Assignment is at the bottom!

In [1]:
```
from sklearn.linear_model import LogisticRegression
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

from pylab import rcParams
rcParams['figure.figsize'] = 20, 10


from sklearn.linear_model import LogisticRegression as Model
```

In [2]:
```
y = np.concatenate([np.zeros(10), np.ones(10)])
x = np.linspace(0, 10, len(y))
```

In [3]:
```
plt.scatter(x, y, c=y)
```

Out[3]:
```
<matplotlib.collections.PathCollection at 0x200a05bd660>
```

In [4]:
```
model = LogisticRegression()
```

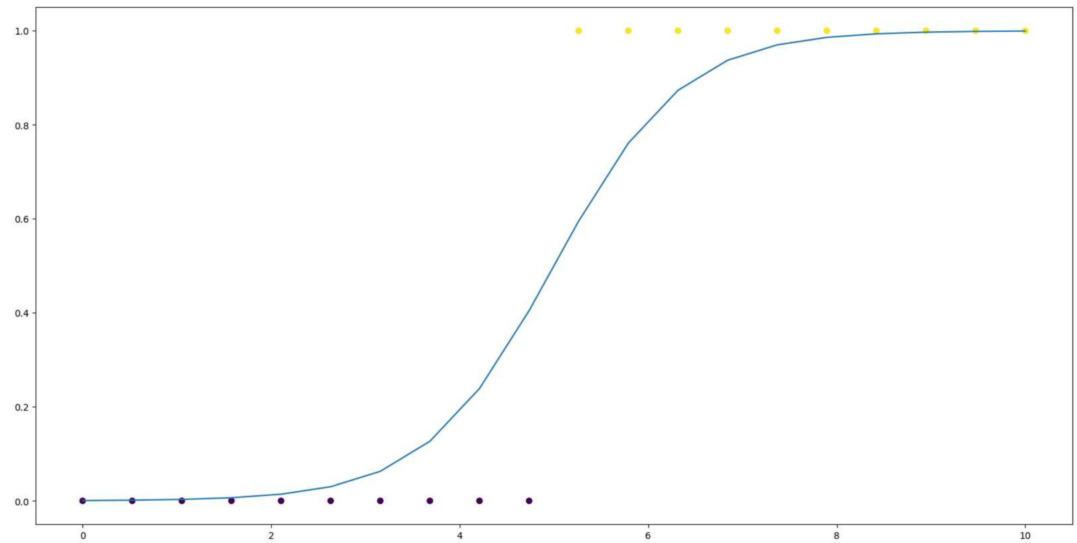
In [5]:
```
model.fit(x.reshape(-1, 1),y)
```

Out[5]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [6]:
```
plt.scatter(x,y, c=y)
plt.plot(x, model.predict_proba(x.reshape(-1, 1))[:,1])
```

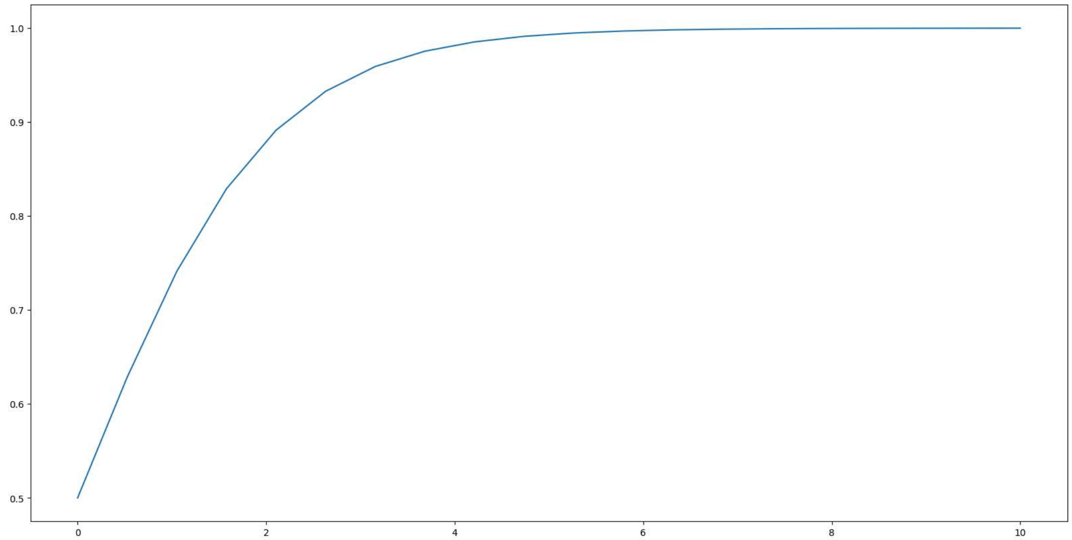
Out[6]:
```
[<matplotlib.lines.Line2D at 0x200a153d450>]
```



