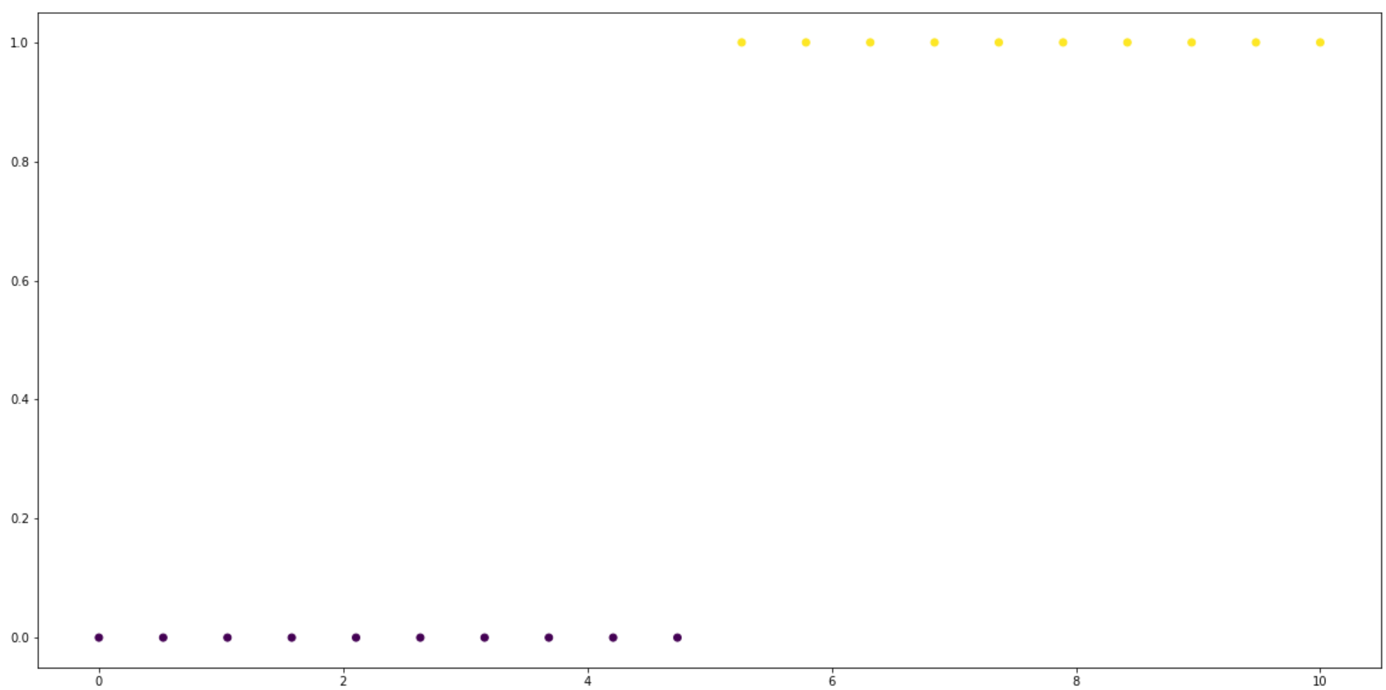
Assignment is at the bottom!

