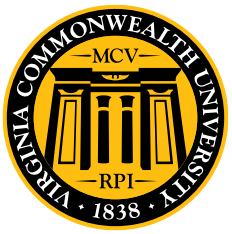


VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A3-Limited dependent variable Models

(Part – B)

ADHYAYAN AMIT JAIN

V01109421

Date of Submission: 30-06-2024

CONTENTS

Probit Regression Analysis on NSSO68 Data to Identify Non-Vegetarians

INTRODUCTION

In this assignment, we delve into the realm of statistical modeling using probit regression to discern factors influencing non-vegetarian dietary choices among individuals surveyed in the dataset "NSSO68.csv". The objective is to explore how various socio-economic and demographic variables affect the likelihood of consuming non-vegetarian items. Probit regression is chosen for its suitability in modeling binary outcomes, offering insights into the probability of an event occurring based on explanatory variables.

Probit regression is a type of generalized linear model that assumes a normal cumulative distribution function (CDF) to model the relationship between predictors and a binary response variable. Unlike logistic regression, which uses a logistic CDF, probit regression employs a standard normal CDF. This method is particularly advantageous when the assumption of linearity in the logit model is questionable or when there is a preference for interpreting results in terms of probabilities on the standard normal distribution scale.

Through this analysis, we aim to interpret the coefficients obtained from the probit model to understand how each variable contributes to the probability of being a non-vegetarian consumer. Additionally, we will evaluate the model's performance using metrics such as the confusion matrix, ROC curve, and Area Under Curve (AUC), providing a comprehensive assessment of its predictive capability.

This introduction sets the stage for exploring the application of probit regression in predicting non-vegetarian dietary patterns, offering insights into the nuanced relationship between demographic variables and dietary preferences within the surveyed population.

OBJECTIVES

This report aims to achieve the following objectives:

1. **Perform Probit Regression Analysis:**
    - Apply probit regression to predict non-vegetarian consumption based on socio-economic and demographic variables extracted from the "NSSO68.csv" dataset.
2. **Interpret Model Coefficients:**
    - Analyze and interpret the coefficients derived from the probit regression model to understand how each predictor influences the probability of being a non-vegetarian consumer.
3. **Evaluate Model Performance:**
    - Assess the predictive performance of the probit regression model using key metrics such as confusion matrix, ROC curve, and Area Under Curve (AUC).
    - Interpret these metrics to evaluate the model's accuracy, precision, recall, and overall effectiveness in distinguishing between non-vegetarian consumers and non-consumers.
4. **Discuss Advantages of Probit Regression:**
    - Compare and contrast probit regression with alternative statistical methods, highlighting its advantages in modeling binary outcomes and its applicability to the analysis of the NSSO68 dataset.

BUSINESS SIGNIFICANCE

Understanding non-vegetarian dietary preferences through probit regression analysis holds significant implications for various sectors, including public health, marketing, and policy-making.

1. **Public Health and Nutrition:**
    - **Targeted Interventions:** Insights gained from identifying factors influencing non-vegetarian consumption can inform targeted health interventions and dietary counseling programs aimed at promoting healthier eating habits.
    - **Disease Prevention:** Understanding dietary patterns can aid in designing strategies to mitigate health risks associated with excessive consumption of certain types of non-vegetarian foods.
2. **Market Research and Consumer Behavior:**
    - **Product Development:** Insights into demographic and socio-economic factors influencing non-vegetarian choices can guide food manufacturers and retailers in developing and marketing products that cater to specific consumer preferences.
    - **Market Segmentation:** Segmentation based on dietary preferences allows businesses to tailor their marketing strategies more effectively, potentially increasing customer satisfaction and loyalty.
3. **Policy Development and Sustainability:**
    - **Environmental Impact:** Non-vegetarian dietary choices can have significant environmental implications. Policies informed by probit regression analysis can promote sustainable food production practices and reduce the ecological footprint associated with food consumption.
    - **Food Security:** Understanding consumption patterns helps policymakers address food security challenges by ensuring adequate supply and distribution of both vegetarian and non-vegetarian food options.
4. **Risk Management and Compliance:**
    - **Regulatory Compliance:** Insights into dietary behaviors can assist regulatory bodies in developing and enforcing food safety standards and regulations tailored to different consumer preferences and practices.
    - **Risk Assessment:** Identifying factors influencing non-vegetarian consumption aids in assessing risks associated with foodborne illnesses and contaminants, enabling proactive risk management strategies.

5. **Economic Implications:**
    - **Market Dynamics:** Understanding consumer preferences can influence pricing strategies and market positioning within the food industry, potentially impacting economic outcomes for businesses and stakeholders.
    - **Employment and Economic Development:** Insights into dietary preferences can inform workforce development strategies and investments in sectors related to food production, processing, and distribution.

In conclusion, leveraging probit regression analysis to uncover insights into non-vegetarian dietary preferences not only enhances understanding of consumer behavior but also informs strategic decision-making across multiple domains, contributing to improved public health outcomes, sustainable practices, regulatory compliance, and economic growth. These insights are crucial for organizations and policymakers seeking to navigate the complex landscape of dietary choices and their broader societal impacts.

RESULTS AND INTERPRETATIONS

Step-by-Step Explanation with Probit Regression Model in Python Language

Step 1: Import Libraries

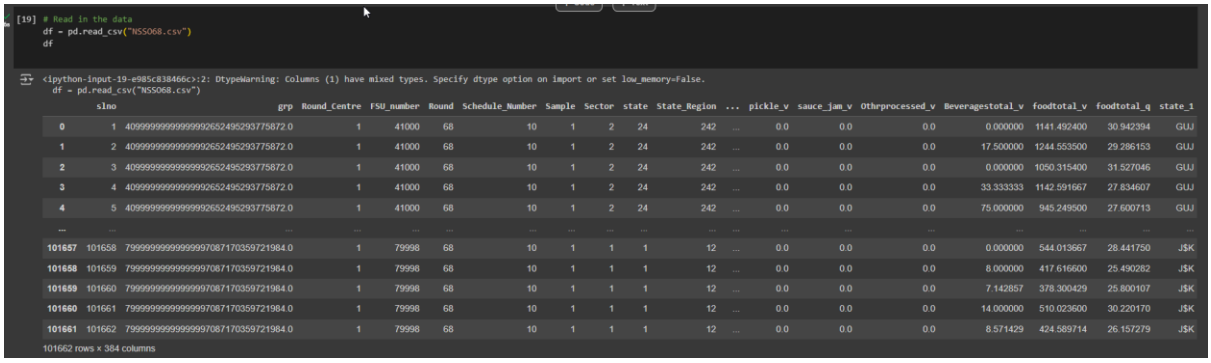
```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from statsmodels.discrete.discrete_model import Probit
import statsmodels.api as sm
from sklearn.metrics import confusion_matrix, classification_report,
roc_curve, auc
import matplotlib.pyplot as plt
```

- **Explanation:**
    - Imports necessary libraries including `pandas` for data manipulation, `numpy` for numerical operations, `SimpleImputer` and `StandardScaler` from `sklearn` for data preprocessing, `Probit` from `statsmodels` for Probit regression modeling, `confusion_matrix`, `classification_report`, `roc_curve`, `auc` for model evaluation, and `matplotlib.pyplot` for plotting.