In [7]:
```
b, b0 = model.coef_, model.intercept_
model.coef_, model.intercept_
```

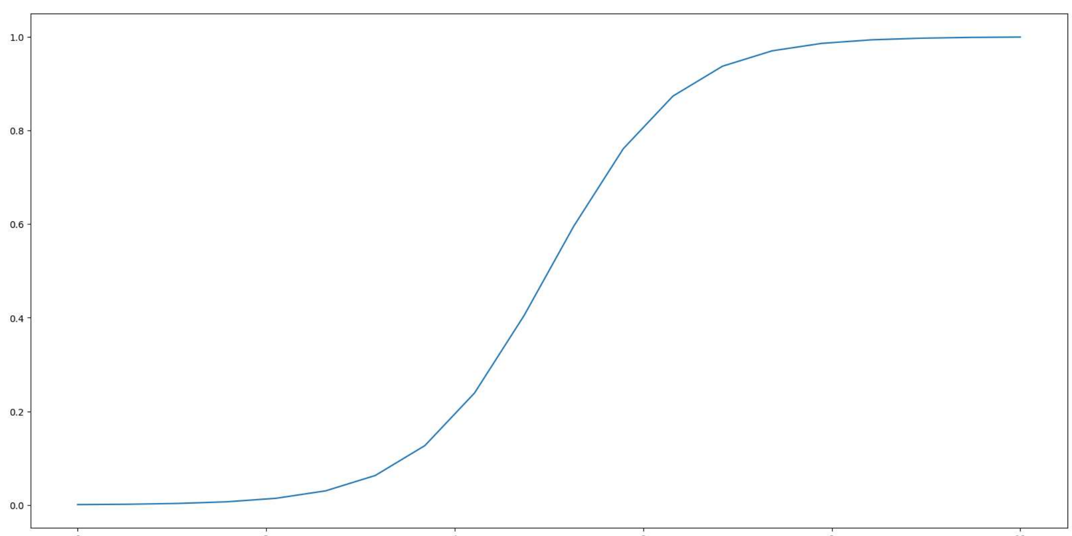
Out[7]:
```
(array([[1.46709085]]), array([-7.33542562]))
```

In [8]:
```
plt.plot(x, 1/(1+np.exp(-x)))
```
Out[8]: [<matplotlib.lines.Line2D at 0x200a0661d20>]



In [9]:
```
b
```
Out[9]: array([[1.46709085]])

In [10]:
```
plt.plot(x, 1/(1+np.exp(-(b[0]*x +b0))))
```
Out[10]: [<matplotlib.lines.Line2D at 0x200a06cb5b0>]