In [279…
```
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 [280…
```
y = np.concatenate([np.zeros(10), np.ones(10)])
x = np.linspace(0, 10, len(y))
```

In [281…
```
plt.scatter(x, y, c=y)
```

Out[281]: <matplotlib.collections.PathCollection at 0x7fd1f360d790>



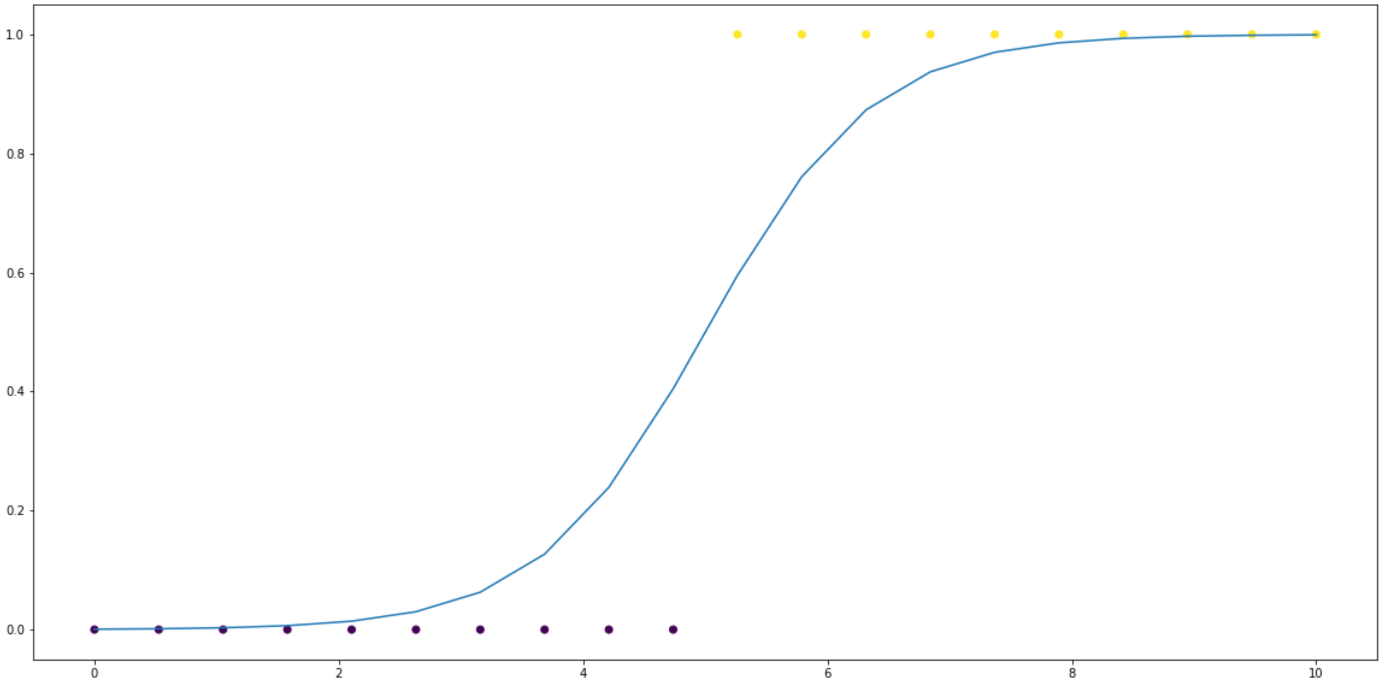
In [282…
```
model = LogisticRegression()
```

In [283…
```
model.fit(x.reshape(-1, 1),y)