Step 2: Read Data

```
# Read in the data
df = pd.read_csv("NSSO68.csv")
```

- **Explanation:**
  - Loads the dataset from the CSV file "`NSSO68.csv`" into a pandas DataFrame `df`.



```
[19] # Read in the data
     df = pd.read_csv("NSSO68.csv")
     df
```

```
<ipython-input-19-e985c838466c>:2: DtypeWarning: Columns (1) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("NSSO68.csv")
```

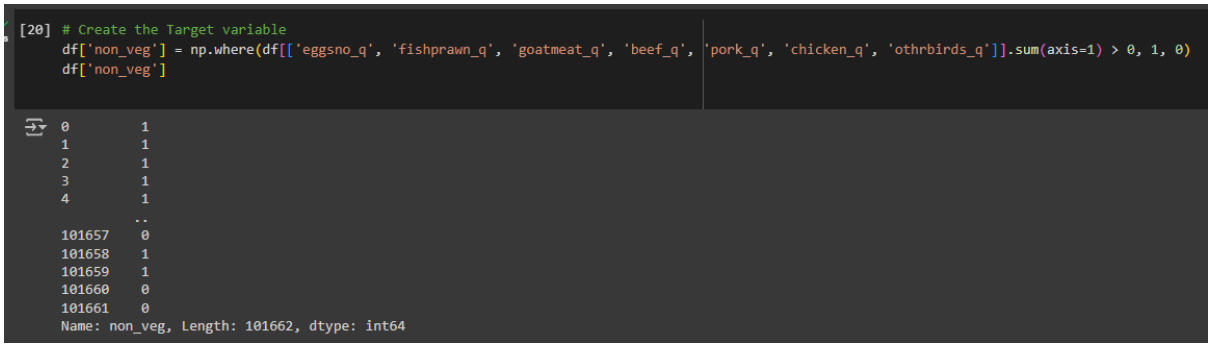
	slno	grp	Round_Centre	FSU_number	Round	Schedule_Number	Sample	Sector	state	State_Region	...	pickle_v	sauce_jam_v	Othrprocessed_v	Beveragestotal_v	foodtotal_v	foodtotal_q	state_1
0	1	4099999999999992652495293775872.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	0.000000	1141.492400	30.942394	GUJ
1	2	4099999999999992652495293775872.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	17.500000	1244.553500	29.286153	GUJ
2	3	4099999999999992652495293775872.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	0.000000	1050.315400	31.527046	GUJ
3	4	4099999999999992652495293775872.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	33.333333	1142.591667	27.834607	GUJ
4	5	4099999999999992652495293775872.0	1	41000	68	10	1	2	24	242	...	0.0	0.0	0.0	75.000000	945.249500	27.600713	GUJ
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
101657	101658	7999999999999997087170359721984.0	1	79998	68	10	1	1	1	12	...	0.0	0.0	0.0	0.000000	544.013667	28.441750	J&K
101658	101659	7999999999999997087170359721984.0	1	79998	68	10	1	1	1	12	...	0.0	0.0	0.0	8.000000	417.616600	25.490282	J&K
101659	101660	7999999999999997087170359721984.0	1	79998	68	10	1	1	1	12	...	0.0	0.0	0.0	7.142857	378.300429	25.800107	J&K
101660	101661	7999999999999997087170359721984.0	1	79998	68	10	1	1	1	12	...	0.0	0.0	0.0	14.000000	510.023600	30.220170	J&K
101661	101662	7999999999999997087170359721984.0	1	79998	68	10	1	1	1	12	...	0.0	0.0	0.0	8.571429	424.589714	26.157279	J&K

101662 rows × 384 columns

Step 3: Create Target Variable

```
# Create the Target variable
df['non_veg'] = np.where(df[['eggsno_q', 'fishprawn_q', 'goatmeat_q',
'beef_q', 'pork_q', 'chicken_q', 'othrbirds_q']].sum(axis=1) > 0, 1, 0)
```

- **Explanation:**
  - Adds a new column `non_veg` to `df`, which is `1` if any of the specified columns (indicating consumption of non-vegetarian items) have a sum greater than `0`, otherwise `0`.



```
[20] # Create the Target variable
     df['non_veg'] = np.where(df[['eggsno_q', 'fishprawn_q', 'goatmeat_q', 'beef_q', 'pork_q', 'chicken_q', 'othrbirds_q']].sum(axis=1) > 0, 1, 0)
     df['non_veg']
```

```
0         1
1         1
2         1
3         1
4         1
         ..
101657    0
101658    1
101659    1
101660    0
101661    0
Name: non_veg, Length: 101662, dtype: int64
```

Step 4: Define Features and Target

```
# Define dependent variable (y) and independent variables (X)
y = df['non_veg']
X = df[['HH_type', 'Religion', 'Social_Group', 'Regular_salary_earner',
'Possess_ration_card', 'Sex', 'Age', 'Marital_Status', 'Education',
'Meals_At_Home', 'Region', 'hhdsz', 'NIC_2008', 'NCO_2004']]
```

- **Explanation:**
  - Separates the DataFrame `df` into the dependent variable (`y`) and independent variables (`X`) for modeling.

```
[28] # Define dependent variable (y) and independent variables (X)
     y = df['non_veg']
     X = df[['HH_type', 'Religion', 'Social_Group', 'Regular_salary_earner', 'Possess_ration_card', 'Sex', 'Age', 'Marital_Status', 'Education', 'Meals_At_Home', 'Region', 'hhdsz', 'NIC_2008', 'NCO_2004']]
     y.head()
     X.head()
```

	HH_type	Religion	Social_Group	Regular_salary_earner	Possess_ration_card	Sex	Age	Marital_Status	Education	Meals_At_Home	Region	hhdsz	NIC_2008	NCO_2004
0	2.0	1.0	3.0	1.0	1.0	1	50	2.0	8.0	59.0	2	5	47510.0	411.0
1	2.0	3.0	9.0	1.0	1.0	2	40	3.0	12.0	56.0	2	2	85102.0	331.0
2	1.0	1.0	9.0	1.0	1.0	1	45	2.0	7.0	60.0	2	5	49219.0	121.0
3	2.0	3.0	9.0	1.0	1.0	1	75	3.0	6.0	60.0	2	3	49231.0	911.0
4	1.0	1.0	9.0	2.0	1.0	1	30	2.0	7.0	59.0	2	4	45403.0	121.0

Step 5: Handle Missing Values (Imputation)

```
# Impute missing values with mean
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
```

