

## ✓ Learning Objectives

At the end of the experiment, you will be able to:

- learn about Classification tasks in Machine learning
- perform Logistic Regression, Softmax Regression
- learn the appropriate performance metrics according to use case
- have an understanding of Decision Boundaries

## ✓ Information

## ✓ Classification

**Classification** refers to a predictive modeling problem where a class label is predicted for a given example of input data.

**Examples include:**

- Email spam detection (spam or not).
- Churn prediction (churn or not).
- Conversion prediction (buy or not).

**Binary classification** refers to those classification tasks that have two class labels.

**Logistic Regression** is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.).

## Implementing Binary Classification with Logistic Regression

## ✓ Dataset

In this example, we will be using "Social\_Network\_Ads" dataset.

The variable descriptions are as follows:

- Age
- EstimatedSalary

The target feature is:

- Purchased

**Problem Statement:** To predict if a person will purchase an item based on age and estimated salary.

## ✓ Setup Steps:

### > Please enter your registration id to start:

Id:

[Show code](#)

### > Please enter your password (your registered phone number) to continue:

password:

[Show code](#)

### > Run this cell to complete the setup for this Notebook

[Show code](#)

## ✓ Importing required packages

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

## ✓ Importing the Dataset

```
df = pd.read_csv('Social_Network_Ads.csv')
X = df.iloc[:, 3].values # estimated salary
y = df.iloc[:, -1].values
X = X.reshape(-1, 1)
df.head()
```



	User ID	Gender	Age	EstimatedSalary	Purchased
--	---------	--------	-----	-----------------	-----------



0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15604000	Male	19	76000	0



Next steps:

[Generate code with df](#)[View recommended plots](#)[New interactive sheet](#)

## ✓ Splitting the dataset into the Training set and Test set

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

```
print(X_train)
```



```
[ 70000]  
[ 33000]  
[ 72000]  
[ 39000]  
[ 31000]  
[ 70000]  
[ 79000]  
[ 81000]  
[ 80000]  
[ 85000]  
[ 39000]  
[ 88000]  
[ 88000]  
[150000]  
[ 65000]  
[ 54000]  
[ 43000]  
[ 52000]  
[ 30000]  
[ 43000]  
[ 52000]  
[ 54000]  
[118000]]
```

```
print(y_train)
```

```
[0 1 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 1 0 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 1  
0 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 1  
1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0  
1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0  
0 0 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 0  
0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0  
0 1 1 0 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0  
0 0 1 0 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 1 1 1 1 0 0 0 0 1  
0 0 0 0]
```

```
print(X_test)
```

```
[ 84000]  
[ 20000]  
[112000]  
[ 58000]  
[ 80000]  
[ 90000]
```

```
[ 04000]
[146000]
[ 48000]
[ 33000]
[ 84000]
[ 96000]
[ 63000]
[ 33000]
[ 90000]
[104000]]
```

```
print(y_test)
```

```
↵ [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0
   0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1
   0 0 0 0 1 1 1 0 0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 1 1 1]
```

## ▼ Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
print(X_train)
```

```
↵
```

```
print(X_test)
```

```
[ 0.41798449]
[-1.43757673]
[ 1.22979253]
[-0.33583725]
[ 0.30201192]
[ 0.59194336]
[-1.14764529]
[ 0.47597078]
[ 1.51972397]
[-1.29261101]
[-0.3648304 ]
[ 1.31677196]
[ 0.53395707]
[-1.089659 ]
[ 0.38899135]
[ 0.30201192]
[-1.20563157]
[-1.43757673]
[-1.49556302]
[-0.79972756]
[ 0.18603934]
[ 0.85288166]
[-1.26361786]
[ 0.38899135]
[ 0.56295021]
[-0.33583725]
[-0.65476184]
[ 0.01208048]
[ 2.331532 ]
[ 0.21503249]
[-0.19087153]
[-1.37959044]
[ 0.56295021]
[ 0.35999821]
[ 0.27301877]
[-0.27785096]
[-1.03167271]
[ 1.08482681]
[ 2.15757314]
[ 0.38899135]
[-0.42281668]
[-1.00267957]
[-0.91570013]
[ 0.30201192]
[ 0.1570462 ]
[ 1.75166912]
[-0.8287207 ]
[-0.27785096]
[-0.16187839]
[ 2.21555943]
[-0.62576869]
[-1.06066585]
[ 0.41798449]
[ 0.76590222]
[-0.19087153]
[-1.06066585]
[ 0.59194336]
[ 0.99784738]]
```

## ✓ Training the Logistic Regression model on the Training set