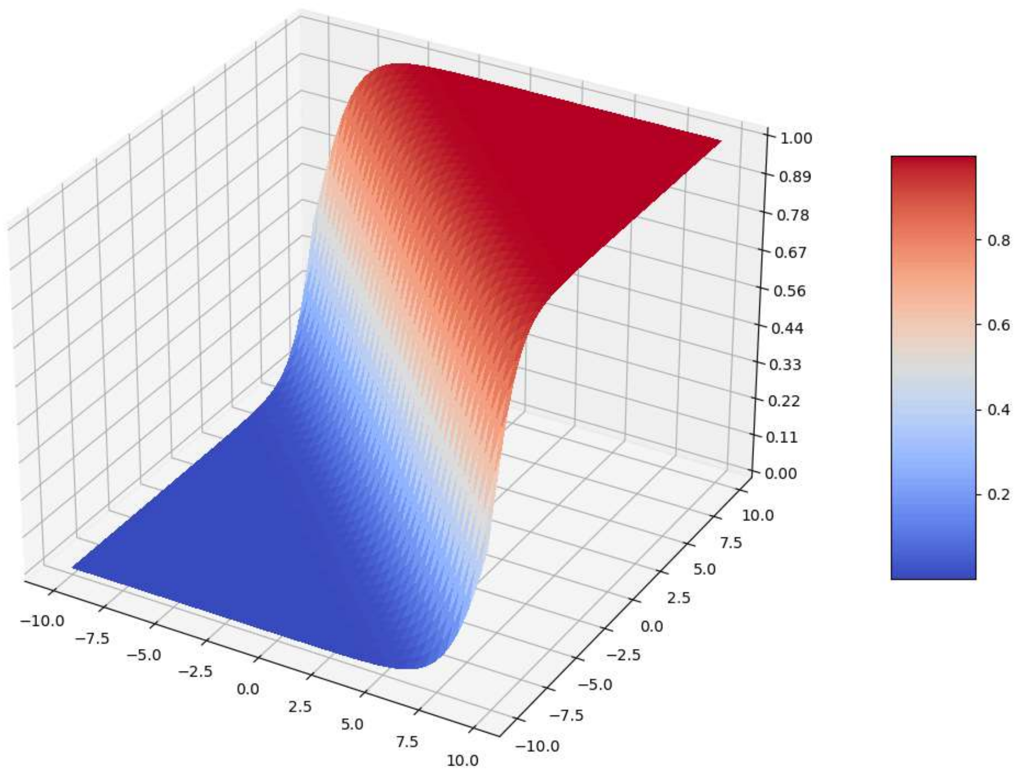
In [14]:

```
# OLD CODE - DOES NOT WORK.
#from mpl_toolkits.mplot3d import Axes3D  # noqa: F401 unused import

#import matplotlib.pyplot as plt
#from matplotlib import cm
#from matplotlib.ticker import LinearLocator, FormatStrFormatter
#import numpy as np


#fig = plt.figure()
#ax = fig.gca(projection='3d')

## Make data.
#X = np.arange(-10, 10, 0.25)
#Y = np.arange(-10, 10, 0.25)
#X, Y = np.meshgrid(X, Y)
#R = np.sqrt(X**2 + Y**2)
#Z = 1/(1+np.exp(-(b[0]*X +b[0]*Y +b0)))
#surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                   #    linewidth=0, antialiased=False)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[14], line 10
      6 import numpy as np
      9 fig = plt.figure()
---> 10 ax = fig.gca(plot_surface='3d')
     12 # Make data.
     13 X = np.arange(-10, 10, 0.25)

TypeError: FigureBase.gca() got an unexpected keyword argument 'plot_surface'


<Figure size 2000x1000 with 0 Axes>
```

In [16]:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # Importing 3D projection

import numpy as np

# Create a figure and specify 3D projection explicitly
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Make data.
X = np.arange(-10, 10, 0.25)
Y = np.arange(-10, 10, 0.25)
X, Y = np.meshgrid(X, Y)

# Assuming some example values for b and b0
b = [1]  # Example value for b
b0 = 0   # Example value for b0

Z = 1 / (1 + np.exp(-(b[0] * X + b[0] * Y + b0)))

# Plot the surface
surf = ax.plot_surface(X, Y, Z, cmap='coolwarm', linewidth=0, antialia

# Customize the z axis.
ax.set_zlim(0, 1)
ax.zaxis.set_major_locator(plt.LinearLocator(10))
ax.zaxis.set_major_formatter(plt.FormatStrFormatter('%.02f'))

# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```



In [17]:
```
X
```

Out[17]:
```
array([[-10.  ,  -9.75,  -9.5 , ...,   9.25,   9.5 ,   9.75],
       [-10.  ,  -9.75,  -9.5 , ...,   9.25,   9.5 ,   9.75],
       [-10.  ,  -9.75,  -9.5 , ...,   9.25,   9.5 ,   9.75],
       ...,
       [-10.  ,  -9.75,  -9.5 , ...,   9.25,   9.5 ,   9.75],
       [-10.  ,  -9.75,  -9.5 , ...,   9.25,   9.5 ,   9.75],
       [-10.  ,  -9.75,  -9.5 , ...,   9.25,   9.5 ,   9.75]])
```

