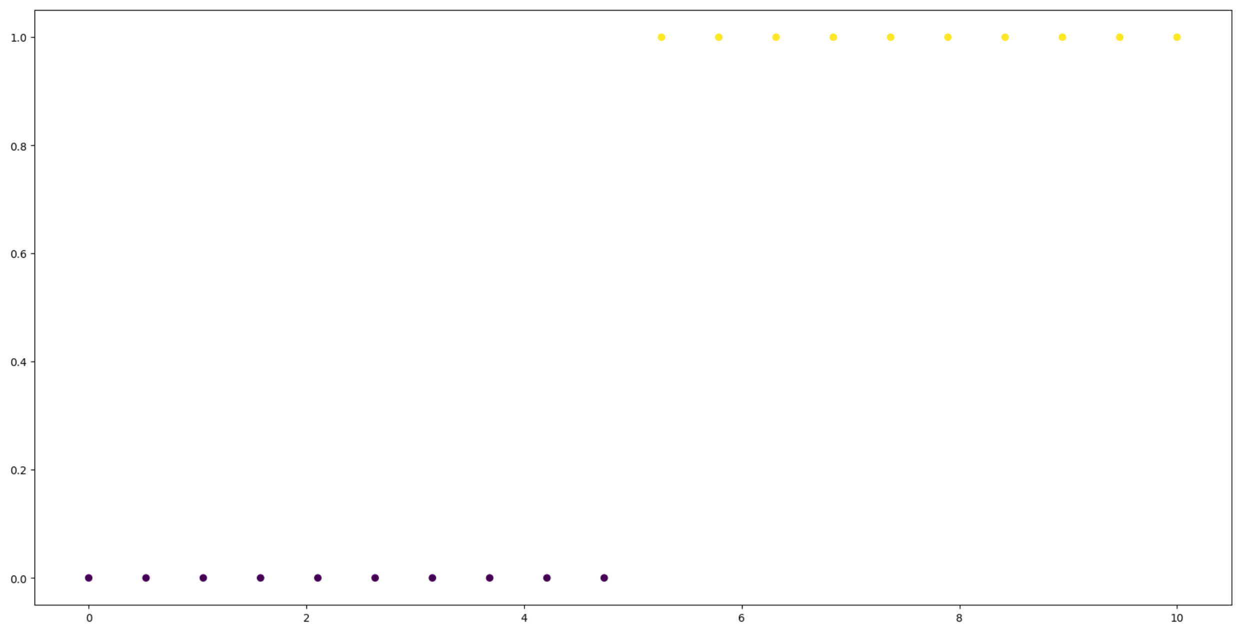
Assignment is at the bottom!