```

Out[283]: LogisticRegression()

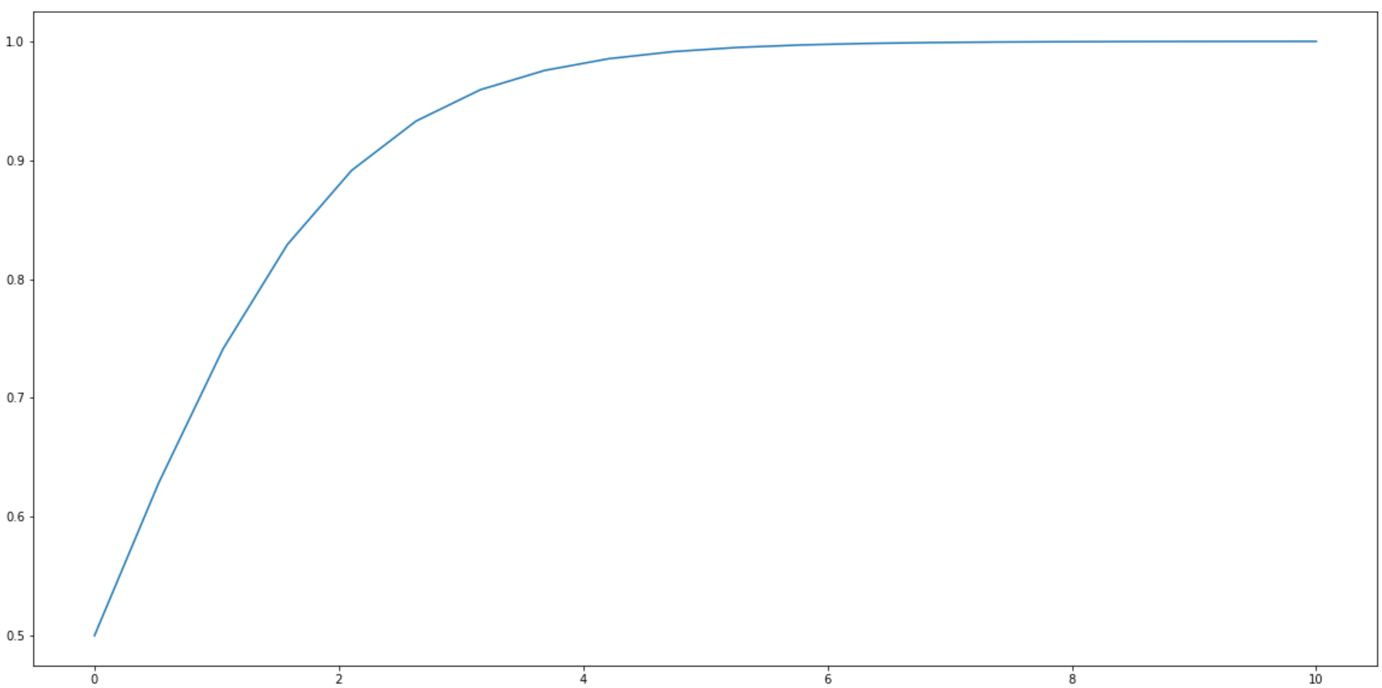
In [284…
```
plt.scatter(x,y, c=y)
plt.plot(x, model.predict_proba(x.reshape(-1, 1))[:,1])
```

Out[284]: [<matplotlib.lines.Line2D at 0x7fd1f251b7f0>]



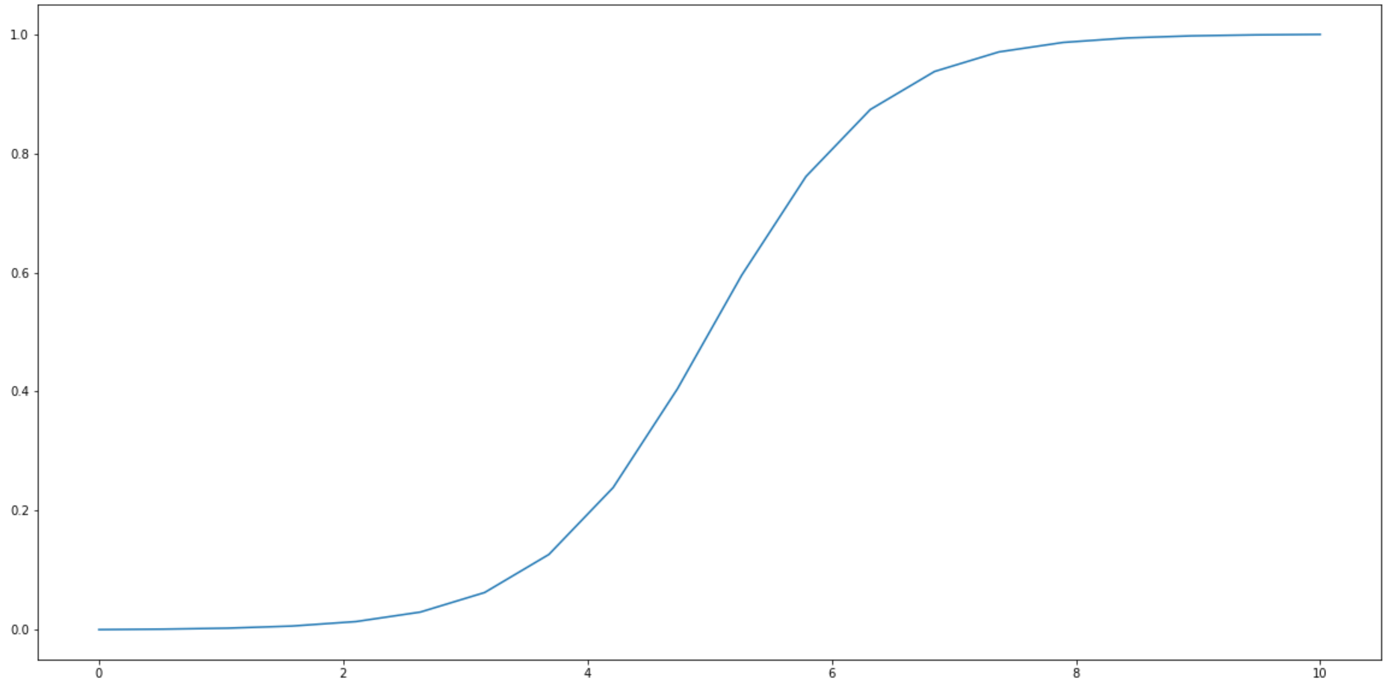
In [285…
```
b, b0 = model.coef_, model.intercept_
model.coef_, model.intercept_
```
Out[285]:
```