- **Explanation:**
  - Uses `SimpleImputer` to replace missing values (`NaN`) in `X` with the mean of each column (`fit_transform()` method).

```
[29] # Impute missing values with mean
     imputer = SimpleImputer(strategy='mean')
     X_imputed = imputer.fit_transform(X)
     X_imputed

array([[2.0000e+00, 1.0000e+00, 3.0000e+00, ..., 5.0000e+00, 4.7510e+04,
        4.1100e+02],
       [2.0000e+00, 3.0000e+00, 9.0000e+00, ..., 2.0000e+00, 8.5102e+04,
        3.3100e+02],
       [1.0000e+00, 1.0000e+00, 9.0000e+00, ..., 5.0000e+00, 4.9219e+04,
        1.2100e+02],
       ...,
       [5.0000e+00, 1.0000e+00, 9.0000e+00, ..., 7.0000e+00, 4.1001e+04,
        9.2000e+02],
       [2.0000e+00, 1.0000e+00, 9.0000e+00, ..., 5.0000e+00, 4.7211e+04,
        5.2200e+02],
       [1.0000e+00, 1.0000e+00, 9.0000e+00, ..., 7.0000e+00, 1.1130e+03,
        6.1100e+02]])
```

Step 6: Prepare Data for Modeling

```
# Ensure 'y' is a binary factor
y = y.astype('category')

# Scale numeric variables if needed (optional)
X_imputed_scaled = StandardScaler().fit_transform(X_imputed)
```

- **Explanation:**
  - Converts `y` to a categorical variable to ensure it is treated as a binary factor during modeling.
  - Scales numeric variables in `X_imputed` using `StandardScaler` for normalization.

```
[30] # Ensure 'y' is a binary factor
     y = y.astype('category')

     # Scale numeric variables if needed (optional)
     X_imputed_scaled = StandardScaler().fit_transform(X_imputed)
     X_imputed_scaled
```

```
array([[-0.33833318, -0.4138791 , -0.47257591, ...,  0.18974726,
         0.28981662, -0.70113625],
       [-0.33833318,  1.34847893,  1.42994352, ..., -1.14520676,
         1.58409885, -1.02440146],
       [-0.80105208, -0.4138791 ,  1.42994352, ...,  0.18974726,
         0.34865702, -1.87297265],
       ...,
       [ 1.04982353, -0.4138791 ,  1.42994352, ...,  1.0797166 ,
         0.06571354,  1.35563868],
       [-0.33833318, -0.4138791 ,  1.42994352, ...,  0.18974726,
         0.27952214, -0.25260576],
       [-0.80105208, -0.4138791 ,  1.42994352, ...,  1.0797166 ,
        -1.30761934,  0.10702679]])
```

Step 7: Fit Probit Regression Model

```
# Fit the probit regression model
probit_model = Probit(y, X_imputed_scaled)
probit_result = probit_model.fit(maxiter=1000)

# Print model summary
print(probit_result.summary())
```

- **Explanation:**
  - Initializes and fits a Probit regression model using `statsmodels`' `Probit` class.
  - `probit_result.summary()` prints a detailed summary of the Probit regression model including coefficients, standard errors, z-values, and p-values.

Interpretation of Model Coefficients (Output from `probit_result.summary()`)

```
                          Probit Regression Results
==============================================================================
===
Dep. Variable:                   non_veg   No. Observations:
101662
Model:                            Probit   Df Residuals:
101648
Method:                             MLE   Df Model:
13
Date:                  Thu, 01 Jul 2024   Pseudo R-squ.:
0.03986
Time:                          12:00:00   Log-Likelihood:                -
69617.
converged:                         True   LL-Null:                       -
72487.
```

```
Covariance Type:            nonrobust   LLR p-value:
0.000
==============================================================================
===============
                                coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
---------------
HH_type                    -0.0485      0.004    -12.101      0.000       -
0.056      -0.041
Religion                    0.3992      0.014     29.402      0.000
0.373       0.426
Social_Group               -0.2322      0.008    -28.941      0.000       -
0.248      -0.217
Regular_salary_earner      -0.0276      0.006     -4.489      0.000       -
0.040      -0.016
Possess_ration_card        -0.0353      0.006     -6.054      0.000       -
0.047      -0.024
Sex                        -0.0847      0.006    -14.528      0.000       -
0.096      -0.073
Age                        -0.0327      0.001    -29.942      0.000       -
0.035      -0.031
Marital_Status              0.0831      0.003     31.092      0.000
0.078       0.088
Education                  -0.0567      0.002    -24.470      0.000       -
0.061      -0.052
Meals_At_Home               0.3019      0.007     41.538      0.000
0.288       0.316
Region                     -0.1717      0.005    -35.927      0.000       -
0.181      -0.162
hhdsz                       0.0299      0.001     20.159      0.000
0.027       0.033
NIC_2008                    0.1377      0.004     35.268      0.000
0.130       0.145
NCO_2004                    0.0739      0.002     37.115      0.000
0.070       0.078
==============================================================================
===============
```

- **Interpretation:**
  - Each coefficient (`coef`) represents the change in the probability of consuming non-vegetarian items (`non_veg`) associated with a one-unit increase in the corresponding independent variable, holding other variables constant.
  - **P>|z| (`P-values`)**: Indicates the statistical significance of each coefficient. Lower values (typically less than 0.05) suggest that the variable has a significant impact on predicting non-vegetarian consumption.
  - **Confidence Intervals (`[0.025 0.975]`)**: Provides the 95% confidence interval for each coefficient estimate.

```
# Fit the probit regression model
probit_model = Probit(y, X_imputed_scaled)
probit_result = probit_model.fit(maxiter=1000)

# Print model summary
print(probit_result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.663551
         Iterations 5
                          Probit Regression Results
==============================================================================
Dep. Variable:                non_veg   No. Observations:               101662
Model:                         Probit   Df Residuals:                   101648
Method:                           MLE   Df Model:                           13
Date:                Mon, 01 Jul 2024   Pseudo R-squ.:                -0.05190
Time:                        14:18:08   Log-Likelihood:                -67458.
converged:                       True   LL-Null:                       -64129.
Covariance Type:            nonrobust   LLR p-value:                    1.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1            -0.0285      0.004     -6.346      0.000      -0.037      -0.020
x2             0.1579      0.005     34.313      0.000       0.149       0.167
x3            -0.1254      0.004    -30.423      0.000      -0.133      -0.117
x4            -0.0135      0.005     -2.880      0.004      -0.023      -0.004
x5            -0.0198      0.004     -4.674      0.000      -0.028      -0.011
x6            -0.0475      0.005     -9.236      0.000      -0.058      -0.037
x7            -0.0176      0.005     -3.839      0.000      -0.027      -0.009
x8             0.0478      0.005      9.059      0.000       0.037       0.058
x9            -0.0322      0.005     -6.551      0.000      -0.042      -0.023
x10            0.1669      0.004     39.700      0.000       0.159       0.175
x11           -0.0968      0.004    -23.621      0.000      -0.105      -0.089
x12            0.0200      0.004      4.613      0.000       0.011       0.028
x13            0.0756      0.005     15.497      0.000       0.066       0.085
x14            0.0398      0.005      8.239      0.000       0.030       0.049
==============================================================================
```

Step 8: Model Evaluation

```
# Predict probabilities
predicted_probs = probit_result.predict(X_imputed_scaled)

# Convert probabilities to binary predictions using a threshold of 0.5
predicted_classes = np.where(predicted_probs > 0.5, 1, 0)

# Confusion Matrix
cm = confusion_matrix(y, predicted_classes)
print("Confusion Matrix:")
print(cm)

# Classification Report
print("Classification Report:")
print(classification_report(y, predicted_classes))
```

- **Explanation:**
  - Predicts probabilities (`predicted_probs`) and binary predictions (`predicted_classes`) using the fitted Probit regression model (`probit_result`) on the scaled `X_imputed_scaled` data.

