Data Analytics-II

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score

\_

df= pd.read\_csv("Assignment-A5-Social\_Network\_Ads.csv")

df.head() # Print first 5 rows

\_

df.columns

\_

# Convert Gender to numeric

df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})

# Features and Target

X = df[['Gender', 'Age', 'EstimatedSalary']]

y = df['Purchased']

# Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

\_

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

\_

# Train the model

classifier = LogisticRegression()

classifier.fit(X\_train, y\_train)

# Make predictions

y\_pred = classifier.predict(X\_test)

\_

# Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", cm)

# Extract values

TN, FP, FN, TP = cm.ravel()

# Metrics

accuracy = accuracy\_score(y\_test, y\_pred)

error\_rate = 1 - accuracy

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

print(f"True Positives (TP): {TP}")

print(f"False Positives (FP): {FP}")

print(f"True Negatives (TN): {TN}")

print(f"False Negatives (FN): {FN}")

print(f"Accuracy: {accuracy:.2f}")

print(f"Error Rate: {error\_rate:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

\_

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

**Code Explanation:**Alright! Let's explain your code in **simple, easy language** step-by-step:

**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

🔹 **Why this?**

* We are **importing libraries** needed for the project:
  + pandas → to **load and handle dataset** (tables, data).
  + numpy → for **numerical calculations**.
  + matplotlib.pyplot and seaborn → for **creating graphs and charts**.

from sklearn.model\_selection import train\_test\_split

🔹 **Why this?**

* To **split** the data into two parts:
  + **Training data** (to train the model)
  + **Testing data** (to check if model is good)

from sklearn.preprocessing import StandardScaler

🔹 **Why this?**

* To **scale** (adjust) the values in dataset.
* Scaling is needed because **some columns have very large numbers** and **some have small numbers**.
* Scaling makes all numbers **similar in size** (very important for logistic regression).

from sklearn.linear\_model import LogisticRegression

🔹 **Why this?**

* We are **importing Logistic Regression model**.
* This model is used for **classification** (deciding between two classes, like 'Yes' or 'No').

from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score

🔹 **Why this?**

* After building the model, we need to **check how good it is**.
* These functions help us measure:
  + **Confusion Matrix** (TP, TN, FP, FN)
  + **Accuracy**
  + **Precision**
  + **Recall**

2) # Convert Gender to numeric

df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})

# Features and Target

X = df[['Gender', 'Age', 'EstimatedSalary']]

y = df['Purchased']

# Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42):

**Here's a breakdown of each line of code:**

1. **Convert Gender to numeric:**
2. df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})
   * **What it does**: This converts the **Gender** column, which has categorical values ('Male' and 'Female'), into numeric values:
     + 'Male' becomes 0
     + 'Female' becomes 1
   * **Why it's done**: Logistic Regression (and most machine learning algorithms) work better with numeric data. So, converting categories into numbers allows us to use this data in a model.
3. **Features and Target:**
4. X = df[['Gender', 'Age', 'EstimatedSalary']]
5. y = df['Purchased']
   * **What it does**:
     + X is the **features** (independent variables) we will use to predict the target. These are **Gender**, **Age**, and **EstimatedSalary**.
     + y is the **target** variable (dependent variable) we are trying to predict, which is **Purchased** (whether or not the person bought something based on the ad).
6. **Split the data:**
7. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)
   * **What it does**:
     + It splits the data into **training** and **testing** datasets.
     + **X\_train and y\_train** are the features and target variables for the **training** set, which the model will use to learn from.
     + **X\_test and y\_test** are the features and target variables for the **testing** set, which will be used to evaluate the model’s performance after it has been trained.
     + **test\_size=0.25** means 25% of the data will be used for testing, and the remaining 75% will be used for training.
     + **random\_state=42** ensures that the split is reproducible. Every time you run this, the data will be split the same way.