(array([[1.46709085]]), array([-7.33542562]))
```

In [286…
```
plt.plot(x, 1/(1+np.exp(-x)))
```
Out[286]:
```
[<matplotlib.lines.Line2D at 0x7fd251c99880>]
```



In [287…
```
b
```
Out[287]:
```
array([[1.46709085]])
```

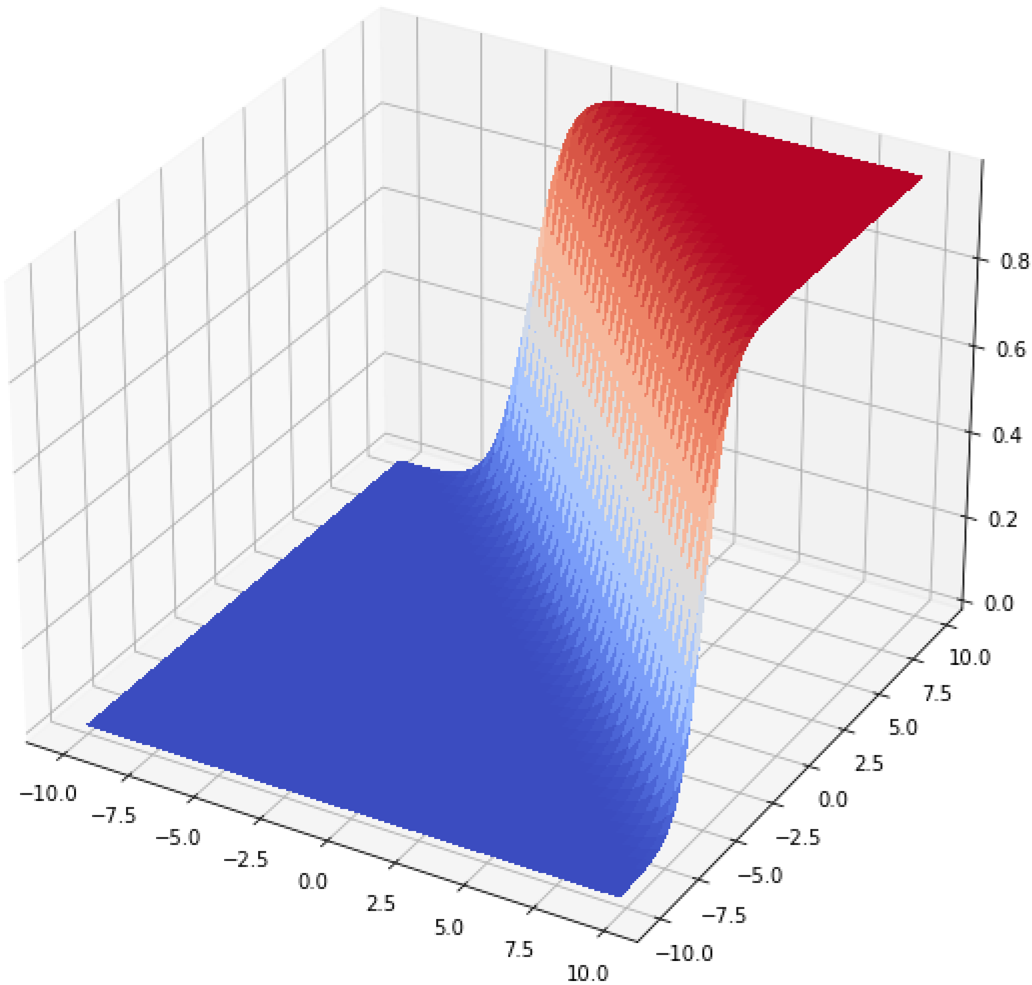
In [288…
```
plt.plot(x, 1/(1+np.exp(-(b[0]*x +b0))))
```
Out[288]:
```
[<matplotlib.lines.Line2D at 0x7fd1ea151ac0>]
```



In [289…

