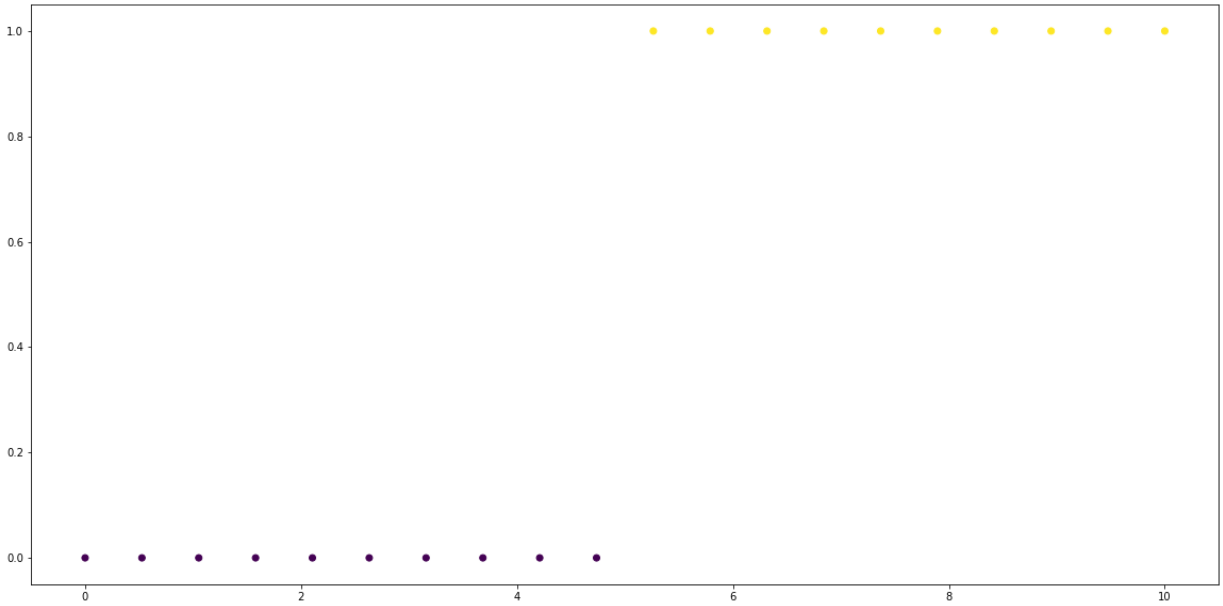
In [6]:
```
from sklearn.linear_model import LogisticRegression
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

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


from sklearn.linear_model import LogisticRegression as Model
```

In [7]:
```
y = np.concatenate([np.zeros(10), np.ones(10)])
x = np.linspace(0, 10, len(y))
```

In [8]:
```
plt.scatter(x, y, c=y)
```

Out[8]:
```
<matplotlib.collections.PathCollection at 0x15a294488e0>
```



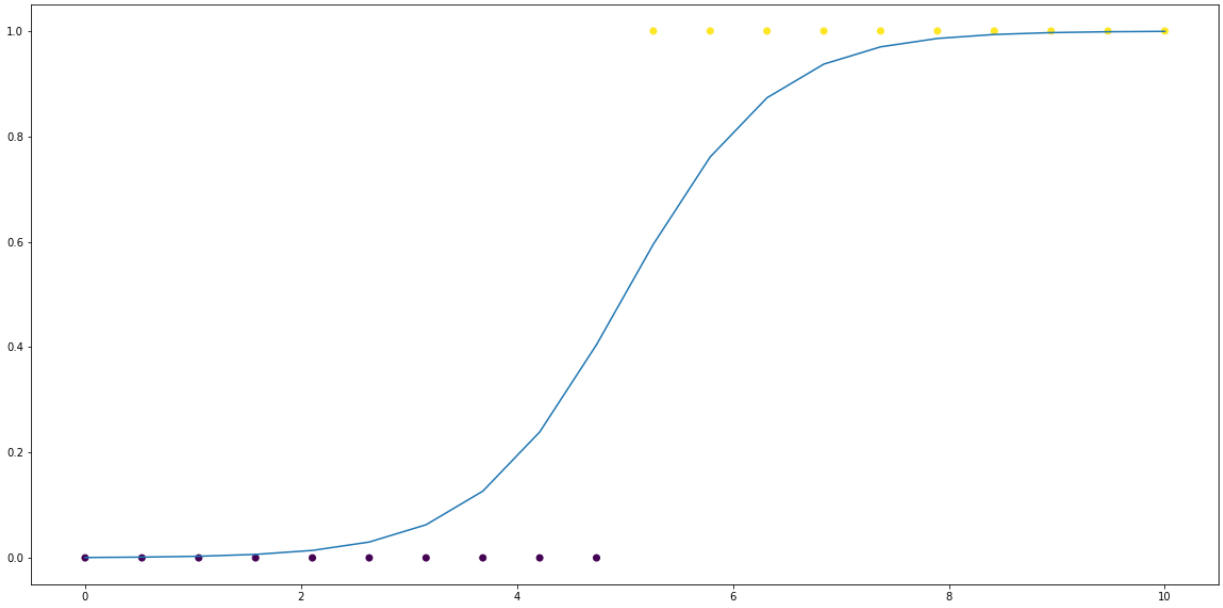
In [9]:
```
model = LogisticRegression()
```