```
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(random_state=0)
```

## ✓ Predicting a new test instance

```
print(classifier.predict(sc.transform([[87000]])))
```

```
[0]
```

## ✓ Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
↵ [0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 1]
[0 0]
[0 1]
[1 1]
[0 0]
[0 0]
[1 0]
[0 1]
[0 1]
[0 0]
[0 0]
[0 1]
[0 0]
[0 0]
[0 1]
[0 0]
[1 1]
[0 0]
[0 1]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 1]
[1 1]
[1 1]
[0 0]
[0 0]
[0 0]
[0 1]
[0 1]
[0 0]
[1 1]
[0 1]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 1]
[0 1]
[1 1]]
```

## ✓ Model Evaluation

To evaluate the performance of a classification model, the following metrics are used:

- Confusion matrix
  - Accuracy
  - Precision
  - Recall
  - F1-Score
- ROC curve
- AUROC

## ✓ Confusion Matrix

- **Confusion matrix:** is a table that is used to describe the performance of a classification model on a set of test data for which the true values are known.
  - **true positive** for correctly predicted event values.
  - **false positive** for incorrectly predicted event values.
  - **true negative** for correctly predicted no-event values.
  - **false negative** for incorrectly predicted no-event values.
- **Accuracy:** it is the ratio of the number of correct predictions to the total number of input samples.

```
# Creating a confusion matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

```
[[66  2]
 [17 15]]
```

	precision	recall	f1-score	support
0	0.80	0.97	0.87	68
1	0.88	0.47	0.61	32
accuracy			0.81	100
macro avg	0.84	0.72	0.74	100
weighted avg	0.82	0.81	0.79	100

This Confusion Matrix tells us that there were 81 correct predictions and 19 incorrect ones.

- True Positive: 15
- True Negative: 66
- False Positive: 2
- False Negative: 17

## ✓ Precision-Recall Metrics

- **Precision:** summarizes the fraction of examples assigned the positive class that belongs to the positive class.

$$\text{Precision} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalsePositive}}$$

- **Recall:** summarizes how well the positive class was predicted and is the same calculation as sensitivity.

$$\text{Recall} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalseNegative}}$$

- **F1-score:** precision and recall can be combined into a single score that seeks to balance both concerns, called the F-score or the F-measure.

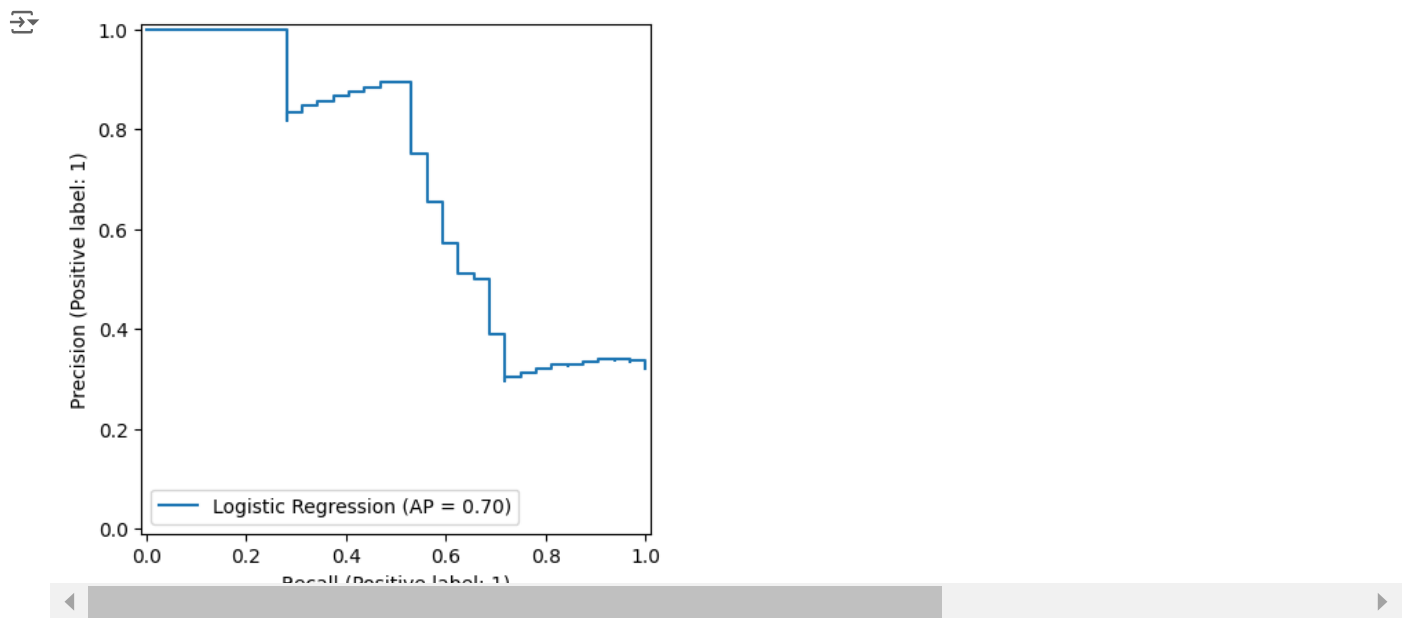
$$\text{F1-score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

## ✓ Plotting precision-recall curve using sklearn

# Use sklearn to plot precision-recall curves

```
from sklearn.metrics import PrecisionRecallDisplay
```

```
display = PrecisionRecallDisplay.from_estimator(
    classifier, # Your trained model
    X_test,     # Test features
    y_test,     # True labels
    name='Logistic Regression' # Name to display on the plot
)
```



The above diagram shows the blue line as precision-recall curve.

## ✓ ROC-AUC curve

A ROC curve is a diagnostic plot for summarizing the behavior of a model by calculating the false positive rate and true positive rate for a set of predictions by the model under different thresholds.

Area Under Curve (AUC) is one of the most widely used metrics for evaluation. It is used for binary classification problems.

AUC has a range of [0, 1]. The greater the value, the better is the performance of our model.

## ✓ Plotting the ROC-AUC curve for Logistic Regression algorithm using matplotlib

```
# roc_curve() computes the ROC for the classifier and returns the FPR, TPR, and threshold values
from sklearn.metrics import roc_curve
```

```
classifier.fit(X_train, y_train)
pred_prob1 = classifier.predict_proba(X_test)
```

```
# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(y_test, pred_prob1[:,1], pos_label=1)
```

```
# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)
```

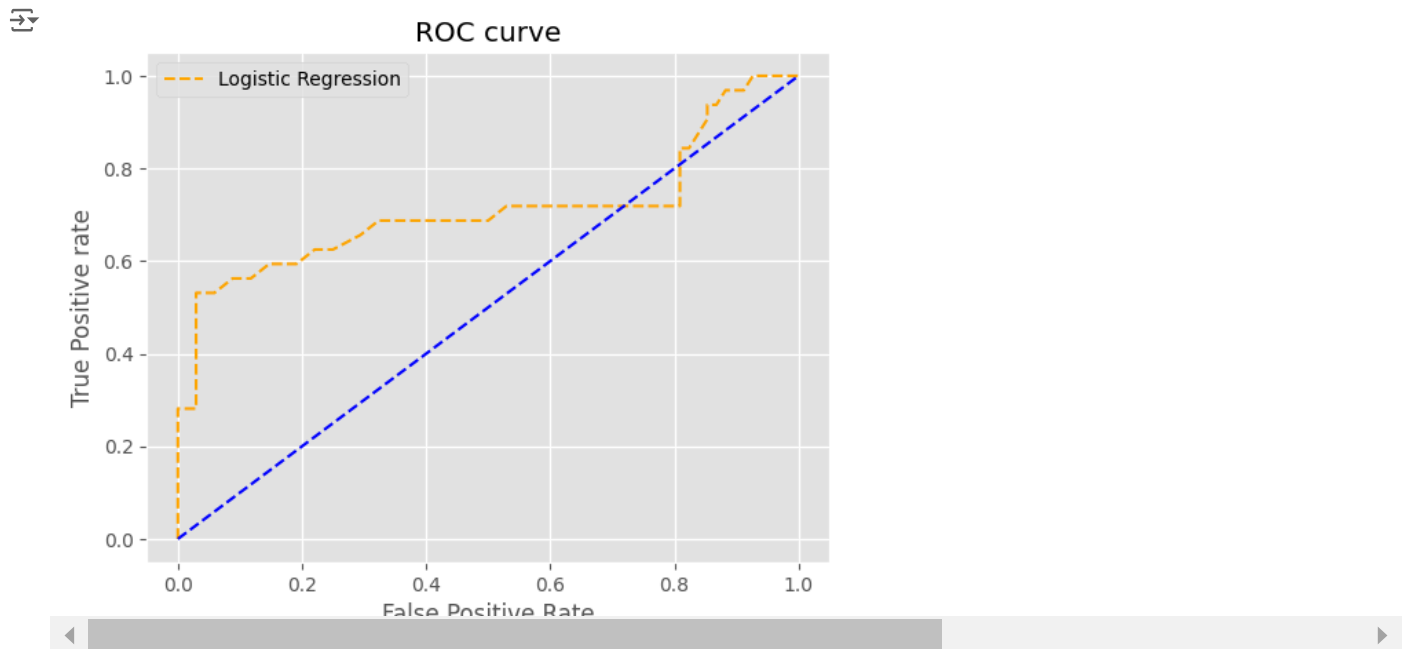
```
plt.style.use('ggplot') #using ggplot cause seaborn isn't working
```

```
# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--', color='orange', label='Logistic Regression')
```

```
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')
```

```
plt.legend(loc='best')
plt.savefig('ROC', dpi=300)
plt.show();
```





The above diagram shows:

ROC curve: is the orange dotted line

AUROC: is the area under the orange dotted line

The blue dotted line is the reference line.

Please refer to the given [link](#) for further information on Performance metrics and [ROC-AUC curve](#)

## ✓ Example: Predicting Diabetes with Logistic Regression

Let us now apply the above learnings to perform a logistic regression using a 'UCI PIMA Indian Diabetes' dataset.

- Fit the model
- Do the prediction
- Plot the ROC-AUC curve for the Logistic Regression algorithm

## ✓ Dataset

In this example, we will be using the "UCI PIMA Indian Diabetes" dataset.

The datasets consist of several medical predictor variables and one target variable, Outcome. Predictor variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

The variable descriptions are as follows:

- Pregnancies: Number of Pregnancies
- Glucose: Plasma glucose concentration over 2 hours in an oral glucose tolerance test
- Blood pressure: Diastolic blood pressure (mm Hg)
- SkinThickness: Triceps skinfold thickness (mm)
- Insulin: 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/(height in m)<sup>2</sup>)
- DiabetesPedigreeFunction: Diabetes pedigree function (a function which scores likelihood of diabetes based on family history)
- Age: Age (years)
- Outcome: Class variable (0 if non-diabetic, 1 if diabetic)

Problem statement:

We will be using this dataset to predict if a person has diabetes or not using the medical attributes provided.

## ✓ Loading the dataset

```
DF = pd.read_csv('diabetes.csv')
```

```
print(DF.head())
```

```

Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI  \
0           6       148             72           35      0   33.6
1           1        85             66           29      0   26.6
2           8       183             64            0      0   23.3
3           1        89             66           23     94   28.1
4           0       137             40           35    168   43.1

DiabetesPedigreeFunction  Age  Outcome
0              0.627     50         1
1              0.351     31         0
2              0.672     32         1
3              0.167     21         0
4              2.288     33         1

```

### ✓ Finding if there are any null values

```
print(DF.isnull().sum())
```

```

Pregnancies      0
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
DiabetesPedigreeFunction  0
Age              0
Outcome          0
dtype: int64

```

### ✓ Training our model

```

# Separating the data into independent and dependent variables
X = DF.iloc[:, :-1].values # All rows, all columns except the last
y = DF.iloc[:, -1].values

```

### ✓ Splitting the data into training and testing data

```

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

### ✓ Training the Logistic Regression model on the Training set

```

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state=42, max_iter=500) # Increase max_iter to 500
model.fit(X_train, y_train)

```

```

LogisticRegression
LogisticRegression(max_iter=500, random_state=42)

```

### ✓ Training/Fitting the Model

```
# YOUR CODE HERE
```

### ✓ Making Predictions

```

y_pred = model.predict(X_test)

# Print predictions (optional)
print("\nPredictions on the test set:")
print(y_pred)

```

```

Predictions on the test set:
[0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 1 1 0 0 0 0 1 1 1 1 1 1
 0 1 1 0 1 1 0 0 1 1 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 1 1 0 0
 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0 1 0 1 1 1 0 0 1 0
 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 1 1 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0

```

```
0 1 0 0 0 0]
```

## ▼ Confusion Matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print results
print("\nConfusion Matrix:")
print(cm)
print(f"\nAccuracy: {accuracy:.2f}")
```



```
Confusion Matrix:
[[78 21]
 [18 37]]
```

```
Accuracy: 0.75
```

## ▼ Plotting the ROC curve for Logistic Regression algorithm using matplotlib

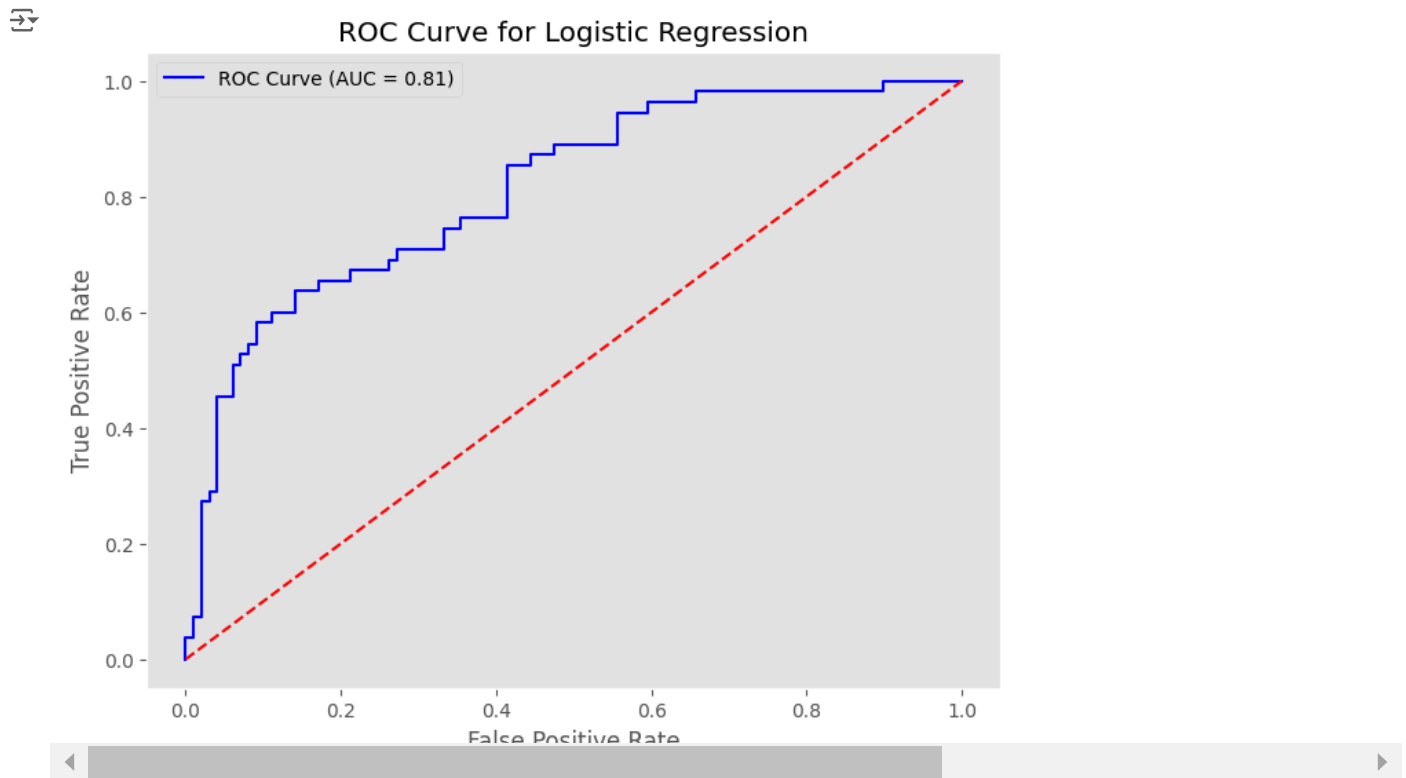
```
# YOUR CODE HEREimport matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