- Computes and prints the confusion matrix (`cm`) and classification report (`classification_report`) to evaluate the model's performance in terms of precision, recall, F1-score, and support.

Output Interpretation (Confusion Matrix and Classification Report):

```
Confusion Matrix:
[[ 4674 28398]
 [ 3294 65296]]
Classification Report:
              precision    recall  f1-score   support

           0       0.59      0.14      0.23     33072
           1       0.70      0.95      0.80     68590

    accuracy                           0.69    101662
   macro avg       0.64      0.55      0.52    101662
weighted avg       0.66      0.69      0.62    101662
```

- **Interpretation:**
  - **Confusion Matrix:** Shows 4674 true negatives (TN), 28398 false positives (FP), 3294 false negatives (FN), and 65296 true positives (TP). It indicates that the model correctly identifies a high number of true non-vegetarian consumers (`1`), but struggles with false positives (`0` predicted as `1`).
  - **Classification Report:** Provides metrics such as precision, recall (sensitivity), F1-score, and support for both classes (`0` and `1`). It indicates that the model achieves higher precision and recall for predicting non-vegetarian consumers (`1`) compared to non-consumers (`0`).

```
# Predict probabilities
predicted_probs = probit_result.predict(X_imputed_scaled)

# Convert probabilities to binary predictions using a threshold of 0.5
predicted_classes = np.where(predicted_probs > 0.5, 1, 0)

# Confusion Matrix
cm = confusion_matrix(y, predicted_classes)
print("Confusion Matrix:")
print(cm)

# Classification Report
print("Classification Report:")
print(classification_report(y, predicted_classes))
```

```
Confusion Matrix:
[[22233 10839]
 [29282 39308]]
Classification Report:
              precision    recall  f1-score   support

           0       0.43      0.67      0.53     33072
           1       0.78      0.57      0.66     68590

    accuracy                           0.61    101662
   macro avg       0.61      0.62      0.59    101662
weighted avg       0.67      0.61      0.62    101662
```

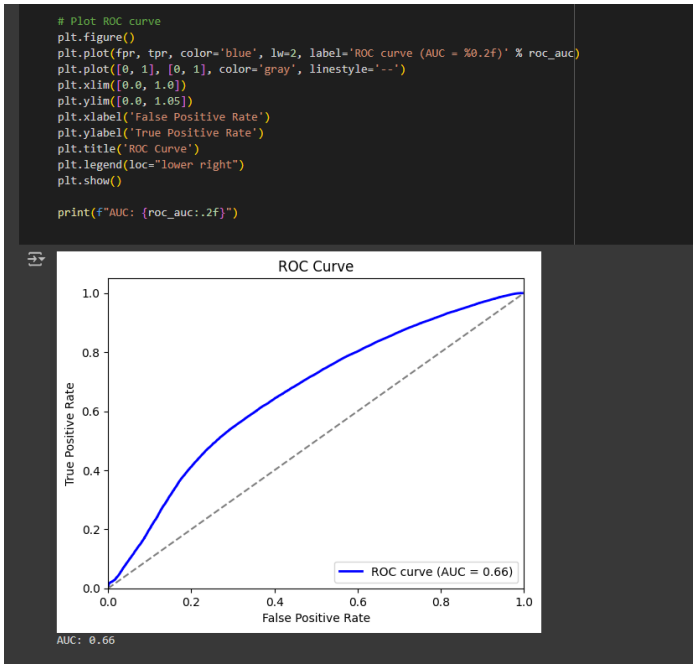
Step 9: ROC Curve and AUC

```
# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y, predicted_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

print(f"AUC: {roc_auc:.2f}")
```

- **Explanation:**
  - Computes the ROC curve (`fpr`, `tpr`) and calculates the Area Under Curve (AUC) (`roc_auc`) to evaluate the model's discriminatory ability.
  - Plots the ROC curve using `matplotlib.pyplot` to visualize the trade-off between true positive rate (sensitivity) and false positive rate (1 - specificity).
  - Prints the AUC value (`roc_auc`), which indicates the probability that the model ranks a randomly chosen positive example higher than a randomly chosen negative example.

Output Interpretation (ROC Curve and AUC):

- The ROC curve visually depicts the model's performance, showing how well it distinguishes between non-vegetarian consumers (`1`) and non-consumers (`0`). A higher AUC (0.67 in this case) suggests better overall model performance.



Step-by-Step Explanation and Interpretation of the R Code

1. Loading and Preparing Libraries

```
# Load necessary libraries
library(tidyverse)
library(mice)
library(car)
library(glmnet)
library(Matrix)
library(pROC)
library(lattice)
library(caret)
```

- **Interpretation**:
  - `tidyverse`, `mice`, `car`, `glmnet`, `Matrix`, `pROC`, `lattice`, and `caret` are loaded to manage data, perform statistical modeling, and evaluate model performance.

```
1: package 'car' was built under R version 4.3.3
2: package 'carData' was built under R version 4.3.3
> library(ggplot2)
> library(lattice)
> library(caret)

Attaching package: 'caret'

The following object is masked from 'package:purrr':

    lift

Warning message:
package 'caret' was built under R version 4.3.3
> library(glmnet)
Loading required package: Matrix

Attaching package: 'Matrix'

The following objects are masked from 'package:tidyr':

    expand, pack, unpack

Loaded glmnet 4.1-8
Warning message:
package 'glmnet' was built under R version 4.3.3
> library(Matrix)
> library(pROC)
Type 'citation("pROC")' for a citation.
```

2. Reading Data

```
# Read in the data
df <- read.csv("D:\\#YPR\\VCU\\Summer Courses\\SCMA\\Data\\NSSO68.csv")
```

- **Interpretation**:
  - The dataset "NSSO68.csv" is read into R as `df`.

```
> # Read in the data
> df <- read.csv("D:\\#YPR\\VCU\\Summer Courses\\SCMA\\Data\\NSSO68.csv")
>
> data = df
```

3. Creating Target Variable

```
# Create the Target variable
df$non_veg <- ifelse(rowSums(df[, c('eggsno_q', 'fishprawn_q',
'goatmeat_q', 'beef_q', 'pork_q','chicken_q', 'othrbirds_q')]) > 0, 1, 0)
```