In [18]:
```
Y
```

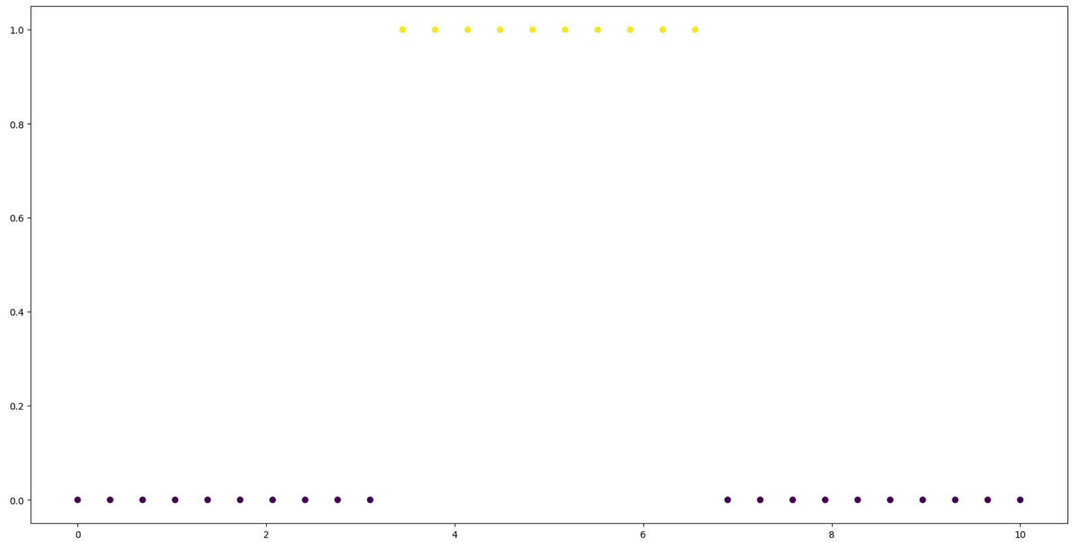
Out[18]:
```
array([[-10.  , -10.  , -10.  , ..., -10.  , -10.  , -10.  ],
       [ -9.75,  -9.75,  -9.75, ...,  -9.75,  -9.75,  -9.75],
       [ -9.5 ,  -9.5 ,  -9.5 , ...,  -9.5 ,  -9.5 ,  -9.5 ],
       ...,
       [  9.25,   9.25,   9.25, ...,   9.25,   9.25,   9.25],
       [  9.5 ,   9.5 ,   9.5 , ...,   9.5 ,   9.5 ,   9.5 ],
       [  9.75,   9.75,   9.75, ...,   9.75,   9.75,   9.75]])
```

What if the data doesn't really fit this pattern?

In [19]:
```
y = np.concatenate([np.zeros(10), np.ones(10), np.zeros(10)])
x = np.linspace(0, 10, len(y))
```