# Obtain predicted probabilities
y_prob = model.predict_proba(X_test)[: , 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

# Compute AUC
auc_score = roc_auc_score(y_test, y_prob)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {auc_score:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--') # Diagonal line for random guessing
plt.title('ROC Curve for Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid()
plt.show()
```



## ▼ Softmax Regression

The **Softmax regression** is a form of logistic regression that normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1.

It is also called **multinomial logistic regression**.

Performing Softmax Regression on the above dataset "Social\_Network\_Ads"

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender']) # Converts 'Male' to 1 and 'Female' to 0

# After encoding, concatenate 'Gender' with the 'Age' and 'EstimatedSalary' columns
X = df.iloc[:, [1, 2, 3]].values
y = df.iloc[:, -1].values

df.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	1	19	19000	0
1	15810944	1	35	20000	0
2	15668575	0	26	43000	0
3	15603246	0	27	57000	0
4	15604000	1	19	76000	0

Next steps:

[Generate code with df](#)
[View recommended plots](#)
[New interactive sheet](#)

## ▼ Splitting the dataset into the Training set and Test set


```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```




## ▼ Feature Scaling

```
sc = StandardScaler()
X_train[:, 1:] = sc.fit_transform(X_train[:, 1:]) # Only scale Age and EstimatedSalary (columns 1 and 2)
X_test[:, 1:] = sc.transform(X_test[:, 1:])
```

## Training the Softmax Regression model on the Training set


```
softmax_reg = LogisticRegression(multi_class='multinomial', # switch to Softmax Regression
                                solver='lbfgs', # handle multinomial loss, L2 penalty
                                C=10)
softmax_reg.fit(X, y)
```

 /usr/local/lib/python3.10/dist-packages/sklearn/linear\_model/\_logistic.py:1237: FutureWarning: 'multi\_class' was deprecated in warnings.warn()


 LogisticRegression    
LogisticRegression(C=10, multi\_class='multinomial')

## Predicting a new result

```
softmax_reg.predict(sc.transform([[30,87000]]))
```

 array([0])

```
softmax_reg.predict_proba(sc.transform([[30,87000]]))
```

 array([[9.9996675e-01, 3.3253487e-06]])

## Decision Boundary

In classification problems with two or more classes, a decision boundary is a hypersurface that separates the underlying vector space into sets, one for each class.