In [10]:
```
model.fit(x.reshape(-1, 1),y)
```

Out[10]:
```
LogisticRegression()
```

In [11]:
```
plt.scatter(x,y, c=y)
plt.plot(x, model.predict_proba(x.reshape(-1, 1))[:,1])
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x15a29bc6d60>]
```

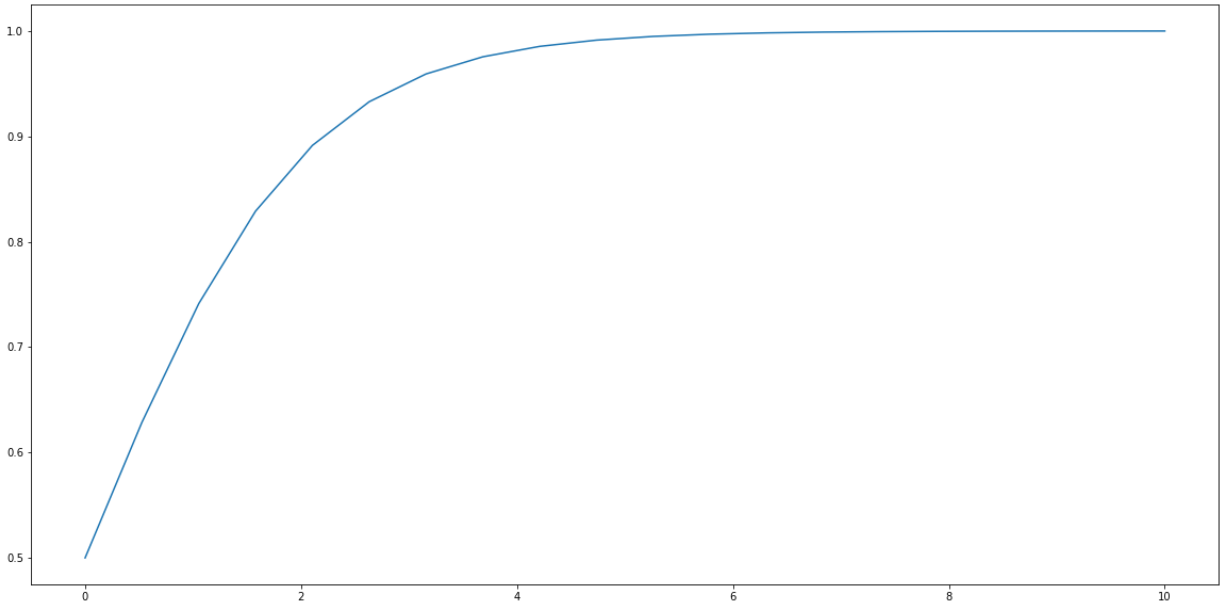


In [12]:
```
b, b0 = model.coef_, model.intercept_
model.coef_, model.intercept_
```

Out[12]:
```
(array([[1.46709085]]), array([-7.33542562]))
```

In [13]:
```
plt.plot(x, 1/(1+np.exp(-x)))
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x15a29c399d0>]
```