In [20]:
```
plt.scatter(x,y, c=y)
```

Out[20]: <matplotlib.collections.PathCollection at 0x200a6bf51e0>



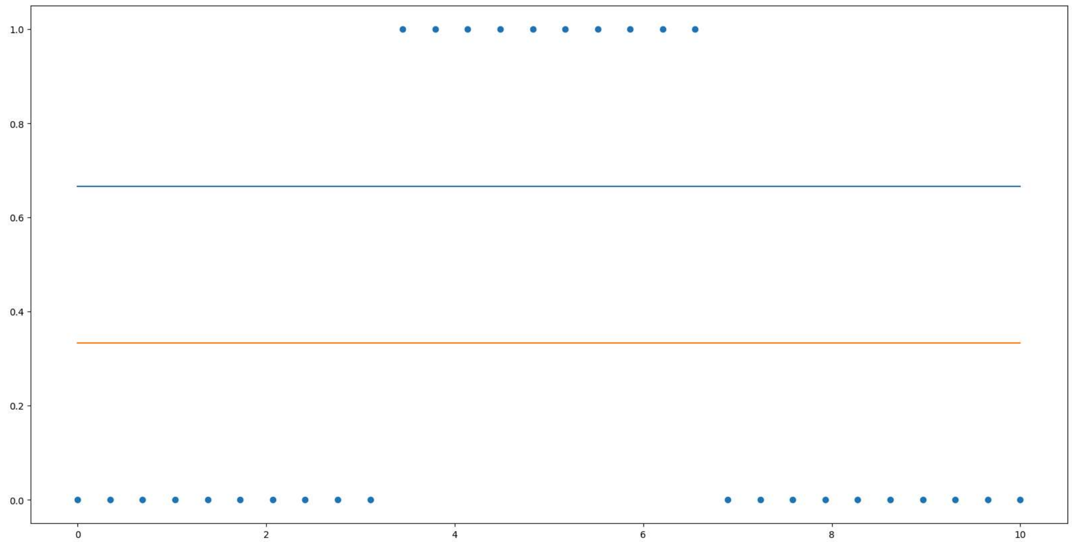
In [21]:
```
model.fit(x.reshape(-1, 1),y)
```

Out[21]: LogisticRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [22]:
```
plt.scatter(x,y)
plt.plot(x, model.predict_proba(x.reshape(-1, 1)))
```

Out[22]:
```
[&lt;matplotlib.lines.Line2D at 0x200a5356ef0>,
 <matplotlib.lines.Line2D at 0x200a5356f50>]
```



In [23]:
```
model1 = LogisticRegression()
model1.fit(x[:15].reshape(-1, 1),y[:15])
```

Out[23]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

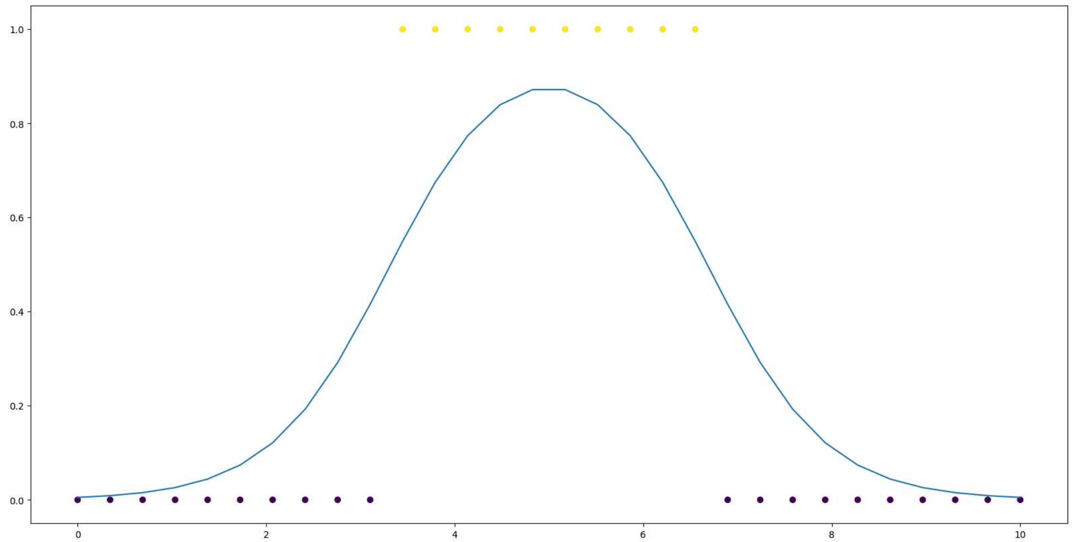
In [24]:
```
model2 = LogisticRegression()
model2.fit(x[15:].reshape(-1, 1),y[15:])
```

Out[24]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [25]:
```
plt.scatter(x,y, c=y)
plt.plot(x, model1.predict_proba(x.reshape(-1, 1))[:,1] * model2.predi
```

Out[25]: [<matplotlib.lines.Line2D at 0x200a648ca90>]



In [29]:
```
df = pd.read_csv('adult.data', index_col=False)
golden = pd.read_csv('adult.test', index_col=False)
```

In [30]:
```
from sklearn import preprocessing

enc = preprocessing.OrdinalEncoder()
```