## Creating Dummy Dataset

```
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=200, n_features=2, n_informative=2, n_redundant=0, n_classes=2, random_state=1)
```

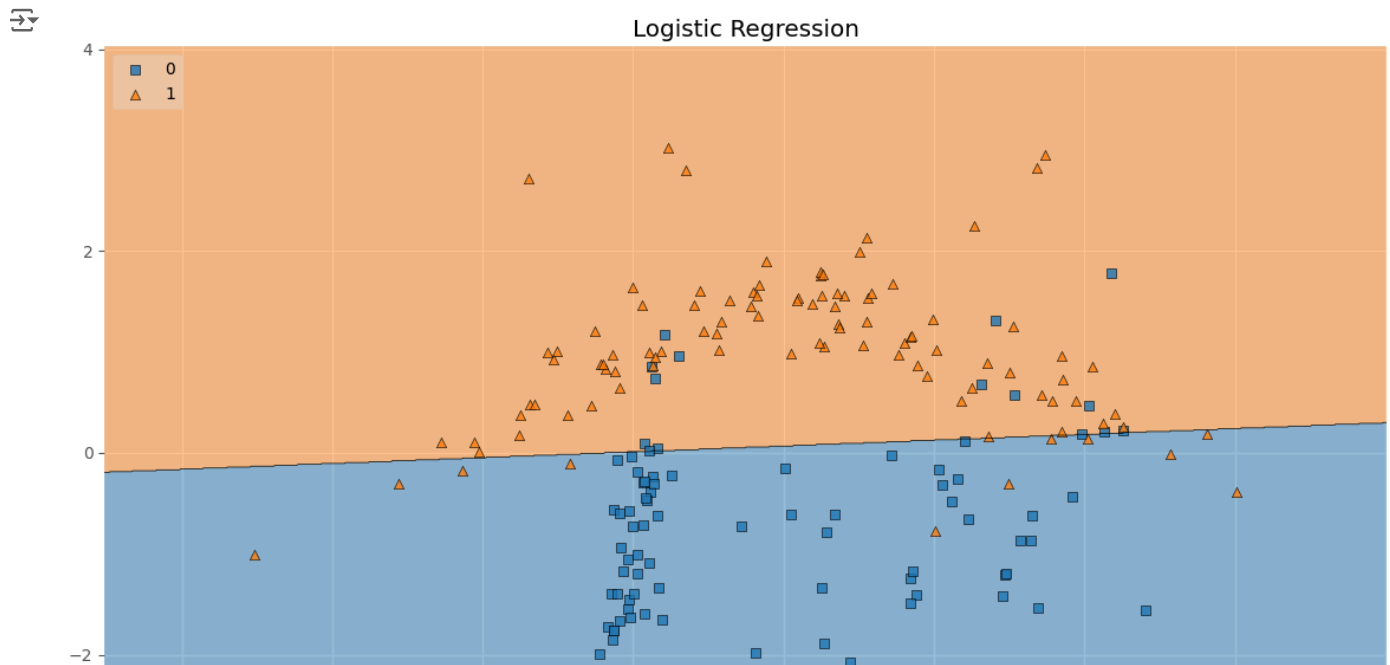
## Creating Decision Boundary

```
import matplotlib.gridspec as gridspec
from mlxtend.plotting import plot_decision_regions
gs = gridspec.GridSpec(3, 2)
```

```
fig = plt.figure(figsize=(14,10))
```

```
label = 'Logistic Regression'
clf = LogisticRegression()
clf.fit(X, y)
```

```
fig = plot_decision_regions(X=X, y=y, clf=clf, legend=2)
plt.title(label)
plt.show()
```



### Reference

<https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f>

### Theory Questions

1. Is it a good idea to stop Mini-batch Gradient Descent immediately when the validation error goes up?

Both Mini-batch and Stochastic gradient descent are not guaranteed to minimize the cost function after each step because they both have a degree of randomness built into them. Mini-batch randomly chooses which training examples to perform gradient descent on while Stochastic randomly chooses a single example. A better option is to save the model at regular intervals. When the model has not