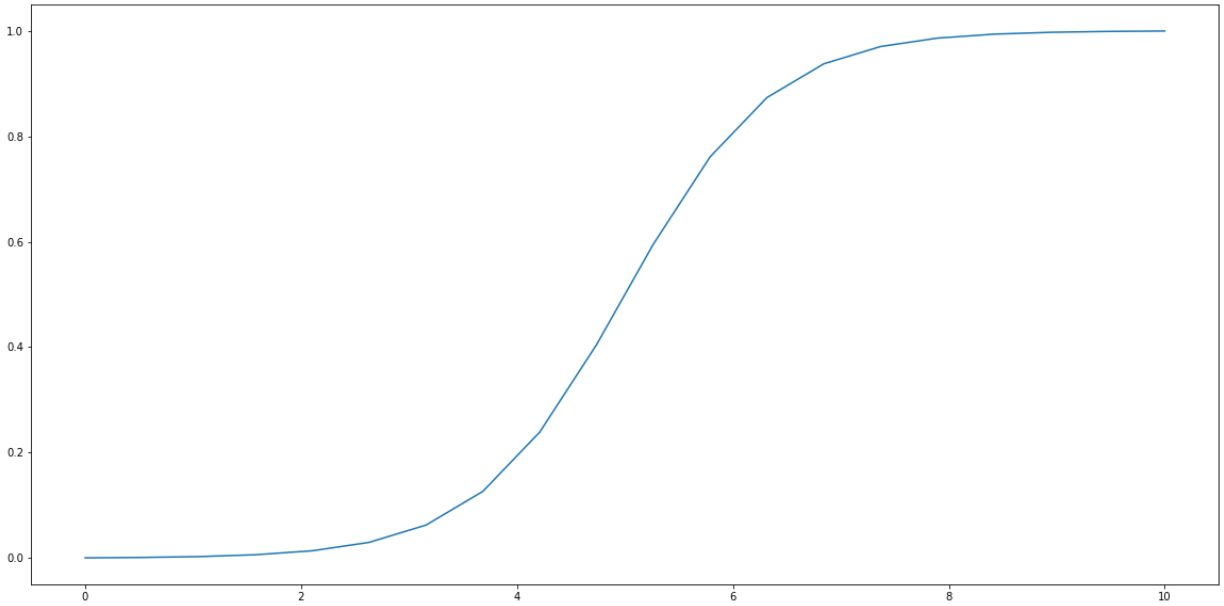


In [14]:
```
b
```

Out[14]:
```
array([[1.46709085]])
```

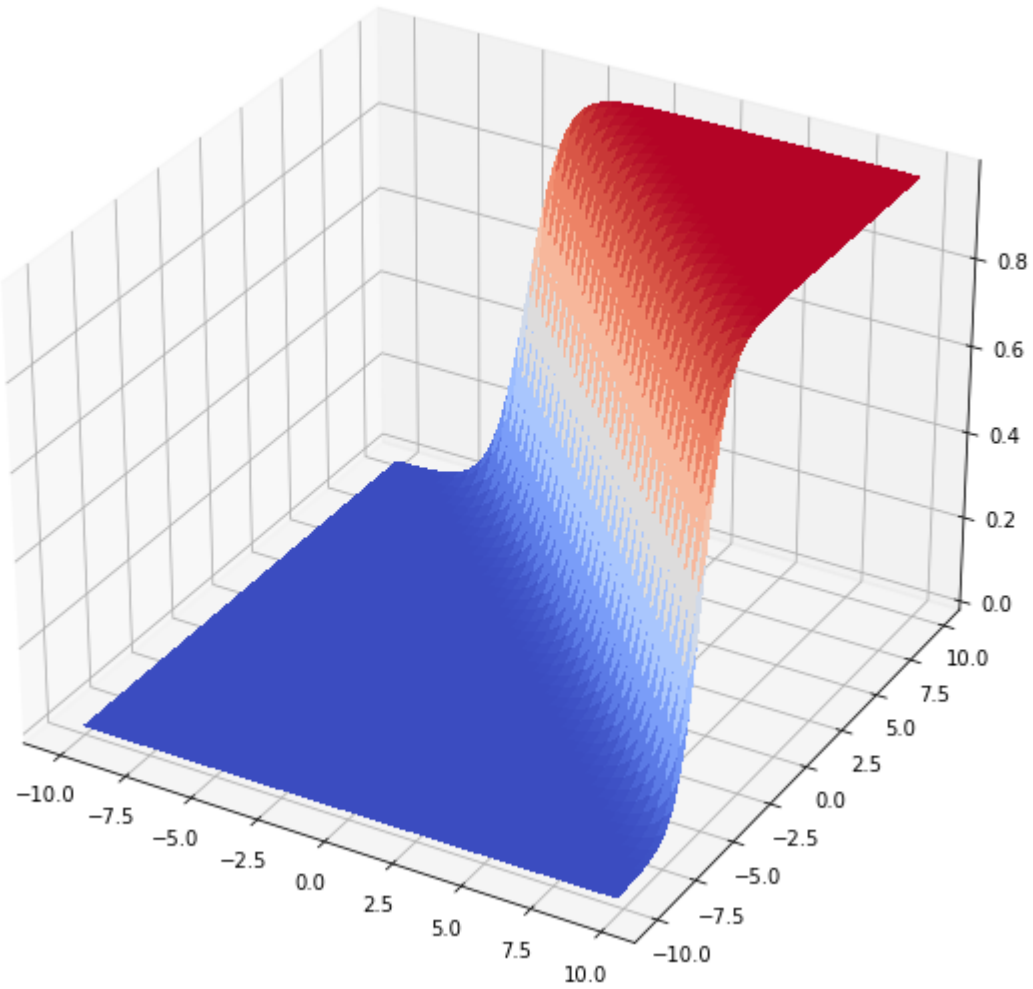
In [15]:
```
plt.plot(x, 1/(1+np.exp(-(b[0]*x +b0))))
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x15a29f4f070>]
```