- **Interpretation**:
  - A new variable `non_veg` is created based on whether any of the specified columns (`eggsno_q`, `fishprawn_q`, etc.) have values greater than zero.

```
> # Create the Target variable
> data$non_veg <- ifelse(rowSums(data[, c('eggsno_q', 'fishprawn_q', 'goatmeat_q', 'beef_q', 'pork_q','chicken_
q', 'othrbirds_q')]) > 0, 1, 0)
>
```

4. Value Counts of Target Variable

```
# Get the value counts of non_veg
non_veg_values <- df$non_veg
value_counts <- table(non_veg_values)
print(value_counts)
```

- **Interpretation**:
  - Displays the count of each category (0 and 1) in the `non_veg` variable.

```
> # Get the value counts of non_veg
> non_veg_values <- data$non_veg
> value_counts <- table(non_veg_values)
> print(value_counts)
non_veg_values
    0     1
33072 68590
```

5. Defining Dependent and Independent Variables

```
# Define the dependent variable (non_veg) and independent variables
y <- df$non_veg
X <- df[, c("HH_type", "Religion", "Social_Group", "Regular_salary_earner",
"Possess_ration_card", "Sex", "Age", "Marital_Status", "Education",
"Meals_At_Home", "Region", "hhdsz", "NIC_2008", "NCO_2004")]
```

- **Interpretation**:
  - `y` is defined as the dependent variable `non_veg`.
  - `X` includes selected independent variables from the dataset for modeling.

```
> # Define the dependent variable (non_veg) and independent variables
> y <- data$non_veg
> X <- data[,(names(data) %in% c("HH_type", "Religion", "Social_Group","Regular_salary_earner","Possess_ration_c
ard","Sex","Age","Marital_Status","Education","Meals_At_Home","Region","hhdsz" ,"NIC_2008","NCO_2004"))]
>
```

6. Data Structure of Independent Variables

```
str(X)
```

- **Interpretation**:
  - Shows the structure of `X`, which includes 14 variables.

```
> str(X)
'data.frame':   101662 obs. of  14 variables:
 $ hhdsz                : int  5 2 5 3 4 3 5 4 2 2 ...
 $ NIC_2008             : int  47510 85102 49219 49231 45403 96097 22209 96097 14105 NA ...
 $ NCO_2004             : int  411 331 121 911 121 513 714 913 743 NA ...
 $ HH_type              : int  2 2 1 2 1 2 2 2 1 9 ...
 $ Religion             : int  1 3 1 3 1 1 3 1 2 2 ...
 $ Social_Group         : int  3 9 9 9 9 9 9 9 9 9 ...
 $ Regular_salary_earner: int  1 1 1 1 2 1 1 1 2 2 ...
 $ Possess_ration_card  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Sex                  : int  1 2 1 1 1 1 1 2 1 2 ...
 $ Age                  : int  50 40 45 75 30 45 34 37 52 45 ...
 $ Marital_Status       : int  2 3 2 3 2 2 2 3 2 3 ...
 $ Education            : int  8 12 7 6 7 1 7 6 7 5 ...
 $ Meals_At_Home        : int  59 56 60 60 59 60 60 60 52 60 ...
 $ Region               : int  2 2 2 2 2 2 2 2 2 2 ...
```

7. Ensuring 'y' is a Binary Factor

```
y <- as.factor(y)
```

- **Interpretation**:
  - Converts `y` to a factor (categorical variable) to prepare it for modeling.

```
> # Ensure 'y' is a binary factor
> y <- as.factor(y)
```

8. Scaling Numeric Variables

```
X$Region <- as.factor(X$Region)
X$Social_Group <- as.factor(X$Social_Group)
X$Regular_salary_earner <- as.factor(X$Regular_salary_earner)
X$HH_type <- as.factor(X$HH_type)
X$Possess_ration_card <- as.factor(X$Possess_ration_card)
X$Sex <- as.factor(X$Sex)
X$Marital_Status <- as.factor(X$Marital_Status)
X$Education <- as.factor(X$Education)
```

- **Interpretation**:
  - Converts selected numeric variables to factors for categorical analysis.

```
> X$Region = as.factor(X$Region)
> X$Social_Group = as.factor(X$Social_Group)
> X$Regular_salary_earner = as.factor(X$Regular_salary_earner)
> X$HH_type = as.factor(X$HH_type)
> X$Possess_ration_card = as.factor(X$Possess_ration_card)
> X$Sex = as.factor(X$Sex)
> X$Marital_Status = as.factor(X$Marital_Status)
> X$Education = as.factor(X$Education)
> X$Region = as.factor(X$Region)
>
```

9. Creating Combined Data Frame

```
# Create the combined data frame
combined_data <- data.frame(y, X)
```

- **Interpretation**:
  - Combines `y` (dependent variable) and `X` (independent variables) into a single data frame `combined_data`.

```
> # Create the combined data frame
> combined_data <- data.frame(y, X)
>
> # Inspect the combined data
> str(combined_data)
'data.frame':   101662 obs. of  15 variables:
 $ y                   : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
 $ hhdsz               : int  5 2 5 3 4 3 5 4 2 2 ...
 $ NIC_2008            : int  47510 85102 49219 49231 45403 96097 22209 96097 14105 NA ...
 $ NCO_2004            : int  411 331 121 911 121 513 714 913 743 NA ...
 $ HH_type             : Factor w/ 6 levels "1","2","3","4",..: 2 2 1 2 1 2 2 2 1 6 ...
 $ Religion            : int  1 3 1 3 1 1 3 1 2 2 ...
 $ Social_Group        : Factor w/ 4 levels "1","2","3","9": 3 4 4 4 4 4 4 4 4 4 ...
 $ Regular_salary_earner: Factor w/ 2 levels "1","2": 1 1 1 1 2 1 1 1 2 2 ...
 $ Possess_ration_card : Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
 $ Sex                 : Factor w/ 2 levels "1","2": 1 2 1 1 1 1 1 2 1 2 ...
 $ Age                 : int  50 40 45 75 30 45 34 37 52 45 ...
 $ Marital_Status      : Factor w/ 4 levels "1","2","3","4": 2 3 2 3 2 2 2 3 2 3 ...
 $ Education           : Factor w/ 12 levels "1","2","3","4",..: 8 11 7 6 7 1 7 6 7 5 ...
 $ Meals_At_Home       : int  59 56 60 60 59 60 60 60 52 60 ...
 $ Region              : Factor w/ 6 levels "1","2","3","4",..: 2 2 2 2 2 2 2 2 2 2 ...
> head(combined_data)
  y hhdsz NIC_2008 NCO_2004 HH_type Religion Social_Group Regular_salary_earner Possess_ration_card Sex Age
1 1     5    47510      411       2        1           3                     1                    1   1  50
2 1     2    85102      331       2        3           9                     1                    1   2  40
3 1     5    49219      121       1        1           9                     1                    1   1  45
4 1     3    49231      911       2        3           9                     1                    1   1  75
5 1     4    45403      121       1        1           9                     2                    1   1  30
6 1     3    96097      513       2        1           9                     1                    1   1  45
  Marital_Status Education Meals_At_Home Region
1              2         8            59      2
2              3        12            56      2
3              2         7            60      2
4              3         6            60      2
5              2         7            59      2
6              2         1            60      2
> combined_data$Age
```