```
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)
```

```
/var/folders/x7/6srm3p515nl536_rx3q2gtfm0000gn/T/ipykernel_10910/290434025.py:10: Matplo
tlibDeprecationWarning: Calling gca() with keyword arguments was deprecated in Matplotli
b 3.4. Starting two minor releases later, gca() will take no keyword arguments. The gca
() function should only be used to get the current axes, or if no axes exist, create new
axes with default keyword arguments. To create a new axes with non-default arguments, us
e plt.axes() or plt.subplot().
  ax = fig.gca(projection='3d')
```



In [290…
```
X
```

Out[290]:
```
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 [291…
```
Y
```

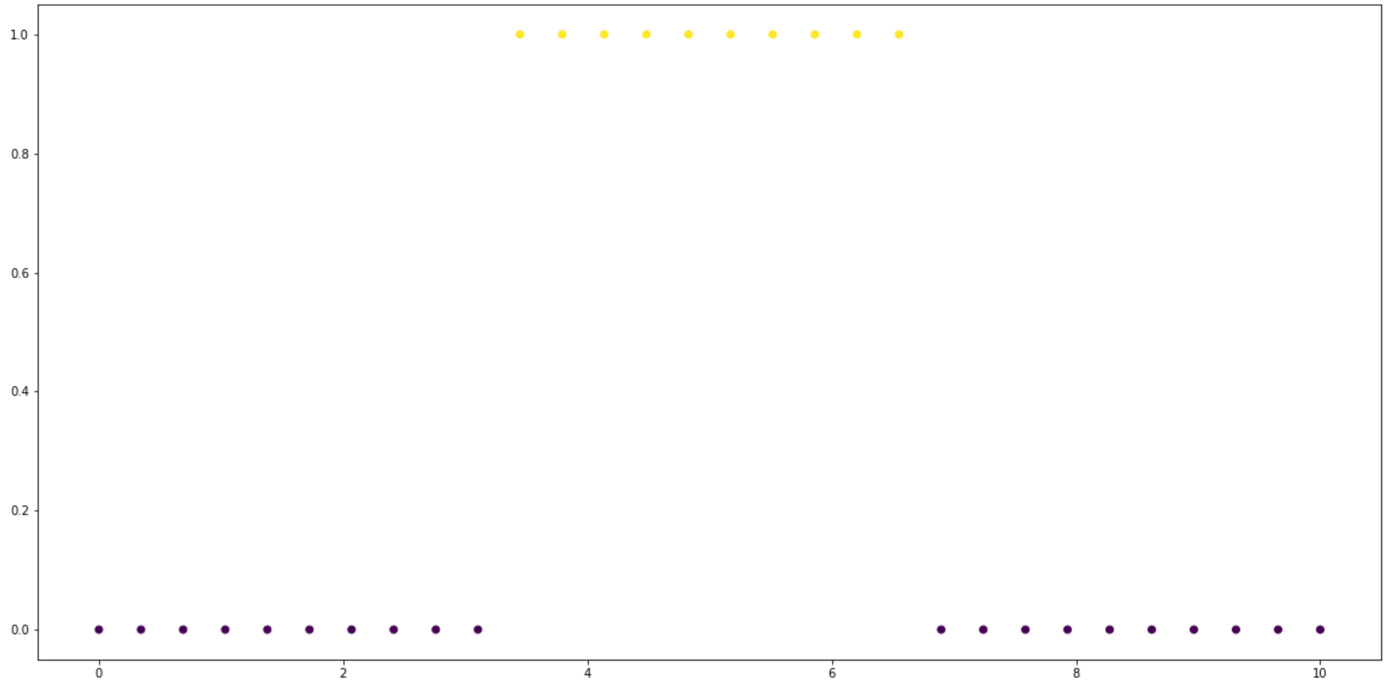
Out[291]:
```
array([[-10.  , -10.  , -10.  , ..., -10.  , -10.  , -10.  ],
       [ -9.75,  -9.75,  -9.75, ...,  -9.75,  -9.75,  -9.75],
       [ -9.5 ,  -9.5 ,  -9.5 , ...,  -9.5 ,  -9.5 ,  -9.5 ],
       ...,
       [  9.25,   9.25,   9.25, ...,   9.25,   9.25,   9.25],
       [  9.5 ,   9.5 ,   9.5 , ...,   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 [292…
```
y = np.concatenate([np.zeros(10), np.ones(10), np.zeros(10)])
x = np.linspace(0, 10, len(y))
```

In [293…
```
plt.scatter(x,y, c=y)
```

Out[293]:
```
<matplotlib.collections.PathCollection at 0x7fd221976190>
```



In [294…