In [31]:
```
transform_columns = ['sex', 'workclass', 'education', 'marital-status'
                     'occupation', 'relationship', 'race', 'sex',
                     'native-country', 'salary']
```

In [32]:
```
x = df.copy()

x[transform_columns] = enc.fit_transform(df[transform_columns])

golden['salary'] = golden.salary.replace(' <=50K.', ' <=50K').replace(
xt = golden.copy()

xt[transform_columns] = enc.transform(golden[transform_columns])
```

In [33]:
```
df.salary.unique()
```

Out[33]: array([' <=50K', ' >50K'], dtype=object)

In [34]:
```
golden.salary.replace(' <=50K.', ' <=50K').replace(' >50K.', ' >50K').
```

Out[34]: array([' <=50K', ' >50K'], dtype=object)

In [35]:
```
model.fit(preprocessing.scale(x.drop('salary', axis=1)), x.salary)
```

Out[35]: LogisticRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [36]:
```
pred = model.predict(preprocessing.scale(x.drop('salary', axis=1)))
pred_test = model.predict(preprocessing.scale(xt.drop('salary', axis=1
```

In [37]:
```
x.head()
```

Out[37]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	ra
0	39	7.0	77516	9.0	13	4.0	1.0	1.0	
1	50	6.0	83311	9.0	13	2.0	4.0	0.0	
2	38	4.0	215646	11.0	9	0.0	6.0	1.0	
3	53	4.0	234721	1.0	7	2.0	6.0	0.0	
4	28	4.0	338409	9.0	13	2.0	10.0	5.0	

In [38]:
```
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

In [39]:
```
accuracy_score(x.salary, pred)
```

Out[39]:
```
0.8250360861152913
```

In [40]:
```
confusion_matrix(x.salary, pred)
```

Out[40]:
```
array([[23300,  1420],
       [ 4277,  3564]], dtype=int64)
```

In [41]:
```
print(classification_report(x.salary, pred))
```

```
              precision    recall  f1-score   support

         0.0       0.84      0.94      0.89     24720
         1.0       0.72      0.45      0.56      7841

    accuracy                           0.83     32561
   macro avg       0.78      0.70      0.72     32561
weighted avg       0.81      0.83      0.81     32561
```

In [42]:
```
print(classification_report(xt.salary, pred_test))
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.94      0.89     12435
         1.0       0.70      0.45      0.55      3846

    accuracy                           0.82     16281
   macro avg       0.77      0.69      0.72     16281
weighted avg       0.81      0.82      0.81     16281
```

Assignment

1. Use your own dataset (`Heart.csv` is acceptable), create a train and a test set, and build 2 models: Logistic Regression and Decision Tree (shallow). Compare the test results using `classification_report` and `confusion_matrix`

In [68]:

```python
# Loading packages

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
```

In [64]:

```python
# PRE-PROCESSING

# Reading in dataset
heart = pd.read_csv('heart.csv', index_col=False)

# Dropping rows with missing values
heart = heart.dropna()

# Previewing dataset
heart.head()

# Creating binary variable for AHD - Adult Heart Disease
heart['AHD_yn'] = np.where(heart['AHD'] == 'Yes', 1, 0)

# Checking for missing values
heart['AHD_yn'].isna().sum()

# Getting value counts
heart['AHD_yn'].value_counts()

# Drop non-numeric columns (assuming 'AHD' and 'AHD_binary' are alread
numeric_columns = heart.select_dtypes(include=['number']).columns
heart_numeric = heart[numeric_columns]
```

In [69]:

```
# SPLITTING INTO TRAINING & TESTING DATASETS

# Define features (X) and target (y)
X = heart_numeric.drop(['AHD_yn'], axis=1)  # Features
y = heart_numeric['AHD_yn']  # Target variable

# Split data into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

# Scaling data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [74]:

```
# LOGISTIC REGRESSION

# Setting up logistic regression model
logreg_model = LogisticRegression()

# Training model
logreg_model.fit(X_train_scaled, y_train)

# Predicting test data
y_pred_logreg = logreg_model.predict(X_test_scaled)

# Producing accuracy score, classification report and confusion matrix
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred_logreg))
print("Classification Report:")
print(classification_report(y_test, y_pred_logreg))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_logreg))
```

```
Accuracy Score:
0.8
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.89      0.84        36
           1       0.80      0.67      0.73        24

    accuracy                           0.80        60
   macro avg       0.80      0.78      0.78        60
weighted avg       0.80      0.80      0.80        60

Confusion Matrix:
[[32  4]
 [ 8 16]]
```