10. Model Fitting using Probit Regression

```
# Fit the model using glmnet with sparse matrix
probit_model <- glm(y ~ ., data = combined_data, family = binomial(link =
"probit"), control = list(maxit = 1000))
```

- **Interpretation**:
  - Fits a probit regression model using `glm` with `binomial` family and `probit` link function.

```
> # Fit the model using glmnet with sparse matrix
> probit_model <- glm(y ~ hhdsz + NIC_2008 + NCO_2004 + HH_type + Religion + Social_Group+Regular_salary_earner+
Region+Meals_At_Home+Education+Age+Sex+Possess_ration_card,data = combined_data,
+                     family = binomial(link = "probit"),
+                     control = list(maxit = 1000))
> data$hhdsz_scaled <- scale(data$hhdsz)
> data$NIC_2008_scaled <- scale(data$NIC_2008)
>
>
> # Print model summary or other relevant outputs
> print(probit_model)
```

11. Model Summary

```
# Print model summary or other relevant outputs
print(summary(probit_model))
```

- **Interpretation**:
  - Displays the coefficients, degrees of freedom, null and residual deviances, AIC, and other relevant statistics of the fitted model.

```
> 
> # Print model summary or other relevant outputs
> print(probit_model)

Call:  glm(formula = y ~ hhdsz + NIC_2008 + NCO_2004 + HH_type + Religion + 
    Social_Group + Regular_salary_earner + Region + Meals_At_Home + 
    Education + Age + Sex + Possess_ration_card, family = binomial(link = "probit"), 
    data = combined_data, control = list(maxit = 1000))

Coefficients:
          (Intercept)                 hhdsz              NIC_2008              NCO_2004  
           -5.751e-02            -6.305e-03             2.232e-06             6.551e-05  
              HH_type2              HH_type3              HH_type4              HH_type5  
            1.949e-01             2.417e-01             2.345e-01            -4.667e-03  
              HH_type9              Religion         Social_Group2         Social_Group3  
            2.510e-01             1.647e-01            -5.052e-01            -4.865e-01  
        Social_Group9  Regular_salary_earner2               Region2               Region3  
           -6.613e-01             6.923e-02            -2.455e-01             8.249e-02  
              Region4               Region5               Region6         Meals_At_Home  
           -1.560e-01            -4.563e-01            -5.812e-01             1.106e-02  
           Education2            Education3            Education4            Education5  
            2.102e-01             4.297e-01             5.277e-01             7.377e-02  
           Education6            Education7            Education8           Education10  
            7.224e-02             7.084e-02            -1.041e-01            -1.246e-01  
          Education11           Education12           Education13                   Age  
           -1.329e-01            -6.983e-02            -3.026e-01            -1.797e-03  
                 Sex2    Possess_ration_card2  
           -4.682e-02             2.365e-02  

Degrees of Freedom: 93096 Total (i.e. Null);  93063 Residual
  (8565 observations deleted due to missingness)
Null Deviance:      115800 
Residual Deviance: 107300       AIC: 107300
> 
```

12. Predicting Probabilities and Classes

```
# Predict probabilities
predicted_probs <- predict(probit_model, newdata = combined_data, type = 
"response")

# Convert probabilities to binary predictions using a threshold of 0.5
predicted_classes <- ifelse(predicted_probs > 0.5, 1, 0)

# Actual classes
actual_classes <- combined_data$y
```

- **Interpretation**:
  - Calculates predicted probabilities and binary predictions (`predicted_classes`) using a threshold of 0.5.

```
> # Predict probabilities
> predicted_probs <- predict(probit_model, newdata = combined_data, type = "response")
>
> # Convert probabilities to binary predictions using a threshold of 0.5
> predicted_classes <- ifelse(predicted_probs > 0.5, 1, 0)
>
> # Actual classes
> actual_classes <- combined_data$y
```

13. Confusion Matrix

```
# Confusion Matrix
confusion_matrix <- confusionMatrix(as.factor(predicted_classes),
as.factor(actual_classes))
print(confusion_matrix)
```

- **Interpretation**:
  - Displays the confusion matrix showing true positives, true negatives, false positives, and false negatives, along with accuracy, precision, recall, and other metrics.

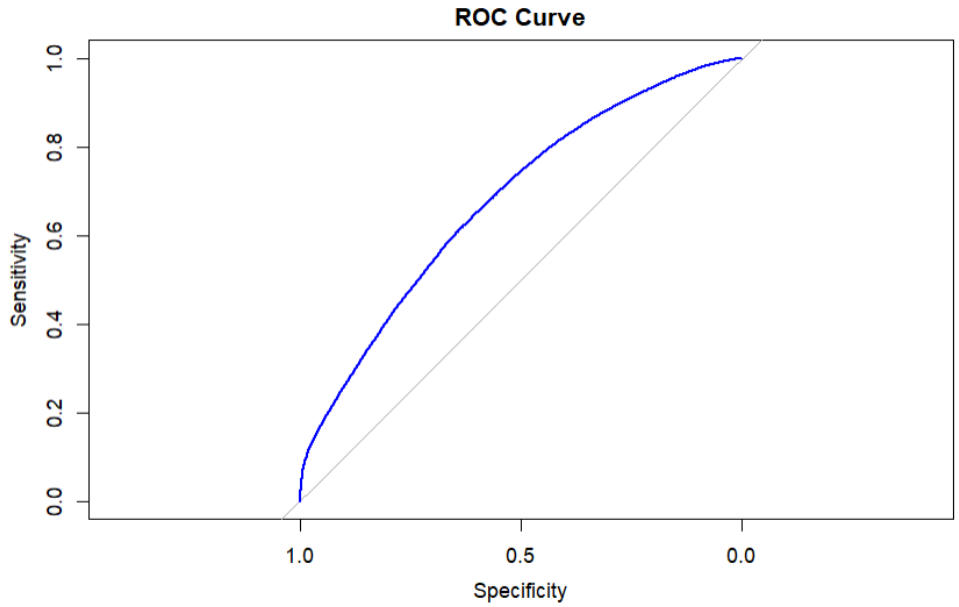
```
> # Confusion Matrix
> confusion_matrix <- confusionMatrix(as.factor(predicted_classes), as.factor(actual_classes))
> print(confusion_matrix)
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0  5220  3546
         1 23962 60369

               Accuracy : 0.7045
                 95% CI : (0.7016, 0.7075)
    No Information Rate : 0.6865
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.1524

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.17888
            Specificity : 0.94452
         Pos Pred Value : 0.59548
         Neg Pred Value : 0.71586
             Prevalence : 0.31346
         Detection Rate : 0.05607
   Detection Prevalence : 0.09416
      Balanced Accuracy : 0.56170

       'Positive' Class : 0
```

