnull().sum())
# Drop rows with any missing values
subset_data.dropna(inplace=True)
# Define the independent variables (X) and dependent variable (y)
X = subset_data[['MPCE_URP', 'Age', 'Meals_At_Home', 'Possess_ration_card', 'Education', 'foodtotal_q']]
y = subset_data['MPCE_MRP']
# Add constant to the features
X = sm.add_constant(X)

# Split data into training and test sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Fit the OLS model
model = sm.OLS(y_train, X_train).fit()

# Print model summary
```

```
print(model.summary())
# Check for multicollinearity using VIF
def calculate_vif(X):
    vif = pd.DataFrame()
    vif["Variable"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    return vif
# ipython-input-13-7d005dd0a72d
!pip install statsmodels
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Check for multicollinearity using VIF
def calculate_vif(X):
    vif = pd.DataFrame()
    vif["Variable"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    return vif
# Calculate VIF for independent variables
vif_data = X_train.drop(columns=['const'])  # Exclude the constant column
vif_scores = calculate_vif(vif_data)
print(vif_scores)

# Extract coefficients from the model
coefficients = model.params

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

# Print the equation
print(equation)
```

2b-NSSO R prgm

```
data=read.csv("/content/NSSO68.csv")
library(dplyr)
unique(data$state_1)
# Subset data to state Gujarat
subset_data <- data %>%
  filter(state_1 == 'GUJ') %>%
  select(foodtotal_q, MPCE_MRP, MPCE_URP,Age,Meals_At_Home,Possess_ration_card,Education,
No_of_Meals_per_day)
print(subset_data)
sum(is.na(subset_data$MPCE_MRP))
sum(is.na(subset_data$MPCE_URP))
```

```r
sum(is.na(subset_data$Age))
sum(is.na(subset_data$Possess_ration_card))
sum(is.na(data$Education))
impute_with_mean <- function(data, columns) {
  data %>%
    mutate(across(all_of(columns), ~ ifelse(is.na(.), mean(., na.rm = TRUE), .)))
}

# Columns to impute
columns_to_impute <- c("Education")

# Impute missing values with mean
data <- impute_with_mean(data, columns_to_impute)

sum(is.na(data$Education))
# Fit the regression model
model <- lm(MPCE_MRP ~ MPCE_URP + Age + Meals_At_Home + Possess_ration_card + Education +
foodtotal_q, data = subset_data)
summary(model)
install.packages("car")
library(car)
# Check for multicollinearity using Variance Inflation Factor (VIF)
vif(model) # VIF Value more than 8 its problematic
# Extract the coefficients from the model
coefficients <- coef(model)

# Construct the equation
equation <- paste0("y = ", round(coefficients[1], 2))
for (i in 2:length(coefficients)) {
  equation <- paste0(equation, " + ", round(coefficients[i], 6), "*x", i-1)
}
# Print the equation
print(equation)
print(head(subset_data$MPCE_MRP,1))
print(head(subset_data$MPCE_URP,1))
print(head(subset_data$Age,1))
print(head(subset_data$Meals_At_Home,1))
print(head(subset_data$Possess_ration_card,1))
print(head(subset_data$Education,1))
print(head(subset_data$foodtotal_q,1))
```

Recommendations

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