```
model.fit(x.reshape(-1, 1),y)
```

Out[294]:

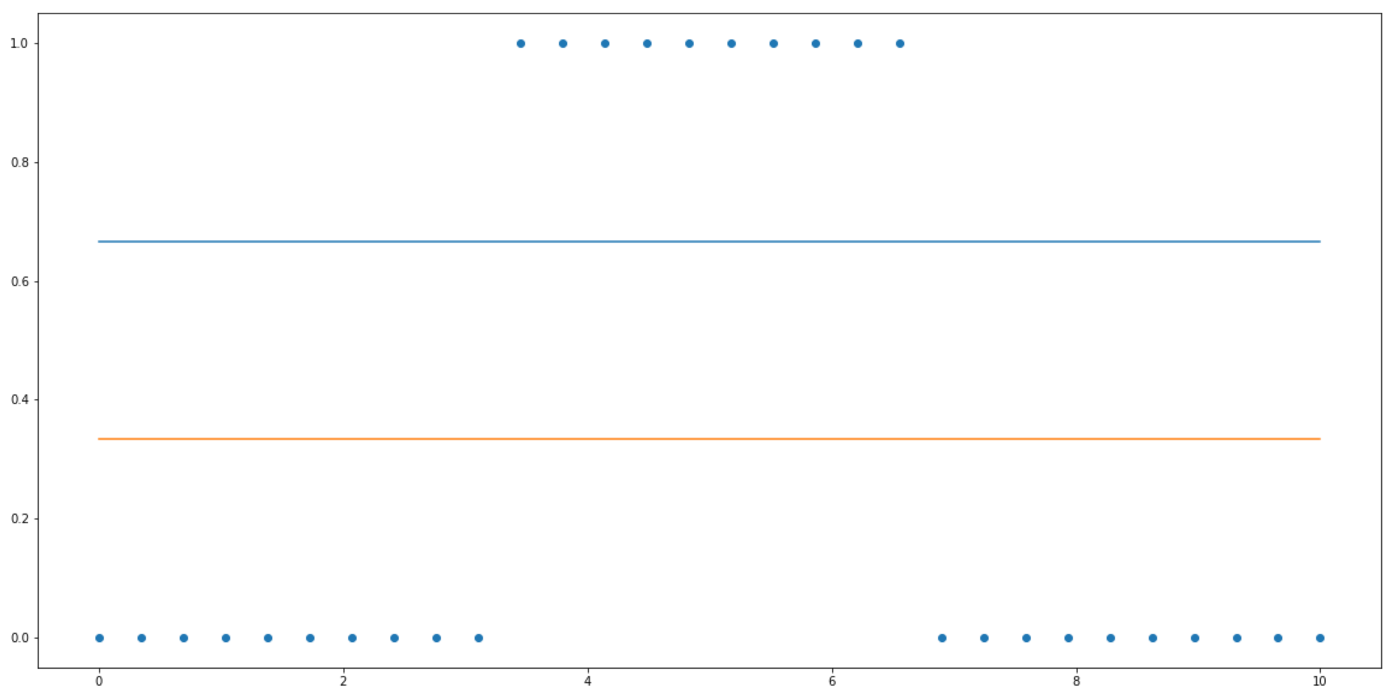
```
LogisticRegression()
```

In [295…

```
plt.scatter(x,y)
plt.plot(x, model.predict_proba(x.reshape(-1, 1)))
```

Out[295]:

```
[<matplotlib.lines.Line2D at 0x7fd1ea02cf10>,
 <matplotlib.lines.Line2D at 0x7fd1ea02cf70>]
```



In [296…

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

Out[296]:

```
LogisticRegression()
```

In [297…

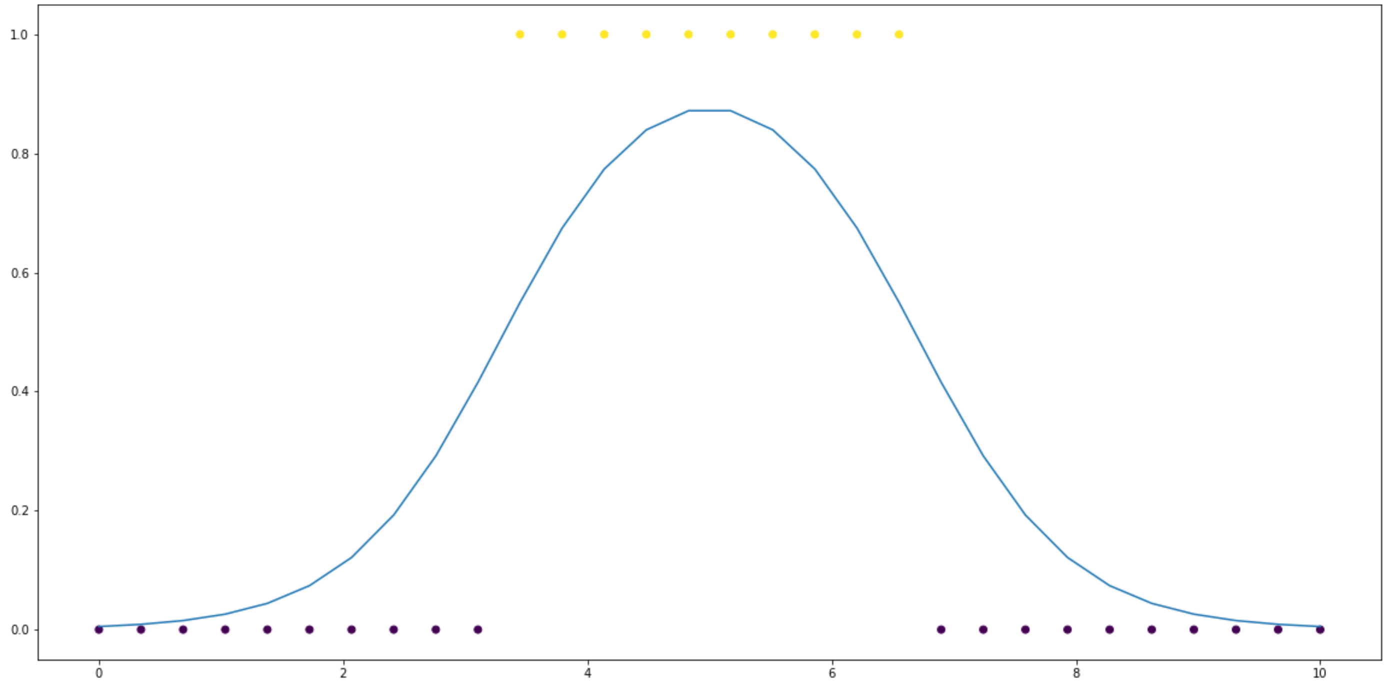
```
model2 = LogisticRegression()
model2.fit(x[15:].reshape(-1, 1),y[15:])
```

Out[297]:

```
LogisticRegression()
```

In [298…
```
plt.scatter(x,y, c=y)
plt.plot(x, model1.predict_proba(x.reshape(-1, 1))[:,1] * model2.predict_proba(x.reshape
```

Out[298]: [<matplotlib.lines.Line2D at 0x7fd1e97e2130>]



In [299…
```
df = pd.read_csv('../data/adult.data', index_col=False)
golden = pd.read_csv('../data/adult.test', index_col=False)
```

In [300…
```
from sklearn import preprocessing

enc = preprocessing.OrdinalEncoder()
```

In [301…
```
transform_columns = ['sex', 'workclass', 'education', 'marital-status',
                     'occupation', 'relationship', 'race', 'sex',
                     'native-country', 'salary']
```

In [302…
```
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 [303…
```
df.salary.unique()
```

Out[303]: array([' <=50K', ' >50K'], dtype=object)

In [304…
```
golden.salary.replace(' <=50K.', ' <=50K').replace(' >50K.', ' >50K').unique()
```

Out[304]: array([' <=50K', ' >50K'], dtype=object)

In [305…
```
model.fit(preprocessing.scale(x.drop('salary', axis=1)), x.salary)
```

Out[305]: LogisticRegression()

In [306…
```
pred = model.predict(preprocessing.scale(x.drop('salary', axis=1)))
pred_test = model.predict(preprocessing.scale(xt.drop('salary', axis=1)))
```

In [307…
```
x.head()
```

Out[307]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain
0	39	7.0	77516	9.0	13	4.0	1.0	1.0	4.0	1.0	2174
1	50	6.0	83311	9.0	13	2.0	4.0	0.0	4.0	1.0	0
2	38	4.0	215646	11.0	9	0.0	6.0	1.0	4.0	1.0	0
3	53	4.0	234721	1.0	7	2.0	6.0	0.0	2.0	1.0	0
4	28	4.0	338409	9.0	13	2.0	10.0	5.0	2.0	0.0	0

In [308…
```
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

In [309…
```
accuracy_score(x.salary, pred)
```

Out[309]:
```
0.8250360861152913
```

In [310…
```
confusion_matrix(x.salary, pred)
```

Out[310]:
```
array([[23300,  1420],
       [ 4277,  3564]])
```

In [311…
```
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 [312…
```
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 (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`. Which algorithm is superior?

2. Repeat 1. but let the Decision Tree be much deeper to allow over-fitting. Compare the two models' test results again, and explain why it's superior

1. Use your own dataset (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. Which algorithm is superior?

In [313…
```
col_names = ['id','clump thickness','unif_cell size','unif_cell shape','marg_adh','sing_
             'bare nuc','bland chr','normal_nuc','mitoses','class']
bcw = pd.read_csv('../add_data/breast-cancer-wisconsin.data',names=col_names,
                  index_col='id')
bcw.head()
```

Out[313]:

id	clump thickness	unif_cell size	unif_cell shape	marg_adh	sing_epi_cell size	bare nuc	bland chr	normal_nuc	mitoses	cla
1000025	5	1	1	1	2	1	3	1	1	
1002945	5	4	4	5	7	10	3	2	1	
1015425	3	1	1	1	2	2	3	1	1	
1016277	6	8	8	1	3	4	3	7	1	
1017023	4	1	1	3	2	1	3	1	1	

For this week's assignment, I chose to use UCI's Breast Cancer Wisconsin dataset. I chose the "class" column, which denotes whether the patient's breast mass was benign or malignant (0 or 1), as the dependent variable of the characteristics of the mass.

In [314…
```
bcw['bare nuc'].value_counts()
```

Out[314]:
```
1     402
10    132
2      30
5      30
3      28
8      21
4      19
?      16
9       9
7       8
6       4
Name: bare nuc, dtype: int64
```

In [315…
```
bcw['bare nuc'].replace(to_replace='?',value=1,inplace=True)
bcw['bare nuc'] = bcw['bare nuc'].astype(int)
bcw['bare nuc'].value_counts()
```

Out[315]:
```
1     418
10    132
```