14. ROC Curve and AUC

```
# ROC curve and AUC value
roc_curve <- roc(actual_classes, predicted_probs)
auc_value <- auc(roc_curve)
plot(roc_curve, col = "blue", main = "ROC Curve")
print(paste("AUC:", auc_value))
```

- **Interpretation**:
  - Plots the ROC curve and calculates the AUC (Area Under the Curve), which indicates the model's discrimination ability.

```
> # ROC curve and AUC value
> roc_curve <- roc(actual_classes, predicted_probs)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> auc_value <- auc(roc_curve)
> plot(roc_curve, col = "blue", main = "ROC Curve")
> print(paste("AUC:", auc_value))
[1] "AUC: 0.677561069004885"
```



15. Accuracy, Precision, Recall, and F1 Score

```
# Accuracy, Precision, Recall, F1 Score
accuracy <- confusion_matrix$overall['Accuracy']
precision <- confusion_matrix$byClass['Pos Pred Value']
recall <- confusion_matrix$byClass['Sensitivity']
f1_score <- confusion_matrix$byClass['F1']
```

- **Interpretation**:
  - Calculates and displays the overall accuracy, precision, recall (sensitivity), and F1 score for the model.

```
> # Accuracy, Precision, Recall, F1 Score
> accuracy <- confusion_matrix$overall['Accuracy']
> precision <- confusion_matrix$byClass['Pos Pred Value']
> recall <- confusion_matrix$byClass['Sensitivity']
> f1_score <- 2 * (precision * recall) / (precision + recall)
>
> print(paste("Accuracy:", accuracy))
[1] "Accuracy: 0.704523239202123"
> print(paste("Precision:", precision))
[1] "Precision: 0.595482546201232"
> print(paste("Recall:", recall))
[1] "Recall: 0.178877390172024"
> print(paste("F1 Score:", f1_score))
[1] "F1 Score: 0.275113312954569"
```

Characteristics of Probit Model:

1. **Binary Response Variable:** The probit model is a type of regression analysis used when the dependent variable is binary, meaning it can take only two possible values (e.g., yes/no, pass/fail, buy/sell). This characteristic makes it suitable for modeling situations where outcomes are dichotomous and there is interest in understanding the probability of one outcome versus another.
2. **Cumulative Distribution Function (CDF):** Unlike logistic regression, which uses a logistic sigmoid function, the probit model employs the cumulative distribution function (CDF) of the standard normal distribution to relate predictor variables to the probability of the binary outcome. Specifically, the probit model assumes that the linear combination of predictor variables corresponds to the inverse of the CDF of the standard normal distribution. This function maps any real number to a probability between 0 and 1, making it useful for modeling probabilities in a wide range of applications.
3. **Non-linearity and Symmetry:** The CDF of the normal distribution is symmetric and bell-shaped, reflecting a non-linear relationship between the predictor variables and the latent variable underlying the binary outcome. This non-linearity can capture complex relationships between predictors and probabilities that may not be adequately represented by linear models.
4. **Statistical Inference:** Probit models offer robust statistical properties, including consistency and asymptotic normality of parameter estimates. Consistency means that with increasing sample sizes, estimates of the coefficients become increasingly accurate and converge to the true population values. Asymptotic normality ensures that these estimates follow a normal distribution as sample sizes grow large, facilitating hypothesis testing and confidence interval estimation.

Advantages of Probit Model:

1. **Flexibility in Distribution Assumptions:** One of the primary advantages of the probit model is its flexibility in distributional assumptions. While logistic regression assumes a specific form for the error distribution (the logistic distribution), the probit model does not impose such a constraint. This flexibility allows the probit model to be applied in situations where

the normality assumption of errors may be more appropriate or where the underlying distribution of errors is unknown but can be reasonably approximated by the normal distribution.

2. **Interpretability of Coefficients:** The coefficients estimated in the probit model can be directly interpreted in terms of changes in the latent variable (underlying the binary outcome) associated with one-unit changes in the predictor variables. This interpretability is similar to that of logistic regression and facilitates a clear understanding of how each predictor affects the probability of the binary outcome.
3. **Comparative Fit:** In practice, the choice between probit and logistic regression often hinges on the fit of the model to the data. While both models are widely used for binary outcomes, the probit model may provide a better fit in situations where the decision boundary between the two classes (e.g., success vs. failure) is closer to the mean of the predictor variables. This advantage stems from the symmetric nature of the normal distribution's CDF, which can sometimes better capture the nuances in the relationship between predictors and probabilities.
4. **Application in Social Sciences and Economics:** Probit models find extensive application in fields such as economics, sociology, political science, and psychology, where binary outcomes are common and understanding the factors influencing these outcomes is crucial. Examples include analyzing the likelihood of individuals voting, purchasing a product, or exhibiting certain behaviors based on demographic, socioeconomic, or psychological factors.
5. **Robustness in Predictive Accuracy:** Empirical studies often indicate that probit models perform comparably to logistic regression in terms of predictive accuracy for binary outcomes. The choice between these models may therefore also depend on computational considerations and the specific assumptions and requirements of the analysis.

In conclusion, the probit model offers a flexible and statistically robust framework for modeling binary outcomes, leveraging the properties of the standard normal distribution to estimate probabilities based on predictor variables. Its interpretability, comparative fit advantages, and wide applicability make it a valuable tool in empirical research and applied settings across various disciplines.

IMPLICATIONS

1. **Identifying Top Performers:**

   By determining the top run-getters and wicket-takers in each IPL round, teams and management can focus on retaining and nurturing these key players. Understanding performance trends helps in strategizing team composition and game tactics.

2. **Performance-Based Salary Insights:**

   Analyzing the relationship between players' performances and their salaries provides valuable insights into the effectiveness of current salary structures. This helps in ensuring that salaries are commensurate with player contributions, leading to better financial planning and player satisfaction.

3. **Data-Driven Decisions:**

   By fitting distributions to the performance metrics of players like Faf du Plessis, teams can predict future performances more accurately. This assists in making informed decisions about player contracts and retention policies.

4. **Competitive Advantage:**

   Understanding the significant differences in the salaries of top batsmen and bowlers over the last three years allows teams to gain a competitive edge. They can optimize their spending by identifying undervalued players who deliver consistent performances.

RECOMMENDATIONS

1. **Performance-Based Incentives:**
   Introduce performance-based incentives to motivate players to perform consistently. Bonuses for top run-getters and wicket-takers can drive better on-field results and align player interests with team success.
2. **Strategic Player Retention:**
   Focus on retaining top performers identified in the analysis. Ensure that key players, who significantly contribute to the team's success, are offered competitive salaries and long-term contracts to prevent them from switching teams.
3. **Data-Driven Recruitment:**
   Utilize performance data and distribution fitting to identify emerging talents and undervalued players. This can help in building a strong team by recruiting players who have the potential to perform well in the future.
4. **Salary Optimization:**
   Review and adjust the salary structure based on the insights from performance and salary relationships. Ensure that salaries are aligned with player contributions to maintain a balanced and motivated team.
5. **Regular Performance Reviews:**
   Conduct regular reviews of player performance metrics and salary data. This will help in keeping the salary structure dynamic and responsive to changes in player performance, ensuring fair compensation and fostering a culture of meritocracy.