**Why are we doing these steps?**

* **Converting Gender to numeric**: Machine learning algorithms (including Logistic Regression) require numeric data for calculations, so we need to convert categorical variables like "Male" and "Female" into numbers.
* **Defining Features and Target**: In a supervised learning task like this, we need to clearly identify which variables are inputs (features) and which one is the output (target). Here, we're using the features **Gender**, **Age**, and **EstimatedSalary** to predict whether someone will purchase the product (target variable **Purchased**).
* **Splitting the data**: We split the dataset into training and testing sets to ensure that the model is evaluated on data it hasn't seen during training. This gives a better indication of how well it will perform on new, unseen data.

Let me know if you need more details!

3): sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

Let me break it down into simpler steps and explain why and how we scale the data:

**1. Why Scale the Data?**

Imagine you have two features in your dataset:

* **Age**: Values range from 20 to 60.
* **Estimated Salary**: Values range from 10,000 to 200,000.

These features have very **different ranges**. The **Age** has values between 20 and 60, while the **Estimated Salary** has values between 10,000 and 200,000. This difference in scales can be a problem for many machine learning algorithms (including Logistic Regression) because the model might give more importance to the feature with the larger range (in this case, **Estimated Salary**).

**2. What is Standardization/Scaling?**

Standardization (or scaling) is the process of **transforming your data** to make all the features have the **same scale**. This helps the algorithm treat all features equally, leading to better predictions.

**3. How do we Scale the Data?**

We use **StandardScaler** to scale the features. Here’s what it does:

* **Step 1: Fit the scaler on the training data**: It calculates two things for each feature in your training data:
  + The **mean** (average value).
  + The **standard deviation** (how spread out the values are).
* **Step 2: Transform the data**: Once the mean and standard deviation are calculated, it scales the data by subtracting the mean and dividing by the standard deviation. After this, each feature will have a **mean of 0** and a **standard deviation of 1**.

**4. Breaking Down the Code**

sc = StandardScaler() # Create a StandardScaler object.

This line creates a **scaler** that we will use to transform the data.

X\_train = sc.fit\_transform(X\_train)

* **fit\_transform()** does **two things**:
  1. **Fit**: It calculates the **mean** and **standard deviation** of each feature in the **training data** (X\_train).
  2. **Transform**: It then scales the training data by subtracting the mean and dividing by the standard deviation. Now, the **training data** is on the same scale.

X\_test = sc.transform(X\_test)

* **transform()**: This line scales the **test data** (X\_test) using the **mean** and **standard deviation** that were calculated from the **training data**.
  + We do not want to calculate new scaling parameters from the test data. The test data should be scaled using the parameters learned from the training data.

**5. Why Use fit\_transform() and transform() Separately?**

* **fit\_transform()** is used on the **training data** because we need to calculate the scaling parameters (mean and standard deviation) based on it.
* **transform()** is used on the **test data** because we don’t want to recalculate the scaling parameters; we use the parameters from the training data to scale the test data.

**6. What Happens After Scaling?**

After scaling, both **Age** and **EstimatedSalary** will be transformed to have similar scales (a mean of 0 and standard deviation of 1). This ensures that no feature dominates the learning process due to its larger range

4) # Train the model

classifier = LogisticRegression()

classifier.fit(X\_train, y\_train)

# Make predictions

y\_pred = classifier.predict(X\_test)

Let's break down this code step-by-step to understand what is happening:

**1. Training the Logistic Regression Model**

classifier = LogisticRegression()

* This line creates an object classifier of the LogisticRegression class from sklearn.linear\_model.
* **Logistic Regression** is a machine learning algorithm used for binary classification. It helps predict whether something belongs to one of two classes. In your case, the target variable Purchased is a binary classification (whether the customer purchased the product or not).

classifier.fit(X\_train, y\_train)

* **fit()** is a method that is used to train the model.
* X\_train contains the **input features** (such as Gender, Age, and Estimated Salary), and y\_train contains the **target variable** (whether the user purchased the product or not).
* The **model learns** from the training data by finding patterns or relationships between the input features (independent variables) and the target variable (dependent variable).
* The **logistic regression model** estimates the probability of an event (purchase or no purchase) based on the input features.

**2. Making Predictions**

y\_pred = classifier.predict(X\_test)

* **predict()** is used to make predictions after the model has been trained.
* X\_test is the input data that the model has never seen before (this is the test set).
* The classifier.predict(X\_test) will output the predicted labels (whether the customer will purchase or not) for the test data (X\_test).
* These predictions are stored in the variable y\_pred.