```
2        30
5        30
3        28
8        21
4        19
9         9
7         8
6         4
Name: bare nuc, dtype: int64
```

In [316…

```
bcw['class'] = np.where(bcw['class'] == 2, 0, 1)
bcw.head()
```

Out[316]:

id	clump thickness	unif_cell size	unif_cell shape	marg_adh	sing_epi_cell size	bare nuc	bland chr	normal_nuc	mitoses	cla
1000025	5	1	1	1	2	1	3	1	1	
1002945	5	4	4	5	7	10	3	2	1	
1015425	3	1	1	1	2	2	3	1	1	
1016277	6	8	8	1	3	4	3	7	1	
1017023	4	1	1	3	2	1	3	1	1	

In [317…

```
bcw.dtypes
```

Out[317]:

```
clump thickness        int64
unif_cell size         int64
unif_cell shape        int64
marg_adh               int64
sing_epi_cell size     int64
bare nuc               int64
bland chr              int64
normal_nuc             int64
mitoses                int64
class                  int64
dtype: object
```

In [318…

```
from sklearn.model_selection import train_test_split
x = bcw.drop(['class'],axis=1)
y = bcw['class']
x_test, x_train, y_test, y_train = train_test_split(x, y, test_size=0.3)
x_test.dtypes
```

Out[318]:

```
clump thickness        int64
unif_cell size         int64
unif_cell shape        int64
marg_adh               int64
sing_epi_cell size     int64
bare nuc               int64
bland chr              int64
normal_nuc             int64
mitoses                int64
dtype: object
```

In [319…

```
from sklearn.tree import DecisionTreeClassifier
lr = LogisticRegression()
dt = DecisionTreeClassifier(criterion='entropy',max_depth=3)
lr.fit(x_train, y_train)
dt.fit(x_train, y_train)
lrpred1 = lr.predict(x_test)
dtpred1 = dt.predict(x_test)
```

In [320…
```
confusion_matrix(y_test, lrpred1)
```
Out[320]:
```
array([[310,   8],
       [ 14, 157]])
```
In [321…
```
confusion_matrix(y_test, dtpred1)
```
Out[321]:
```
array([[302,  16],
       [ 11, 160]])
```
In [322…
```
print(classification_report(y_test, lrpred1))
```
```
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       318
           1       0.95      0.92      0.93       171

    accuracy                           0.96       489
   macro avg       0.95      0.95      0.95       489
weighted avg       0.95      0.96      0.95       489
```
In [323…
```
print(classification_report(y_test, dtpred1))
```
```
              precision    recall  f1-score   support

           0       0.96      0.95      0.96       318
           1       0.91      0.94      0.92       171

    accuracy                           0.94       489
   macro avg       0.94      0.94      0.94       489
weighted avg       0.95      0.94      0.94       489
```
According to the confusion matrices, the logistic regression model has a lower error rate than the decision tree model (22 inaccurate predictions for the logistic regression, vice 27 for the decision tree. With more performance statistics in the classification reports, the logistic regression has higher macro and weighted averages for all but precision weighted average.

2. Repeat 1. but let the Decision Tree be much deeper to allow over-fitting. Compare the two models' test results again, and explain why it's superior

In [324…
```
dt2 = DecisionTreeClassifier(criterion='entropy',max_depth=15)
dt2.fit(x_train, y_train)
dtpred2 = dt2.predict(x_test)
confusion_matrix(y_test, dtpred2)
```
Out[324]:
```
array([[300,  18],
       [ 15, 156]])
```
In [325…
```
print(classification_report(y_test, dtpred2))
```
```
              precision    recall  f1-score   support

           0       0.95      0.94      0.95       318
           1       0.90      0.91      0.90       171

    accuracy                           0.93       489
   macro avg       0.92      0.93      0.93       489
weighted avg       0.93      0.93      0.93       489
```

With a much deeper decision tree, the logistic regression continues to outperform. At max_depth of 15, the decision tree even slightly underperformed against the shallower decision tree, with averages dropping across the board and 6 more inaccurate predictions.