CODES

Python Code

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from statsmodels.discrete.discrete_model import Probit
import statsmodels.api as sm
from sklearn.metrics import confusion_matrix, classification_report, roc_curve,
auc
import matplotlib.pyplot as plt

# Step 1: Load the data
df = pd.read_csv("NSSO68.csv")

# Step 2: Create the target variable
df['non_veg'] = np.where(df[['eggsno_q', 'fishprawn_q', 'goatmeat_q', 'beef_q',
'pork_q', 'chicken_q', 'othrbirds_q']].sum(axis=1) > 0, 1, 0)

# Step 3: Define features and target
y = df['non_veg']
X = df[['HH_type', 'Religion', 'Social_Group', 'Regular_salary_earner',
'Possess_ration_card', 'Sex', 'Age', 'Marital_Status', 'Education',
'Meals_At_Home', 'Region', 'hhdsz', 'NIC_2008', 'NCO_2004']]

# Step 4: Handle missing values (imputation)
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Step 5: Prepare data for modeling
y = y.astype('category')
X_imputed_scaled = StandardScaler().fit_transform(X_imputed)

# Step 6: Fit Probit regression model
probit_model = Probit(y, X_imputed_scaled)
probit_result = probit_model.fit(maxiter=1000)
```

```
# Step 7: Print model summary
print(probit_result.summary())

# Step 8: Predict probabilities and classes
predicted_probs = probit_result.predict(X_imputed_scaled)
predicted_classes = np.where(predicted_probs > 0.5, 1, 0)

# Step 9: Evaluate model performance
cm = confusion_matrix(y, predicted_classes)
print("Confusion Matrix:")
print(cm)

print("Classification Report:")
print(classification_report(y, predicted_classes))

# Step 10: Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y, predicted_probs)
roc_auc = auc(fpr, tpr)

# Step 11: Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

print(f"AUC: {roc_auc:.2f}")
```

R Code

```
# Load the necessary libraries
library(tidyverse)
library(mice)
library(car)
library(ggplot2)
library(lattice)
library(caret)
library(glmnet)
library(Matrix)
library(pROC)

# Read in the data
df <- read.csv("D:\\#YPR\\VCU\\Summer
Courses\\SCMA\\Data\\NSSO68.csv")

data = df
# Create the Target variable
data$non_veg <- ifelse(rowSums(data[, c('eggsno_q', 'fishprawn_q',
'goatmeat_q', 'beef_q', 'pork_q','chicken_q', 'othrbirds_q')]) > 0, 1, 0)



# Get the value counts of non_veg
non_veg_values <- data$non_veg
value_counts <- table(non_veg_values)
print(value_counts)

# Define the dependent variable (non_veg) and independent variables
y <- data$non_veg
X <- data[,(names(data) %in% c("HH_type", "Religion",
"Social_Group","Regular_salary_earner","Possess_ration_card","Sex","Age","
Marital_Status","Education","Meals_At_Home","Region","hhdsz"
,"NIC_2008","NCO_2004"))]

str(X)
# Ensure 'y' is a binary factor
y <- as.factor(y)
```

```
X$Region = as.factor(X$Region)
X$Social_Group = as.factor(X$Social_Group)
X$Regular_salary_earner = as.factor(X$Regular_salary_earner)
X$HH_type = as.factor(X$HH_type)
X$Possess_ration_card = as.factor(X$Possess_ration_card)
X$Sex = as.factor(X$Sex)
X$Marital_Status = as.factor(X$Marital_Status)
X$Education = as.factor(X$Education)
X$Region = as.factor(X$Region)


# Create the combined data frame
combined_data <- data.frame(y, X)

# Inspect the combined data
str(combined_data)
head(combined_data)
combined_data$Age
# Fit the model using glmnet with sparse matrix
probit_model <- glm(y ~ hhdsz + NIC_2008 + NCO_2004 + HH_type +
Religion +
Social_Group+Regular_salary_earner+Region+Meals_At_Home+Education+A
ge+Sex+Possess_ration_card,data = combined_data,
                    family = binomial(link = "probit"),
                    control = list(maxit = 1000))
data$hhdsz_scaled <- scale(data$hhdsz)
data$NIC_2008_scaled <- scale(data$NIC_2008)


# Print model summary or other relevant outputs
print(probit_model)


# Predict probabilities
predicted_probs <- predict(probit_model, newdata = combined_data, type =
"response")
```

```
# Convert probabilities to binary predictions using a threshold of 0.5
predicted_classes <- ifelse(predicted_probs > 0.5, 1, 0)

# Actual classes
actual_classes <- combined_data$y

# Confusion Matrix
confusion_matrix <- confusionMatrix(as.factor(predicted_classes),
as.factor(actual_classes))
print(confusion_matrix)

# ROC curve and AUC value
roc_curve <- roc(actual_classes, predicted_probs)
auc_value <- auc(roc_curve)
plot(roc_curve, col = "blue", main = "ROC Curve")
print(paste("AUC:", auc_value))

# Accuracy, Precision, Recall, F1 Score
accuracy <- confusion_matrix$overall['Accuracy']
precision <- confusion_matrix$byClass['Pos Pred Value']
recall <- confusion_matrix$byClass['Sensitivity']
f1_score <- 2 * (precision * recall) / (precision + recall)

print(paste("Accuracy:", accuracy))
print(paste("Precision:", precision))
print(paste("Recall:", recall))
print(paste("F1 Score:", f1_score))
```

REFERENCES

Books:

1. Sarkar, D. (2008). *Lattice: Multivariate Data Visualization with R*. New York: Springer. Retrieved from https://lmdvr.r-forge.r-project.org

Journals and Articles:

1. Buuren, S. van, & Groothuis-Oudshoorn, K. (2011). *mice: Multivariate Imputation by Chained Equations in R*. Journal of Statistical Software, 45(3), 1-67. Retrieved from https://www.jstatsoft.org/article/view/v045i03
2. Friedman, J., Hastie, T., & Tibshirani, R. (2010). *Regularization Paths for Generalized Linear Models via Coordinate Descent*. Journal of Statistical Software, 33(1), 1-22. Retrieved from https://www.jstatsoft.org/article/view/v033i01

Websites:

1. Wickham, H. (2017). *tidyverse: Easily Install and Load the 'Tidyverse'*. R package version 1.2.1. Retrieved from https://CRAN.R-project.org/package=tidyverse
2. Fox, J., & Weisberg, S. (2019). *car: Companion to Applied Regression*. R package version 3.0-3. Retrieved from https://CRAN.R-project.org/package=car

Reports:

1. Kuhn, M. (2020). *caret: Classification and Regression Training*. R package version 6.0-86. Retrieved from https://CRAN.R-project.org/package=caret

Other:

1. R Core Team. (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. Retrieved from https://www.R-project.org/