**What happens in the background?**

* **Training (fit)**: The model learns from the training data (X\_train and y\_train).
* **Prediction (predict)**: The model uses the learned patterns to make predictions on the unseen test data (X\_test).

5). # Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", cm)

# Extract values

TN, FP, FN, TP = cm.ravel()

# Metrics

accuracy = accuracy\_score(y\_test, y\_pred)

error\_rate = 1 - accuracy

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

print(f"True Positives (TP): {TP}")

print(f"False Positives (FP): {FP}")

print(f"True Negatives (TN): {TN}")

print(f"False Negatives (FN): {FN}")

print(f"Accuracy: {accuracy:.2f}")

print(f"Error Rate: {error\_rate:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

Let's break down this code step-by-step to understand the calculations and what each metric means.

**1. Confusion Matrix**

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", cm)

* **Confusion Matrix** is a table used to evaluate the performance of a classification model. It shows the actual vs. predicted values.
* The confusion matrix for a binary classification problem (like this one: "Purchased" or "Not Purchased") looks like this:

|  | **Predicted No (0)** | **Predicted Yes (1)** |
| --- | --- | --- |
| **Actual No (0)** | TN (True Negative) | FP (False Positive) |
| **Actual Yes (1)** | FN (False Negative) | TP (True Positive) |

* **True Negative (TN)**: The model correctly predicted "No" (didn't purchase).
* **False Positive (FP)**: The model incorrectly predicted "Yes" (purchased) when it should have been "No."
* **False Negative (FN)**: The model incorrectly predicted "No" (didn't purchase) when it should have been "Yes."
* **True Positive (TP)**: The model correctly predicted "Yes" (purchased).

The confusion matrix cm will display these values in a 2x2 matrix:

[[TN, FP],

[FN, TP]]

**2. Extracting the Values (TN, FP, FN, TP)**

TN, FP, FN, TP = cm.ravel()

* This line **unpacks** the confusion matrix into the four components: **True Negatives (TN), False Positives (FP), False Negatives (FN), and True Positives (TP)**.

**3. Metrics Calculation**

* **Accuracy**: It tells you how often the model is correct.

accuracy = accuracy\_score(y\_test, y\_pred)

Formula:

Accuracy=TP + TNTP + TN + FP + FN\text{Accuracy} = \frac{\text{TP + TN}}{\text{TP + TN + FP + FN}}

It gives the proportion of correct predictions (both "yes" and "no").

* **Error Rate**: It tells you how often the model is incorrect.

error\_rate = 1 - accuracy

Formula:

Error Rate=FP + FNTP + TN + FP + FN=1−Accuracy\text{Error Rate} = \frac{\text{FP + FN}}{\text{TP + TN + FP + FN}} = 1 - \text{Accuracy}

It gives the proportion of incorrect predictions.

* **Precision**: It tells you how many of the predicted "Yes" cases are actually correct.

precision = precision\_score(y\_test, y\_pred)

Formula:

Precision=TPTP + FP\text{Precision} = \frac{\text{TP}}{\text{TP + FP}}

It focuses on the **positive predictions** and measures how accurate those predictions are.

* **Recall**: It tells you how many of the actual "Yes" cases were correctly identified by the model.

recall = recall\_score(y\_test, y\_pred)

Formula:

Recall=TPTP + FN\text{Recall} = \frac{\text{TP}}{\text{TP + FN}}

It focuses on the **actual positives** and measures how well the model identifies them.

**4. Output the Results**

print(f"True Positives (TP): {TP}")

print(f"False Positives (FP): {FP}")

print(f"True Negatives (TN): {TN}")

print(f"False Negatives (FN): {FN}")

print(f"Accuracy: {accuracy:.2f}")

print(f"Error Rate: {error\_rate:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

* This part of the code prints the extracted values and the calculated metrics in a readable format.
* The **{:.2f}** ensures that the floating-point values are displayed with two decimal places.

6). sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

🡪sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

**does 4 things**:

| **Part** | **Meaning** |
| --- | --- |
| sns.heatmap(cm) | Draws a heatmap (colorful table) of the confusion matrix cm. |
| annot=True | Writes the **numbers** (like TP, FP, etc.) inside the boxes. |
| fmt='d' | Format the numbers as **integers** (no decimal points). |
| cmap='Blues' | Uses a **blue color theme**. Darker color = bigger number. |

 **True Negative (TN)**: The model correctly predicted "No" (didn't purchase).

 **False Positive (FP)**: The model incorrectly predicted "Yes" (purchased) when it should have been "No."

 **False Negative (FN)**: The model incorrectly predicted "No" (didn't purchase) when it should have been "Yes."

 **True Positive (TP)**: The model correctly predicted "Yes" (purchased).

Let's break down this code step-by-step to understand the calculations and what each metric means.

**1. Confusion Matrix**

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", cm)

* **Confusion Matrix** is a table used to evaluate the performance of a classification model. It shows the actual vs. predicted values.
* The confusion matrix for a binary classification problem (like this one: "Purchased" or "Not Purchased") looks like this:

|  | **Predicted No (0)** | **Predicted Yes (1)** |
| --- | --- | --- |
| **Actual No (0)** | TN (True Negative) | FP (False Positive) |
| **Actual Yes (1)** | FN (False Negative) | TP (True Positive) |

* **True Negative (TN)**: The model correctly predicted "No" (didn't purchase).
* **False Positive (FP)**: The model incorrectly predicted "Yes" (purchased) when it should have been "No."
* **False Negative (FN)**: The model incorrectly predicted "No" (didn't purchase) when it should have been "Yes."
* **True Positive (TP)**: The model correctly predicted "Yes" (purchased).

The confusion matrix cm will display these values in a 2x2 matrix:

[[TN, FP],

[FN, TP]]

**2. Extracting the Values (TN, FP, FN, TP)**

TN, FP, FN, TP = cm.ravel()

* This line **unpacks** the confusion matrix into the four components: **True Negatives (TN), False Positives (FP), False Negatives (FN), and True Positives (TP)**.

**3. Metrics Calculation**

* **Accuracy**: It tells you how often the model is correct.

accuracy = accuracy\_score(y\_test, y\_pred)

Formula:

Accuracy=TP + TNTP + TN + FP + FN\text{Accuracy} = \frac{\text{TP + TN}}{\text{TP + TN + FP + FN}}

It gives the proportion of correct predictions (both "yes" and "no").

* **Error Rate**: It tells you how often the model is incorrect.

error\_rate = 1 - accuracy

Formula:

Error Rate=FP + FNTP + TN + FP + FN=1−Accuracy\text{Error Rate} = \frac{\text{FP + FN}}{\text{TP + TN + FP + FN}} = 1 - \text{Accuracy}

It gives the proportion of incorrect predictions.

* **Precision**: It tells you how many of the predicted "Yes" cases are actually correct.

precision = precision\_score(y\_test, y\_pred)

Formula:

Precision=TPTP + FP\text{Precision} = \frac{\text{TP}}{\text{TP + FP}}

It focuses on the **positive predictions** and measures how accurate those predictions are.

* **Recall**: It tells you how many of the actual "Yes" cases were correctly identified by the model.

recall = recall\_score(y\_test, y\_pred)

Formula:

Recall=TPTP + FN\text{Recall} = \frac{\text{TP}}{\text{TP + FN}}

It focuses on the **actual positives** and measures how well the model identifies them.

**4. Output the Results**

print(f"True Positives (TP): {TP}")

print(f"False Positives (FP): {FP}")

print(f"True Negatives (TN): {TN}")

print(f"False Negatives (FN): {FN}")

print(f"Accuracy: {accuracy:.2f}")

print(f"Error Rate: {error\_rate:.2f}")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

* This part of the code prints the extracted values and the calculated metrics in a readable format.
* The **{:.2f}** ensures that the floating-point values are displayed with two decimal places.