In [84]:

```
# DECISION TREE - SHALLOW

# Creating shallow decision tree (2 nodes max)
dt_model = DecisionTreeClassifier(max_depth=2, random_state=42)

# Training model
dt_model.fit(X_train_scaled, y_train)

# Predicting test data
y_pred_dt = dt_model.predict(X_test_scaled)

# Calculating accuracy score
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"Decision Tree Accuracy Score: {accuracy_dt}")

# Printing classification report and confusion matrix for Decision Tre
print("\nDecision Tree:")
print(classification_report(y_test, y_pred_dt))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_dt))
```

```
Decision Tree Accuracy Score: 0.7333333333333333

Decision Tree:
              precision    recall  f1-score   support

           0       0.86      0.67      0.75        36
           1       0.62      0.83      0.71        24

    accuracy                           0.73        60
   macro avg       0.74      0.75      0.73        60
weighted avg       0.76      0.73      0.74        60


Confusion Matrix:
[[24 12]
 [ 4 20]]
```

Interpretation: In comparing the logistic regression model and the shallow decision tree, the logistic regression model performed better. Specifically, the logistic regression model achieved an accuracy score of 80%, while the shallow decision tree only reach 73.33%. Specifically, the decision tree model suffered in identifying positive cases of heart disease, as noted by its precision score of 0.62 for positive cases compared to 0.80 for negative cases. Conversely, the logistic regression model achieved a precision score of .80 for both positive and negative cases, suggesting that the model is equally effective at predicting both outcomes. Additionally, both models achieved similar F1 scores for patients without heart disease, however the logistic regression model performed slightly better in predicting positive cases of heart disease. In conclusion, the logistic regression model tends to perform better than the shallow decision tree due to its balance of precision and F1 scores across classes.

2. Repeat 1. but let the Decision Tree be much deeper to allow over-fitting. Compare the two models' test results again, and explain which is optimal

In [85]:

```
# DECISION TREE - DEEP

# Initialize Decision Tree model (DEEP)
dt_model_deep = DecisionTreeClassifier(max_depth=15, random_state=42)

# Fit the model on training data
dt_model_deep.fit(X_train_scaled, y_train)

# Predict on test data
y_pred_dt_deep = dt_model_deep.predict(X_test_scaled)

# Calculate accuracy score
accuracy_dt = accuracy_score(y_test, y_pred_dt_deep)
print(f"Decision Tree Accuracy Score: {accuracy_dt}")

# Print classification report and confusion matrix for Decision Tree
print("\nDecision Tree:")
print(classification_report(y_test, y_pred_dt_deep))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_dt_deep))
```

```
Decision Tree Accuracy Score: 0.8

Decision Tree:
              precision    recall  f1-score   support

           0       0.83      0.83      0.83        36
           1       0.75      0.75      0.75        24

    accuracy                           0.80        60
   macro avg       0.79      0.79      0.79        60
weighted avg       0.80      0.80      0.80        60


Confusion Matrix:
[[30  6]
 [ 6 18]]
```

Interpretation: Both the logistic regression model and the deep decision tree model have an accuracy score of 80% on the test data. In terms of precision, the logistic regression model achieved slightly higher scores for both classes compared to the decision tree, suggesting that the logistic regression is slightly more accurate in its positive predictions. However, the decision tree had higher recall for positive cases of heart disease (75%) than the logistic regression model (67%). This suggests that the deeper decision tree model is better at identifying patients with heart disease. Finally, in terms of F1 score, the logistic regression model achieved slightly higher scores for both classes (.01-.02 points higher).

Overall, both models performed well, however the decision tree may be best suited for this use case in that it performed slightly better at identifying patients with heart disease. Although the decision tree had slightly more false positives than the logistic regression model, it did succeed in identifying more true positive cases. Since identification of a positive case is essential to early intervention, and a false positive is arguably less risky than a false negative, I would suggest using the deep decision tree model in this specific outcome prediction case.