In [16]:
```
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)
```

```
C:\Users\VivianHuo\AppData\Local\Temp\ipykernel_5644\4090702281.py:10: Matplotl
ibDeprecationWarning: Calling gca() with keyword arguments was deprecated in Ma
tplotlib 3.4. Starting two minor releases later, gca() will take no keyword arg
uments. The gca() function should only be used to get the current axes, or if n
o axes exist, create new axes with default keyword arguments. To create a new a
xes with non-default arguments, use plt.axes() or plt.subplot().
  ax = fig.gca(projection='3d')
```



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]])
```

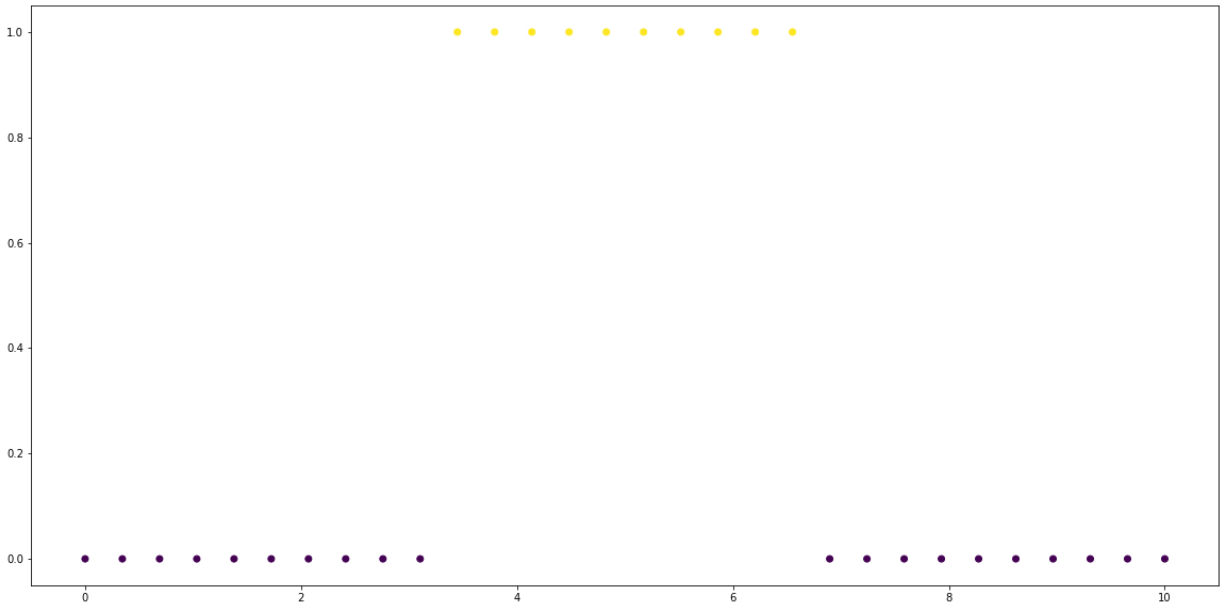
In [18]:
```
Y
```

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]])
```