**Recap of Metrics:**

* **True Positives (TP)**: The number of correct predictions of "Yes" (purchased).
* **False Positives (FP)**: The number of incorrect predictions of "Yes" when the true class is "No."
* **True Negatives (TN)**: The number of correct predictions of "No" (did not purchase).
* **False Negatives (FN)**: The number of incorrect predictions of "No" when the true class is "Yes."
* **Accuracy**: The proportion of correct predictions (both "Yes" and "No").
* **Error Rate**: The proportion of incorrect predictions.
* **Precision**: How many predicted "Yes" values are actually correct.
* **Recall**: How many actual "Yes" values were correctly identified.

**Practical Questions That Could Be Asked:**

1. **What is the confusion matrix, and how does it help evaluate a model?**
   * It provides a detailed view of how the model performs in terms of correct and incorrect predictions for each class.
2. **What is the difference between precision and recall?**
   * Precision is about the accuracy of the positive predictions, while recall is about how well the model captures the actual positives.
3. **What does the accuracy score indicate about the model's performance?**
   * It gives the proportion of correct predictions out of all predictions, showing the overall correctness of the model.
4. **Why is the error rate important to consider?**
   * The error rate gives insight into how often the model makes mistakes, helping identify the limitations of the model.

Let me know if you need further clarification!

**Viva Questions :**

Here are some **basic and advanced questions** that might be asked in a viva based on the **Logistic Regression** model applied to the **Social Network Ads** dataset:

**1. What is Logistic Regression?**

**Answer:**  
Logistic Regression is a classification algorithm used to predict the probability of a categorical dependent variable. It works by estimating the relationship between the independent variables and the dependent binary outcome using the logistic function (sigmoid function). The model outputs probabilities that a given input belongs to a particular class, which can then be thresholded (usually at 0.5) to assign a label.

**2. What is the role of the 'Gender', 'Age', and 'EstimatedSalary' columns in your model?**

**Answer:**

* **Gender**: A categorical feature representing the gender of the user (Male or Female). In the code, it is converted into a numerical format (0 for Male and 1 for Female) to make it usable in the logistic regression model.
* **Age**: A continuous numerical feature that represents the user's age.
* **EstimatedSalary**: A continuous numerical feature representing the estimated salary of the user.

These features are used as independent variables (X) to predict the dependent variable (y), which is whether the user purchased the product (1 for purchased, 0 for not purchased).

**3. Why is the 'Gender' feature mapped to 0 and 1?**

**Answer:**  
The Gender feature is categorical, with values like "Male" and "Female". Logistic Regression can only handle numerical values, so we convert these categorical values into numerical values using the map() function. Here, "Male" is mapped to 0 and "Female" is mapped to 1 to make them suitable for input into the logistic regression model.

**4. Why do we scale the features using StandardScaler?**

**Answer:**  
Scaling the features using StandardScaler is important because Logistic Regression (like many machine learning algorithms) works better when the input features are on a similar scale. StandardScaler standardizes the features by removing the mean and scaling to unit variance, which helps the model converge faster and avoids the dominance of certain features with larger values over others with smaller values.

**5. What does the train\_test\_split() function do?**

**Answer:**  
The train\_test\_split() function from the sklearn.model\_selection module is used to randomly split the dataset into two parts: one for training the model (usually 75-80% of the data) and one for testing the model (the remaining 20-25% of the data). This helps evaluate the model's performance on unseen data, ensuring that the model is not overfitting to the training data.

**6. How does the fit() method work in Logistic Regression?**

**Answer:**  
The fit() method in logistic regression is used to train the model. It calculates the best-fitting line (decision boundary) that separates the classes by adjusting the model’s parameters (weights). The goal of training is to find the coefficients for the independent variables that minimize the error between the predicted and actual values using a cost function (typically log-likelihood).

**7. What does the predict() function do in the Logistic Regression model?**

**Answer:**  
The predict() function is used to make predictions based on the trained logistic regression model. After training, we use predict() to input new, unseen data (in this case, X\_test) and output the predicted class labels (whether the user will purchase or not).