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 0x144d5a0a8f0>
```



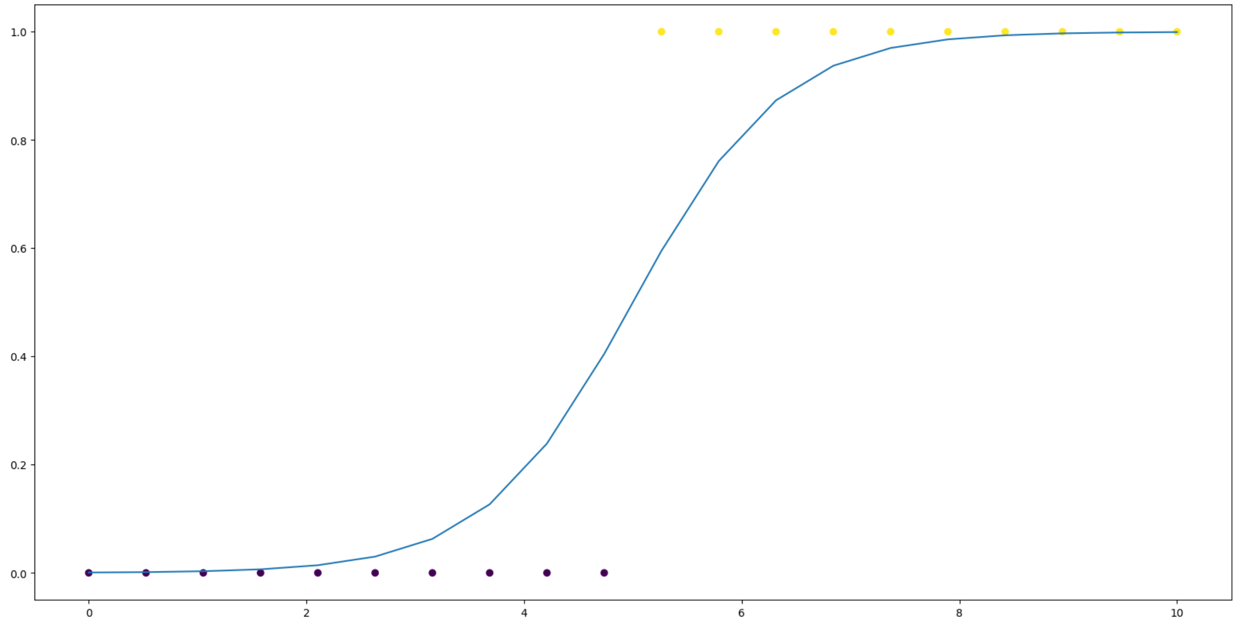
In [4]:
```
model = LogisticRegression()
```

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

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

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 0x144d83303a0>]
```

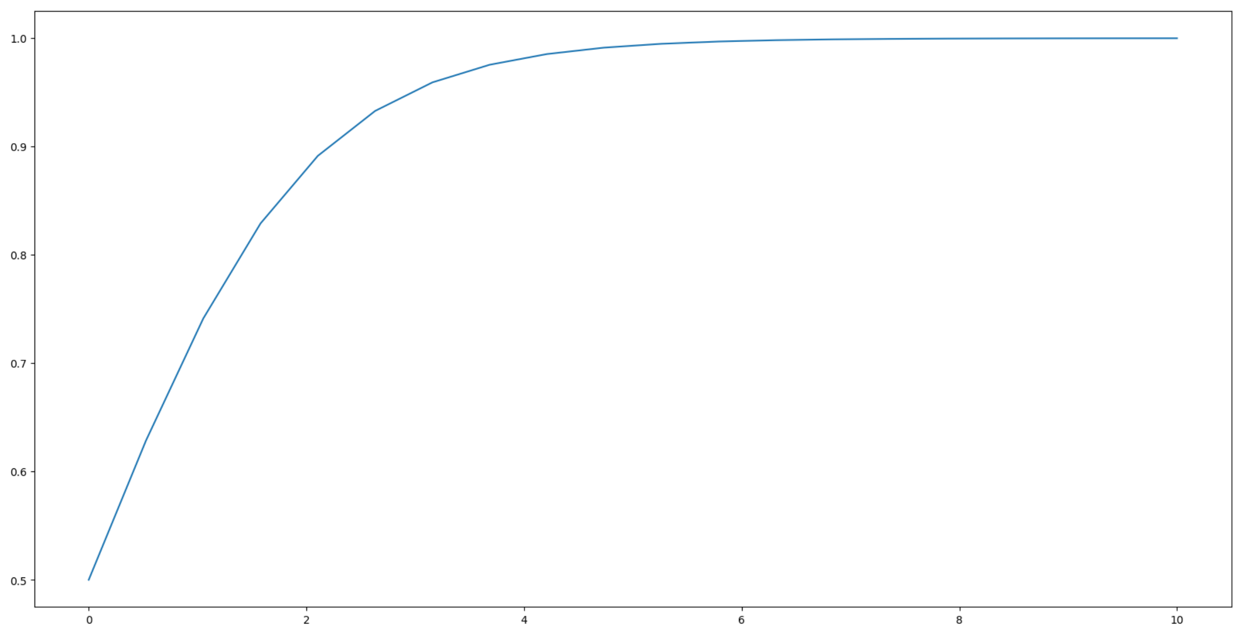


In [7]:
```
b, b0 = model.coef_, model.intercept_
model.coef_, model.intercept_
```

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

In [8]:
```
plt.plot(x, 1/(1+np.exp(-x)))
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x144d88a56c0>]
```

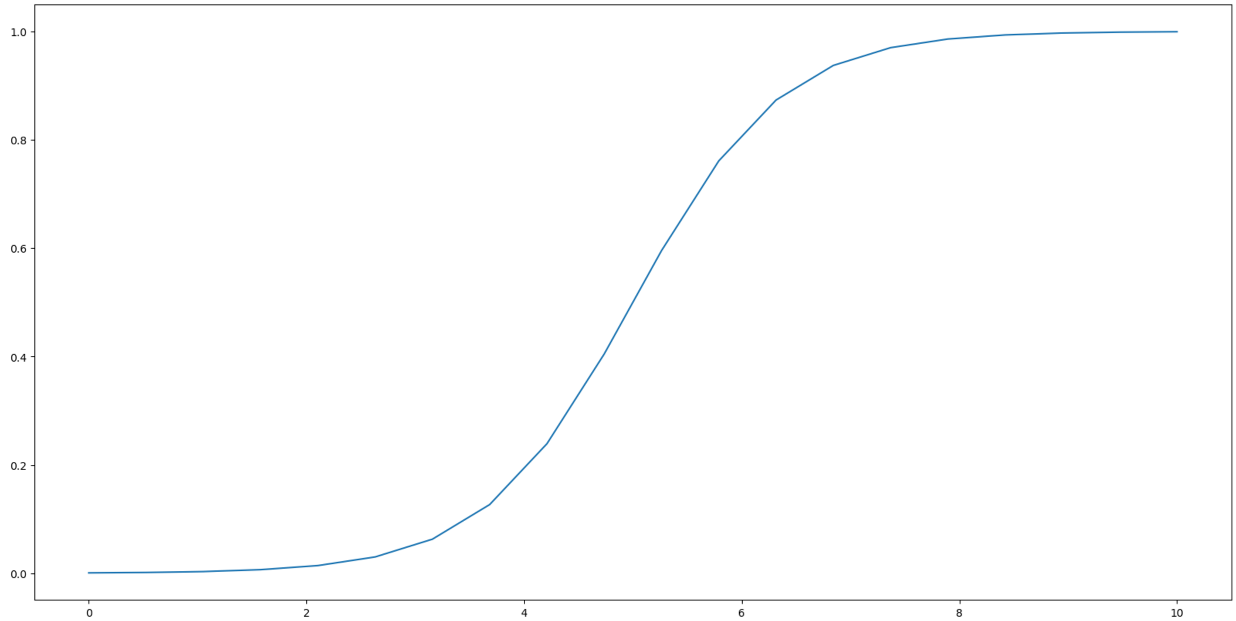


In [9]:
```
b
```

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

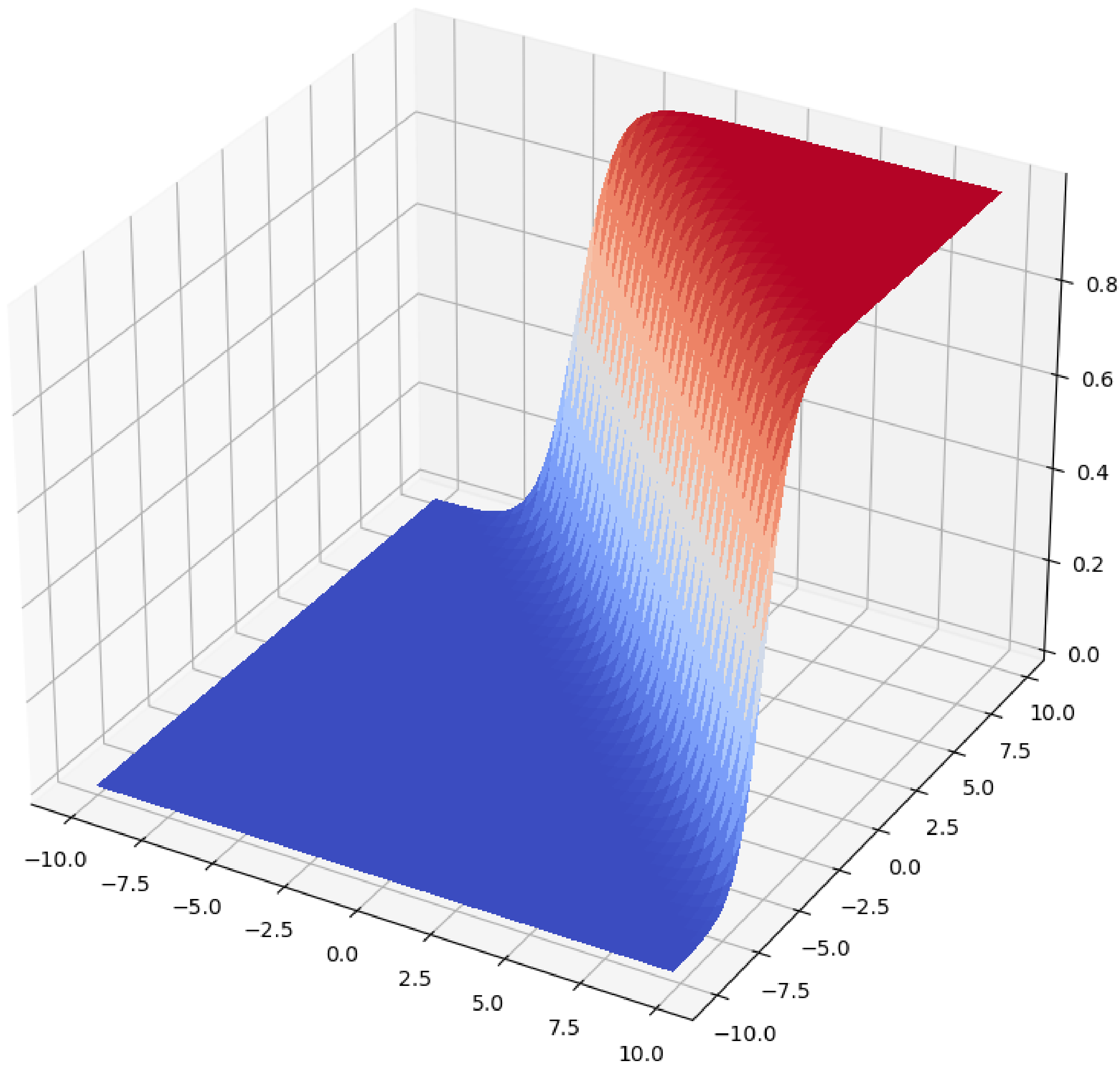
In [10]:
```
plt.plot(x, 1/(1+np.exp(-(b[0]*x +b0))))
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x144d8922b90>]
```



In [11]:

```
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.add_subplot(projection='3d') # gca wasn't working

# 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)
```



In [12]:
```
X
```

Out[12]:
```
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 [13]:
```
Y
```

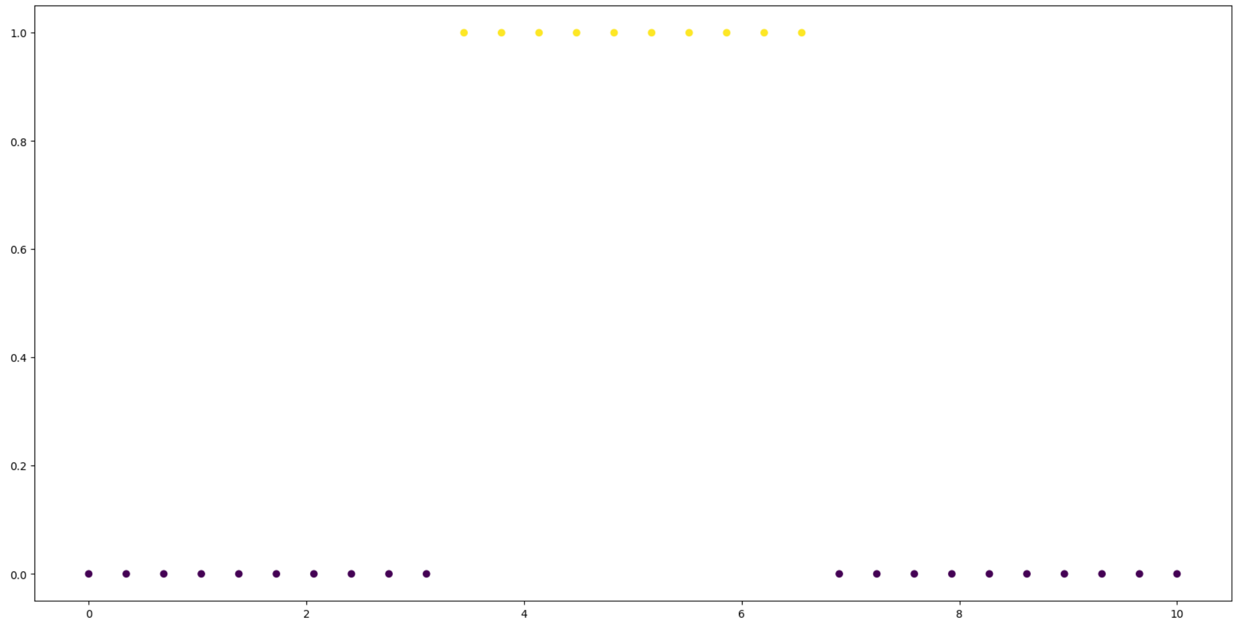
Out[13]:
```
array([[-10.  , -10.  , -10.  , ..., -10.  , -10.  , -10.  ],
       [ -9.75,  -9.75,  -9.75, ...,  -9.75,  -9.75,  -9.75],
       [ -9.5 ,  -9.5 ,  -9.5 , ...,  -9.5 ,  -9.5 ,  -9.5 ],
       ...,
       [  9.25,   9.25,   9.25, ...,   9.25,   9.25,   9.25],
       [  9.5 ,   9.5 ,   9.5 , ...,   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 [14]:
```
y = np.concatenate([np.zeros(10), np.ones(10), np.zeros(10)])
x = np.linspace(0, 10, len(y))
```

In [15]:
```
plt.scatter(x,y, c=y)
```

Out[15]:
```
<matplotlib.collections.PathCollection at 0x144d8bbdc90>
```

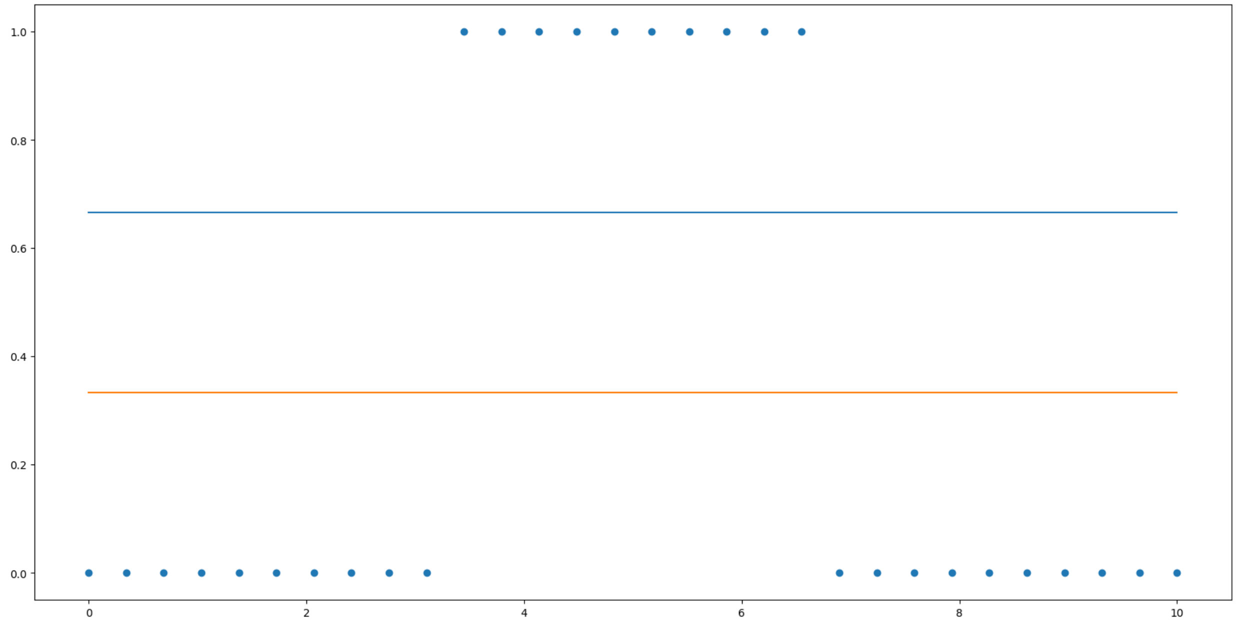


In [16]:
```
model.fit(x.reshape(-1, 1),y)
```

Out[16]:
```
▾ LogisticRegression
LogisticRegression()
```

In [17]:
```
plt.scatter(x,y)
plt.plot(x, model.predict_proba(x.reshape(-1, 1)))
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x144d8ce3760>,
 <matplotlib.lines.Line2D at 0x144d8ce37c0>]
```



In [18]:
```
model1 = LogisticRegression()
model1.fit(x[:15].reshape(-1, 1),y[:15])
```

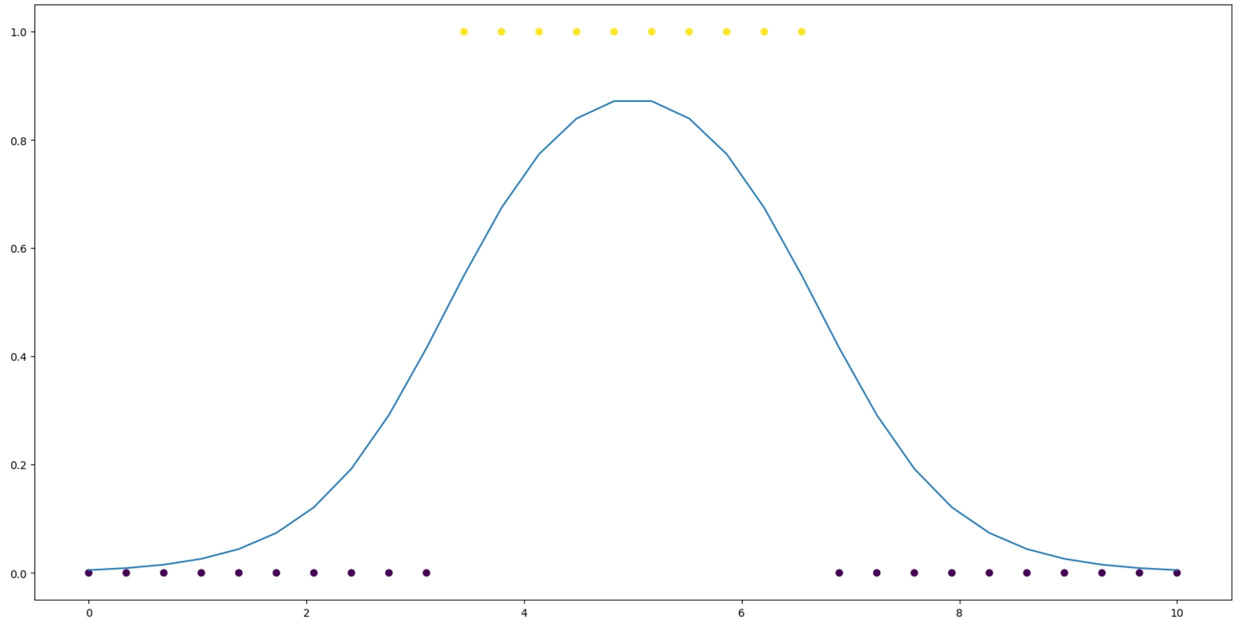
Out[18]:
```
▾ LogisticRegression
LogisticRegression()
```