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 0x15a2a63e9a0>
```

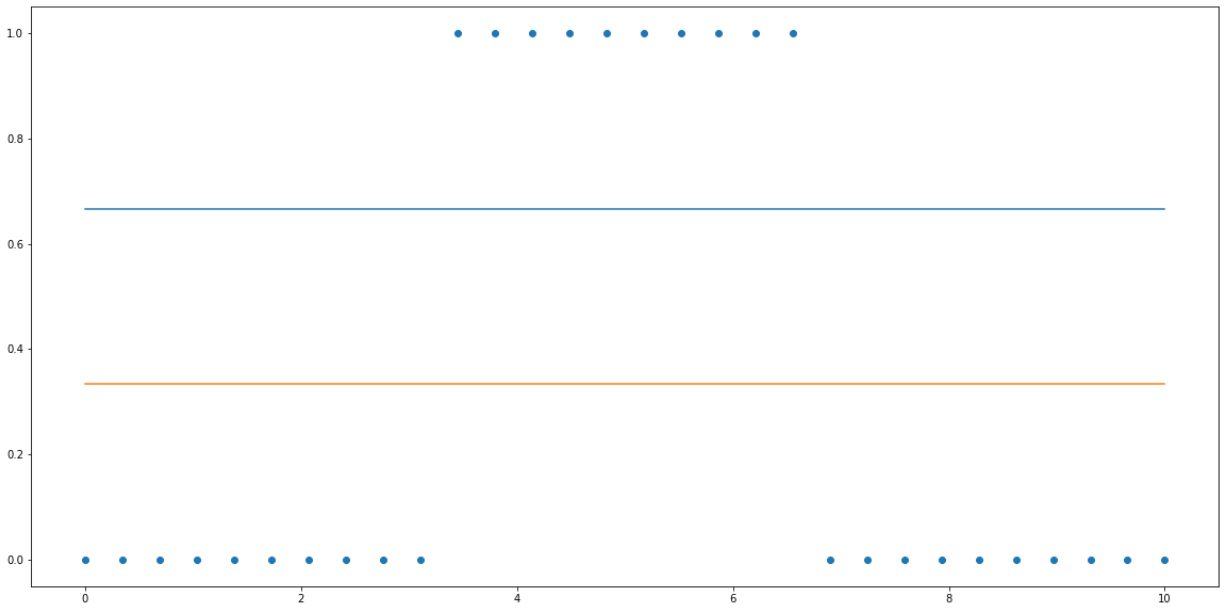


In [21]:
```
model.fit(x.reshape(-1, 1),y)
```

Out[21]:
```
LogisticRegression()
```

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

Out[22]:
```
[<matplotlib.lines.Line2D at 0x15a2a713dc0>,
 <matplotlib.lines.Line2D at 0x15a2a79f910>]
```



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

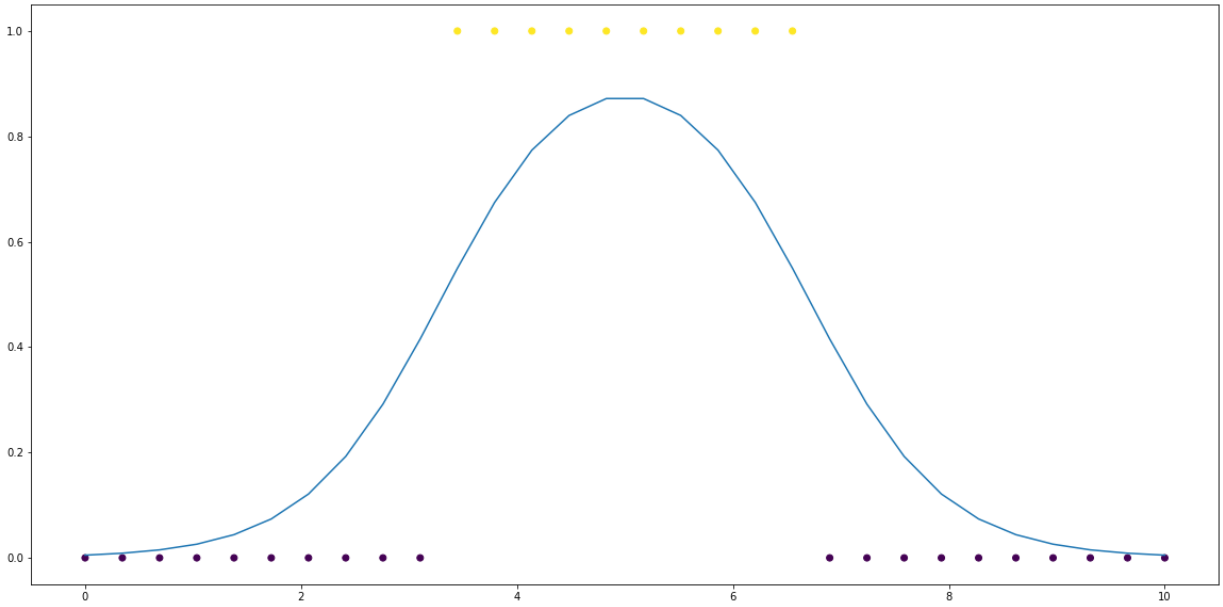
Out[23]:
```
LogisticRegression()
```

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

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

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

Out[25]:
```
[<matplotlib.lines.Line2D at 0x15a2aed6b50>]
```



In [26]:
```
import requests
import pandas as pd
import numpy as np
url = 'https://raw.githubusercontent.com/jhuaap/mlnn/master/data/adult.data'
res = requests.get(url, allow_redirects=True)
with open('adult','wb') as file:
    file.write(res.content)
df = pd.read_csv('adult', index_col=False)
```

In [27]:
```
url = 'https://raw.githubusercontent.com/jhuaap/mlnn/master/data/adult.data'
res = requests.get(url, allow_redirects=True)
with open('adult','wb') as file:
    file.write(res.content)
golden = pd.read_csv('adult', index_col=False)
```

In [28]:
```
from sklearn import preprocessing