**8. What is a Confusion Matrix, and why is it important?**

**Answer:**  
A Confusion Matrix is a table used to evaluate the performance of a classification model. It shows the counts of actual vs. predicted classifications, with the following components:

* **True Positives (TP)**: Correctly predicted positive cases.
* **True Negatives (TN)**: Correctly predicted negative cases.
* **False Positives (FP)**: Incorrectly predicted positive cases.
* **False Negatives (FN)**: Incorrectly predicted negative cases.

It provides insights into how well the model is performing, highlighting where the model is making errors.

**9. What do the metrics Accuracy, Error Rate, Precision, and Recall mean?**

**Answer:**

* **Accuracy**: The proportion of correctly classified instances (both positive and negative) out of all instances.

Accuracy=TP+TNTP+TN+FP+FNAccuracy = \frac{TP + TN}{TP + TN + FP + FN}

* **Error Rate**: The proportion of incorrectly classified instances (both false positives and false negatives) out of all instances.

ErrorRate=1−AccuracyError Rate = 1 - Accuracy

* **Precision**: The proportion of true positives out of all predicted positives. It answers the question: "Out of all instances the model classified as positive, how many are actually positive?"

Precision=TPTP+FPPrecision = \frac{TP}{TP + FP}

* **Recall**: The proportion of true positives out of all actual positives. It answers the question: "Out of all actual positive instances, how many did the model correctly identify?"

Recall=TPTP+FNRecall = \frac{TP}{TP + FN}

**10. What does the heatmap represent in the code?**

**Answer:**  
The heatmap visualizes the confusion matrix. It shows the number of true positives, false positives, true negatives, and false negatives in a color-coded format, making it easier to interpret the performance of the classification model. The diagonal cells (top-left and bottom-right) represent correct predictions, while off-diagonal cells represent misclassifications.

**11. What would happen if you didn’t scale the features before training the Logistic Regression model?**

**Answer:**  
If you don’t scale the features, features with larger ranges (like Age or EstimatedSalary) could dominate the model's learning process, and the model might not perform optimally. Features with smaller ranges would be underrepresented. Feature scaling ensures all features contribute equally to the model's learning.

**12. Why is Logistic Regression suitable for this classification problem?**

**Answer:**  
Logistic Regression is suitable for binary classification problems like this one, where the target variable (whether a person purchased or not) is binary (0 or 1). It is a simple, interpretable algorithm that works well when the relationship between the independent variables and the target is approximately linear, which is the case here.

**13. What would you do if you had an imbalanced dataset (i.e., one class is much more frequent than the other)?**

**Answer:**  
If the dataset is imbalanced, the model may be biased towards the more frequent class. To address this, we could:

* Use **stratified sampling** during data splitting to ensure both classes are well represented in training and testing.
* **Resample** the dataset by either **over-sampling** the minority class or **under-sampling** the majority class.
* Use different metrics like **Precision-Recall AUC** instead of accuracy for evaluation.
* Apply **class weights** to penalize the model more for misclassifying the minority class.

**14. Can you explain the significance of True Positives, False Positives, True Negatives, and False Negatives?**

**Answer:**

* **True Positives (TP)**: These are the instances where the model correctly predicted a positive outcome (e.g., a user bought the product, and the model predicted they would).
* **False Positives (FP)**: These are the instances where the model incorrectly predicted a positive outcome (e.g., the model predicted the user would buy the product, but they did not).
* **True Negatives (TN)**: These are the instances where the model correctly predicted a negative outcome (e.g., a user did not buy the product, and the model predicted they would not).
* **False Negatives (FN)**: These are the instances where the model incorrectly predicted a negative outcome (e.g., the model predicted the user would not buy the product, but they did).

These questions cover a broad spectrum of topics related to the implementation and evaluation of the Logistic Regression model. They also touch on performance metrics, data preprocessing, and specific model characteristics, helping you prepare thoroughly for the viva.

Here are some **basic questions** that can be asked in the viva for your **Logistic Regression** implementation on the **Social Network Ads dataset**:

**1. What is Logistic Regression?**

* **Answer**: Logistic Regression is a statistical model used for binary classification problems. It predicts the probability of the default class using a logistic (sigmoid) function. The output is a value between 0 and 1, which can be interpreted as the probability of belonging to one of the classes.