In [19]:
```
model2 = LogisticRegression()
model2.fit(x[15:].reshape(-1, 1),y[15:])
```

Out[19]:
```
▾ LogisticRegression
LogisticRegression()
```

In [20]:
```
plt.scatter(x,y, c=y)
plt.plot(x, model1.predict_proba(x.reshape(-1, 1))[:,1] * model2.predict_proba(x.resha
```

Out[20]:
```
[<matplotlib.lines.Line2D at 0x144d5a9de70>]
```



In [21]:
```
df = pd.read_csv('../data/adult.data', index_col=False)
golden = pd.read_csv('../data/adult.test', index_col=False)
```

In [22]:
```
from sklearn import preprocessing

enc = preprocessing.OrdinalEncoder()
```

In [23]:
```
transform_columns = ['sex', 'workclass', 'education', 'marital-status',
                     'occupation', 'relationship', 'race', 'sex',
                     'native-country', 'salary']
```

In [24]:
```
x = df.copy()

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

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

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

In [25]:
```
df.salary.unique()
```

Out[25]:
```
array([' <=50K', ' >50K'], dtype=object)
```

In [26]:
```
golden.salary.replace(' <=50K.', ' <=50K').replace(' >50K.', ' >50K').unique()
```

Out[26]:
```
array([' <=50K', ' >50K'], dtype=object)
```

In [27]:
```
model.fit(preprocessing.scale(x.drop('salary', axis=1)), x.salary)
```

Out[27]:
```
▾ LogisticRegression
LogisticRegression()
```

In [28]:
```
pred = model.predict(preprocessing.scale(x.drop('salary', axis=1)))
pred_test = model.predict(preprocessing.scale(xt.drop('salary', axis=1)))
```

In [29]:
```
x.head()
```

Out[29]:

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

In [30]:
```
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

In [31]:
```
accuracy_score(x.salary, pred)
```

Out[31]:
```
0.8250360861152913
```

In [32]:
```
confusion_matrix(x.salary, pred)
```

Out[32]:
```
array([[23300,  1420],
       [ 4277,  3564]], dtype=int64)
```

In [33]:
```
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 [34]:
```
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`. Explain which algorithm is optimal

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 [35]:
```
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(font_scale=1.5)
import numpy as np

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


from sklearn.linear_model import LogisticRegression as Model
```

In [36]:
```
# Logistic Regression
heart = pd.read_csv('../data/Heart.csv') # Had to remove extra '/..'
heart = pd.DataFrame(heart)
heart.dropna(inplace=True)
heart['AHD'] = heart['AHD'].map({'Yes': 1, 'No': 0})
# feature_cols = ['RestBP', 'Chol']
# feature_columns = ['RestBP', 'Chol']
x = heart.drop(['AHD','ChestPain','Thal'], axis=1)
y = heart.AHD
heart['AHD'] = heart['AHD'].map({'Yes': 1, 'No': 0})
```

In [37]:
```
# Train & Test Sets
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2)
(
    xtrain.shape,
#     xtest.shape,
    ytrain.shape #,
#     ytest.shape
)
```

Out[37]:
```
((237, 12), (237,))
```

In [38]:
```
# from sklearn.preprocessing import StandardScaler # Trouble with Reshape, using a sin
# normalizer = StandardScaler()
# [['RestBP', 'Chol']] = normalizer.fit_transform(xtrain)
ytrain
```

Out[38]:
```
56     1
237    1
111    1
162    0
34     0
      ..
79     1
104    1
32     1
253    0
78     0
Name: AHD, Length: 237, dtype: int64
```

In [39]:
```
model = LogisticRegression()
model.fit(xtrain,ytrain)
lr_predictions = model.predict(xtest)
lr_predictions
pd.array(ytest)
```

```
C:\Users\Brett\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[39]:
```
<PandasArray>
[0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0,
 0, 0, 1, 1, 0, 1, 1, 1]
Length: 60, dtype: int64
```

In [40]:
```
# Classification Report
print(classification_report(pd.array(ytest), lr_predictions))
# Confusion Matrix
print(confusion_matrix(pd.array(ytest), lr_predictions))
```

```
              precision    recall  f1-score   support

           0       0.80      0.76      0.78        37
           1       0.64      0.70      0.67        23

    accuracy                           0.73        60
   macro avg       0.72      0.73      0.72        60
weighted avg       0.74      0.73      0.74        60

[[28  9]
 [ 7 16]]
```

In [41]:
```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
decisiontreemodel = DecisionTreeClassifier(criterion='entropy', max_depth= 3)
decisiontreemodel.fit(xtrain, ytrain)
```

```
dt_predictions = decisiontreemodel.predict(xtest)
dt_predictions
```

Out[41]:

```
array([0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1], dtype=int64)
```

In [42]:

```
# Classification Report
print(classification_report(pd.array(ytest), dt_predictions))
# Confusion Matrix
print(confusion_matrix(pd.array(ytest), dt_predictions))
```

```
              precision    recall  f1-score   support

           0       0.79      0.70      0.74        37
           1       0.59      0.70      0.64        23

    accuracy                           0.70        60
   macro avg       0.69      0.70      0.69        60
weighted avg       0.71      0.70      0.70        60

[[26 11]
 [ 7 16]]
```

In [43]:

```
# Comparison
# As seen in the classification reports, the logistic regression is superior in precis
```

In [44]:

```
# Decision Tree - Deeper
decisiontreemodel_deep = DecisionTreeClassifier(criterion='entropy', max_depth= 9999)
decisiontreemodel_deep.fit(xtrain, ytrain)
dt_predictions_deep = decisiontreemodel_deep.predict(xtest)
dt_predictions_deep
```

Out[44]:

```
array([1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1], dtype=int64)
```

In [45]:

```
# Classification Report
print(classification_report(pd.array(ytest), dt_predictions_deep))
# Confusion Matrix
print(confusion_matrix(pd.array(ytest), dt_predictions_deep))
```

```
              precision    recall  f1-score   support

           0       0.72      0.70      0.71        37
           1       0.54      0.57      0.55        23

    accuracy                           0.65        60
   macro avg       0.63      0.63      0.63        60
weighted avg       0.65      0.65      0.65        60

[[26 11]
 [10 13]]
```

In [ ]:

```
# By increasing the depth to a point where the decision tree overfits, we see decrease
# The model is too closely trained on the training set, and not adaptable to other dat
```

In [46]:

```
(
    decisiontreemodel.tree_.node_count,
```

```
    decisiontreemodel_deep.tree_.node_count
)
```

Out[46]: (15, 71)