enc = preprocessing.OrdinalEncoder()
```

In [29]:
```
transform_columns = ['sex', 'workclass', 'education', 'marital-status',
                     'occupation', 'relationship', 'race', 'sex',
                     'native-country', 'salary']
```

In [30]:
```
x = df.copy()

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

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

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

In [31]:
```
df.salary.unique()
```

Out[31]:
```
array([' <=50K', ' >50K'], dtype=object)
```

In [32]:
```
golden.salary.replace(' <=50K.', ' <=50K').replace(' >50K.', ' >50K').unique()
```

Out[32]:
```
array([' <=50K', ' >50K'], dtype=object)
```

In [33]:
```
model.fit(preprocessing.scale(x.drop('salary', axis=1)), x.salary)
```

Out[33]:
```
LogisticRegression()
```

In [34]:
```
pred = model.predict(preprocessing.scale(x.drop('salary', axis=1)))
pred_test = model.predict(preprocessing.scale(xt.drop('salary', axis=1)))
```

In [35]:
```
x.head()
```

Out[35]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	ca
0	39	7.0	77516	9.0	13	4.0	1.0	1.0	4.0	1.0	
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 [36]:
```
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

In [37]:
```
accuracy_score(x.salary, pred)
```

Out[37]:
```
0.8250360861152913
```

In [38]:
```
confusion_matrix(x.salary, pred)
```

Out[38]:
```
array([[23300,  1420],
       [ 4277,  3564]], dtype=int64)
```

In [39]:
```
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 [40]:
```
print(classification_report(xt.salary, pred_test))
```

```
              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 [47]:
```
#1 CREATE MY OWN DATA SET
```

In [24]:
```
import pandas as pd
from sklearn.linear_model import LogisticRegression as LR
from sklearn.tree import DecisionTreeClassifier as DTC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import numpy as np
data = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iri
                   names = ["SepalLength", "Sepal width","Petal length","Petal widt
data.head()
```

Out[24]:

	SepalLength	Sepal width	Petal length	Petal width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [25]:
```
data.Class.value_counts()
```

Out[25]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Class, dtype: int64
```