**2. Why do we need to preprocess the 'Gender' feature?**

* **Answer**: Logistic Regression requires numerical input features. The 'Gender' column is categorical, with values "Male" and "Female." To use this feature in the model, it is converted to numerical values (0 for Male and 1 for Female) using the map() function.

**3. What is the purpose of splitting the dataset using train\_test\_split?**

* **Answer**: The train\_test\_split() function splits the dataset into training and testing sets. This ensures that the model is trained on one part of the data and evaluated on a separate, unseen part. It helps to prevent overfitting and ensures the model generalizes well to new data.

**4. What is the role of the StandardScaler in the code?**

* **Answer**: The StandardScaler is used to scale the features so that they have a mean of 0 and a standard deviation of 1. Feature scaling is important for algorithms like Logistic Regression because it prevents features with larger ranges from dominating the learning process.

**5. How does the Logistic Regression model make predictions?**

* **Answer**: The Logistic Regression model uses the sigmoid function to calculate the probability of an instance belonging to a particular class. After training, the model uses the learned weights to predict the probability that a given input belongs to the positive class. This probability is then thresholded (usually at 0.5) to classify the input.

**6. What does the confusion matrix tell us about the model’s performance?**

* **Answer**: The confusion matrix shows the number of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) predictions made by the model. It helps evaluate the model's accuracy and error distribution, which can be used to compute performance metrics like precision, recall, and accuracy.

**7. What are True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN)?**

* **Answer**:
  + **True Positives (TP)**: Correctly predicted positive cases (e.g., a user bought the product, and the model predicted they would).
  + **False Positives (FP)**: Incorrectly predicted positive cases (e.g., the model predicted the user would buy the product, but they did not).
  + **True Negatives (TN)**: Correctly predicted negative cases (e.g., a user did not buy the product, and the model predicted they would not).
  + **False Negatives (FN)**: Incorrectly predicted negative cases (e.g., the model predicted the user would not buy the product, but they did).

**8. What is Accuracy in the context of the confusion matrix?**

* **Answer**: Accuracy is the proportion of correctly predicted instances (both positive and negative) out of all instances in the dataset. It is calculated as:

Accuracy=TP+TNTP+TN+FP+FNAccuracy = \frac{TP + TN}{TP + TN + FP + FN}

**9. What is Precision, and how is it different from Recall?**

* **Answer**:
  + **Precision**: The proportion of true positive predictions out of all positive predictions made by the model. It answers the question: "Of all the predicted positive cases, how many are actually positive?"

Precision=TPTP+FPPrecision = \frac{TP}{TP + FP}

* + **Recall**: The proportion of true positive predictions out of all actual positive instances. It answers the question: "Of all the actual positive cases, how many did the model correctly identify?"

Recall=TPTP+FNRecall = \frac{TP}{TP + FN}

**10. What is the significance of using a heatmap to visualize the confusion matrix?**

* **Answer**: The heatmap is used to visualize the confusion matrix in a more interpretable format. It shows the values of TP, FP, TN, and FN in a color-coded matrix, making it easier to understand the model's performance and identify areas of improvement.

**11. What would you do if the model’s performance is not satisfactory?**

* **Answer**: If the model's performance is poor, I would consider the following steps:
  + Check the quality and distribution of the data (e.g., missing values, outliers, or class imbalance).
  + Try different preprocessing techniques or feature engineering (e.g., adding/removing features).
  + Tune hyperparameters or try regularization techniques to improve model generalization.
  + Use different classification algorithms (e.g., Random Forest, SVM) to compare results.

**12. What is the difference between Logistic Regression and Linear Regression?**

* **Answer**:
  + **Linear Regression** is used for predicting continuous numerical values, while **Logistic Regression** is used for binary classification tasks (predicting a class label of 0 or 1).
  + In Linear Regression, the model outputs a continuous value, whereas in Logistic Regression, the model outputs probabilities which are then thresholded to make a classification.

These questions focus on the core concepts of Logistic Regression, feature preprocessing, and the model evaluation metrics used in the code. Preparing answers to these basic questions will help you feel more confident in your viva.