In [26]:
```
from sklearn.preprocessing import OrdinalEncoder
new_data = data.copy()

enc = OrdinalEncoder()
transform_columns = ['Class']

new_data[transform_columns] = enc.fit_transform(data[transform_columns])
new_data
```

Out[26]:

	SepalLength	Sepal width	Petal length	Petal width	Class
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2.0
146	6.3	2.5	5.0	1.9	2.0
147	6.5	3.0	5.2	2.0	2.0
148	6.2	3.4	5.4	2.3	2.0
149	5.9	3.0	5.1	1.8	2.0

150 rows × 5 columns

In [27]:
```
X = new_data.iloc[:, :4].to_numpy()  # features
y = new_data.iloc[:,4].to_numpy()  # label
X[:5, :], y[:5]
```

Out[27]:
```
(array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2]]),
 array([0., 0., 0., 0., 0.]))
```

In [28]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[28]:
```
((105, 4), (45, 4), (105,), (45,))
```

In [38]:
```
# Logistic Regression


model1 = LR()
model1.fit(X_train,y_train)
model1.coef_, model1.intercept_
```

Out[38]:
```
(array([[-0.40760271,  0.79142641, -2.30141071, -0.90942356],
        [ 0.48584558, -0.23935063, -0.13784672, -0.91884244],
        [-0.07824287, -0.55207578,  2.43925744,  1.828266  ]]),
 array([  9.38792831,   1.66727747, -11.05520578]))
```

In [39]:
```
y_pred_lr = model1.predict(X_test)
y_pred_lr
```

Out[39]:
```
array([1., 1., 2., 0., 1., 0., 0., 0., 1., 2., 1., 0., 2., 1., 0., 1., 2.,
       0., 2., 1., 1., 1., 1., 1., 2., 0., 2., 1., 2., 0., 1., 2., 0., 2.,
       2., 0., 0., 0., 0., 1., 0., 1., 0., 2., 2.])
```

In [40]:
```
print(classification_report(y_test, y_pred_lr))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        16
         1.0       1.00      0.94      0.97        17
         2.0       0.92      1.00      0.96        12

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

In [41]:
```
print(confusion_matrix(y_test, y_pred_lr))
```

```
[[16  0  0]
 [ 0 16  1]
 [ 0  0 12]]
```

In [42]:
```
# Decision Tree (shallow, max_depth=2)

model2 = DTC(criterion='entropy', max_depth=2, random_state=1234)
model2.fit(X_train, y_train)
```

Out[42]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=1234)
```

In [34]:
```
y_pred_dtc = model2.predict(X_test)
y_pred_dtc
```

Out[34]:
```
array([1., 1., 2., 0., 1., 0., 0., 0., 1., 2., 1., 0., 2., 1., 0., 1., 2.,
       0., 2., 1., 1., 1., 1., 1., 2., 0., 2., 1., 2., 0., 1., 2., 0., 1.,
       2., 0., 0., 0., 0., 1., 0., 1., 0., 2., 2.])
```

In [35]:
```
print(classification_report(y_test, y_pred_dtc))  # accuracy < LR's accuracy
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        16
         1.0       0.94      0.94      0.94        17
         2.0       0.92      0.92      0.92        12

    accuracy                           0.96        45
   macro avg       0.95      0.95      0.95        45
weighted avg       0.96      0.96      0.96        45
```

In [36]:
```
print(confusion_matrix(y_test, y_pred_dtc))
```

```
[[16  0  0]
 [ 0 16  1]
 [ 0  1 11]]
```

In [ ]:
```
# Q1 Comment: the desicion tree works better.
```

In [37]:
```
# Q2 Deeper Decision Tree

model3 = DTC(criterion='entropy', random_state=1234)
model3.fit(X_train, y_train)
y_pred_dtc_deeper = model3.predict(X_test)
print(classification_report(y_test, y_pred_dtc_deeper))  # accuracy = LR's accura
print(confusion_matrix(y_test, y_pred_dtc_deeper))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        16
         1.0       1.00      0.94      0.97        17
         2.0       0.92      1.00      0.96        12

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45

[[16  0  0]
 [ 0 16  1]
 [ 0  0 12]]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: