

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

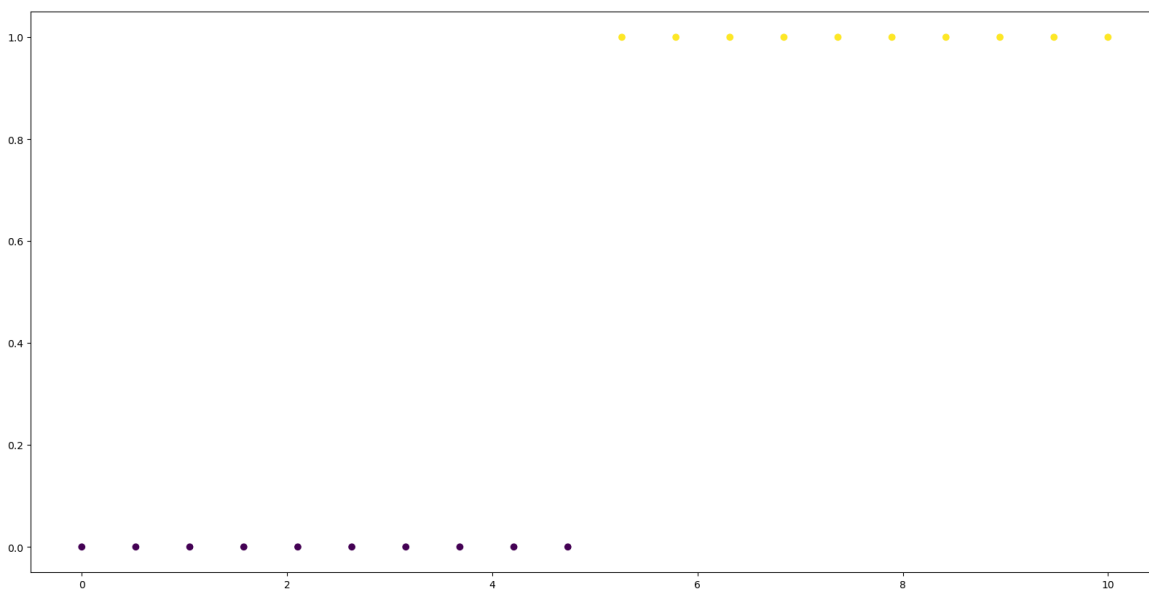
from pylab import rcParams
rcParams['figure.figsize'] = 20, 10

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 0x128b301f0>



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

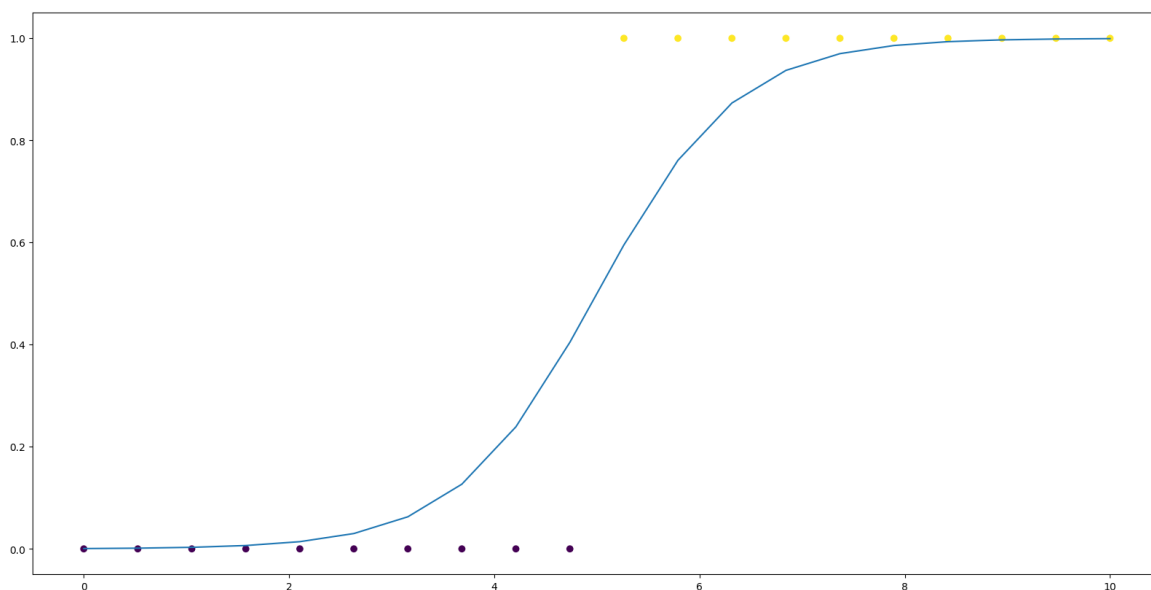
```
In [5]: model.fit(x.reshape(-1, 1), y)
```

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 0x129de95b0>]
```

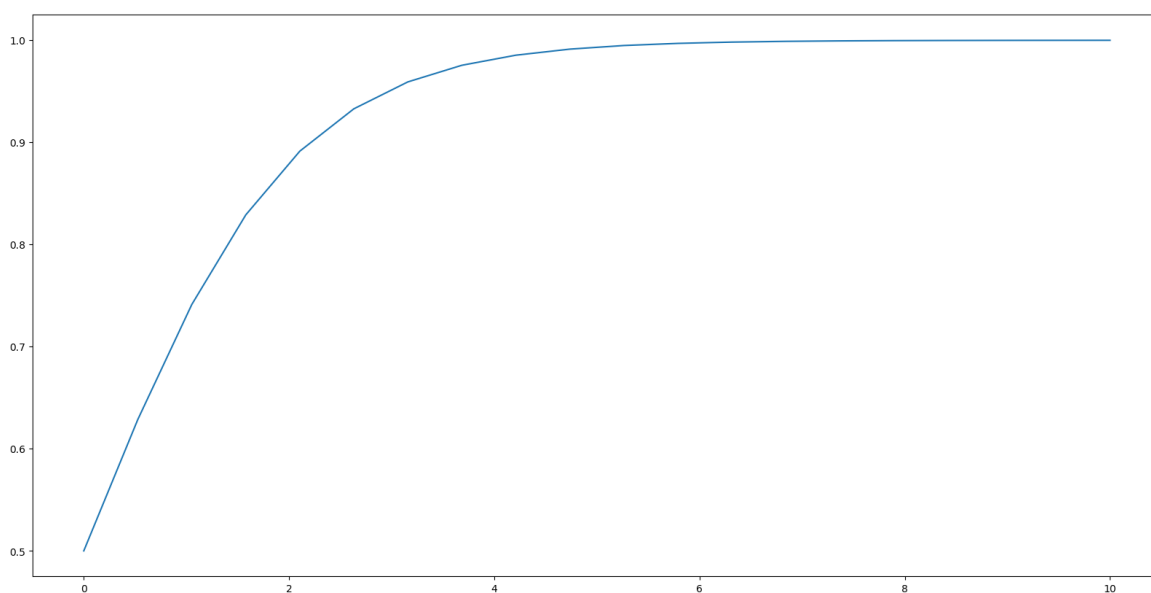


```
In [7]: b, b0 = model.coef_, model.intercept_
model.coef_, model.intercept_
```

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

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

```
Out[8]: [<matplotlib.lines.Line2D at 0x12a349dc0>]
```

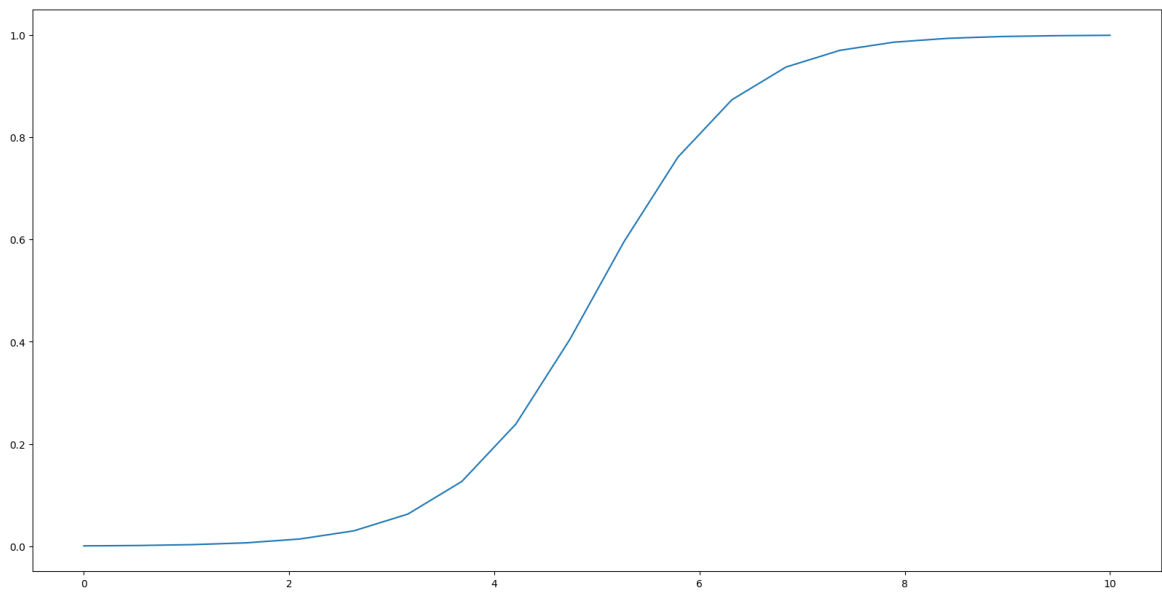


```
In [9]: b
```

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

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

```
Out[10]: [<matplotlib.lines.Line2D at 0x12a3aee50>]
```



```
In [11]: from mpl_toolkits.mplot3d import Axes3D # noqa: F401 unused import

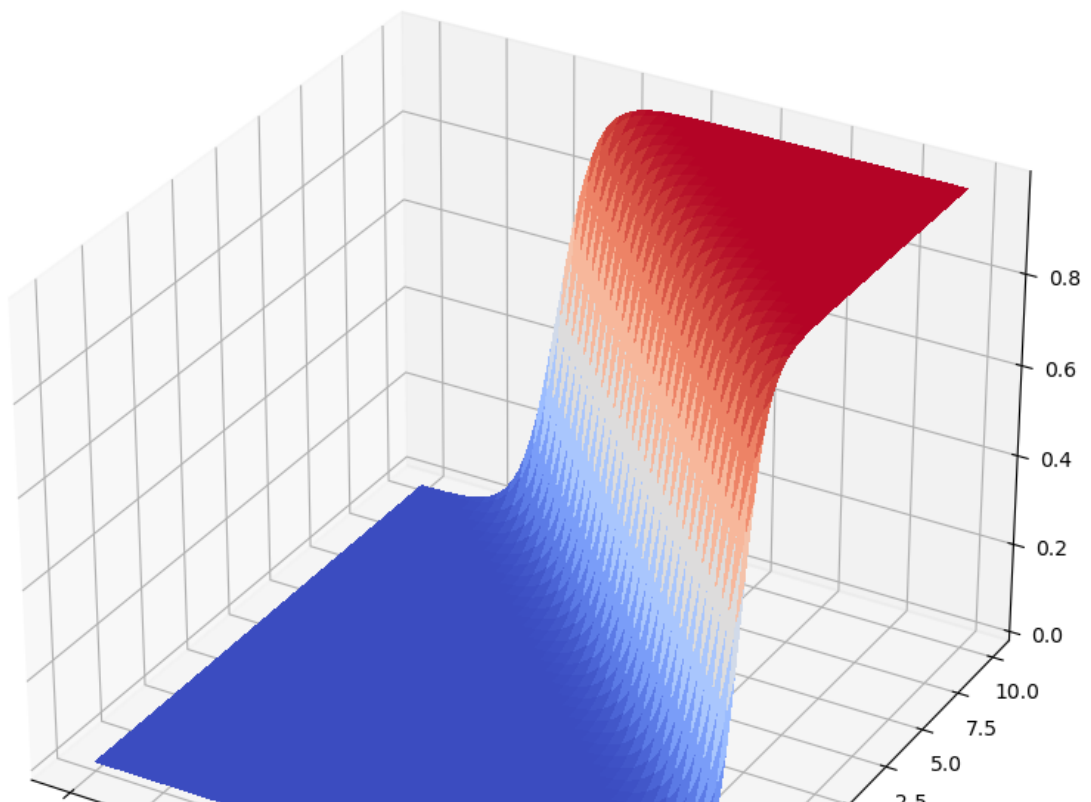
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np

fig = plt.figure()
ax = fig.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)
```

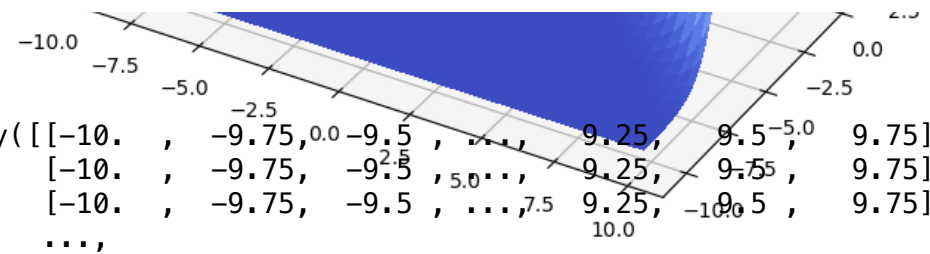
/var/folders/9c/3ylcvqn54zs1r5tlfz4_znf00000gn/T/ipykernel_86853/290434025.py:10: MatplotlibDeprecationWarning: Calling gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two minor releases later, gca() will take no keyword arguments. The gca() function should only be used to get the current axes, or if no axes exist, create new axes with default keyword arguments. To create a new axes with non-default arguments, use plt.axes() or plt.subplot().

```
ax = fig.gca(projection='3d')
```



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]])
```

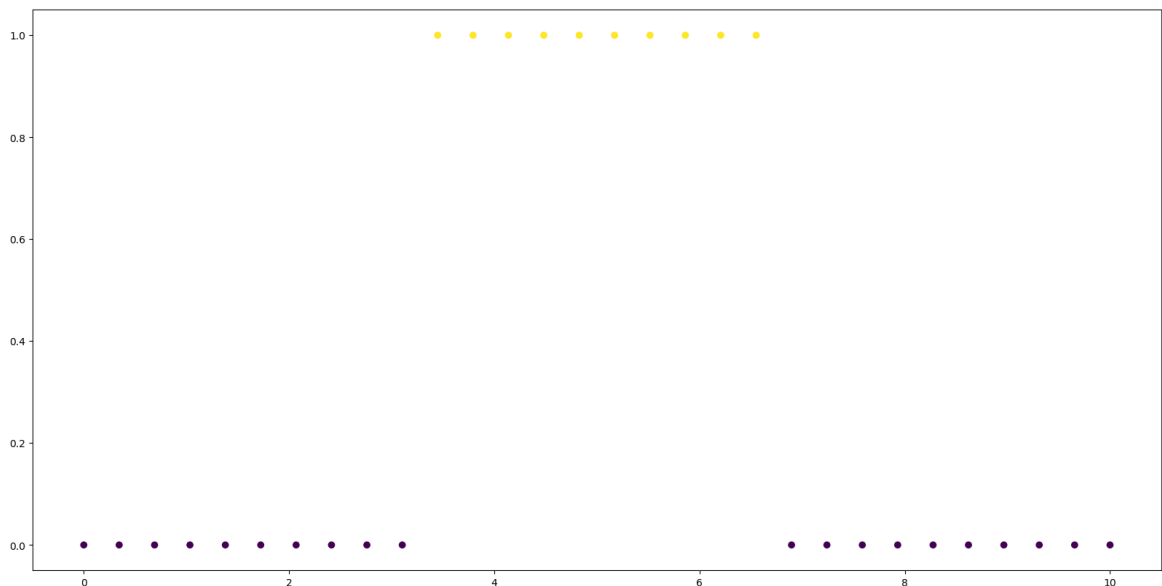
What if the data doesn't really fit this pattern?

```
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 0x12b0beac0>



In [16]:

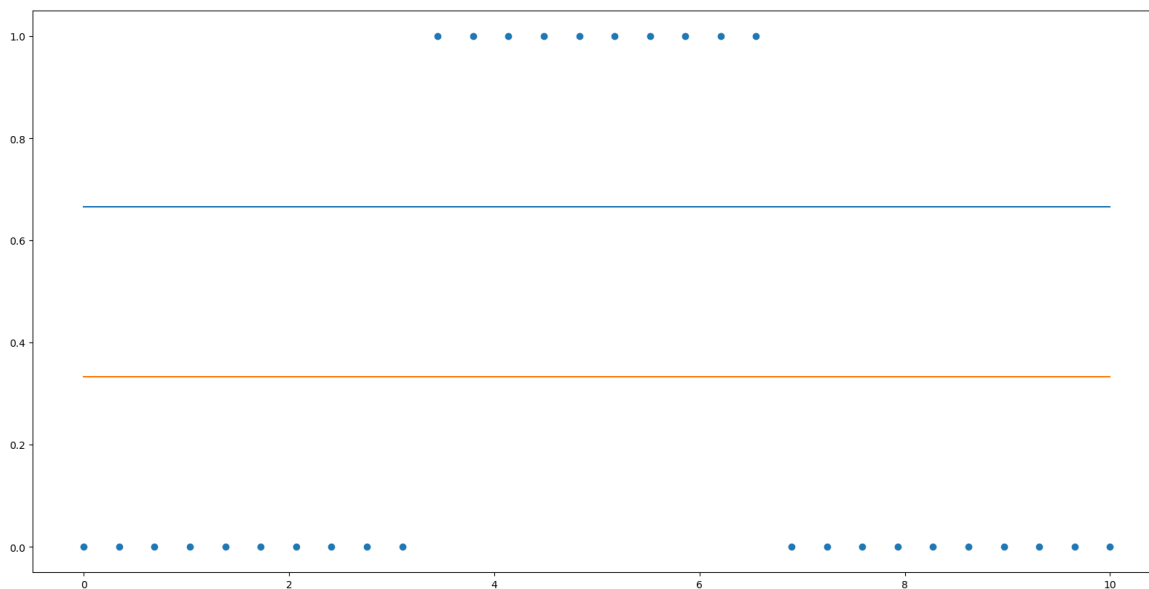
model.fit(x.reshape(-1, 1), y)

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 0x12b04b370>,
<matplotlib.lines.Line2D at 0x12b04b3d0>]
```



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

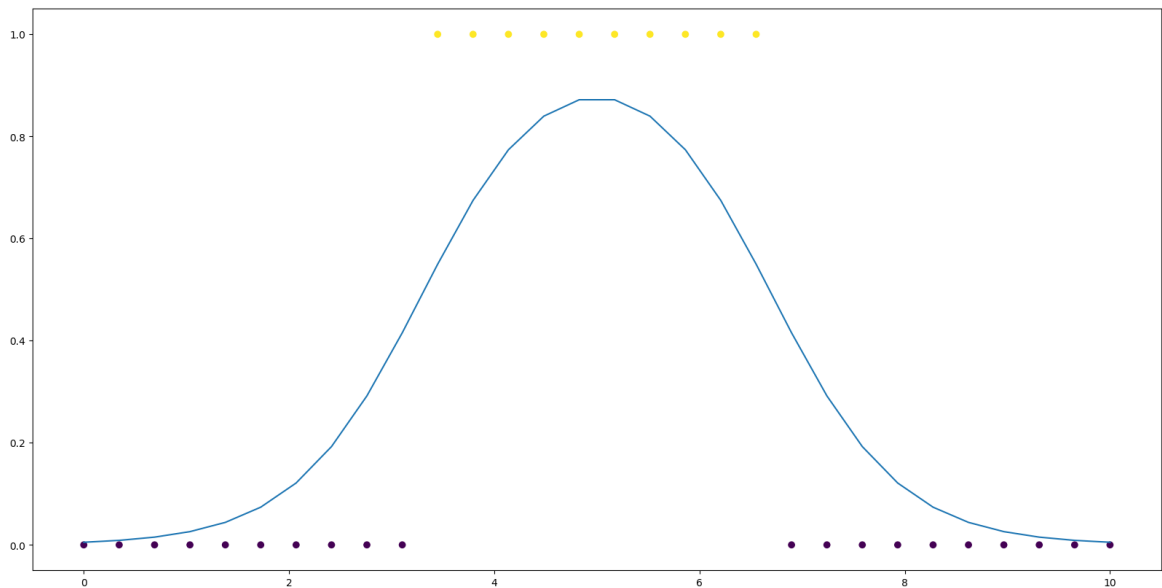
```
Out[18]: ▼ LogisticRegression
LogisticRegression()
```

```
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.predi
```

```
Out[20]: [<matplotlib.lines.Line2D at 0x12c0aad90>]
```



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

```
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
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 [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]])
```

```
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

```
In [35]: heart = pd.read_csv('Heart.csv')
heart
```

Out[35]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpe
0	1	63	1	typical	145	233	1	2	150	0	...
1	2	67	1	asymptomatic	160	286	0	2	108	1	...
2	3	67	1	asymptomatic	120	229	0	2	129	1	...
3	4	37	1	nonanginal	130	250	0	0	187	0	...
4	5	41	0	nontypical	130	204	0	2	172	0	...
...
298	299	45	1	typical	110	264	0	0	132	0	...
299	300	68	1	asymptomatic	144	193	1	0	141	0	...
300	301	57	1	asymptomatic	130	131	0	0	115	1	...
301	302	57	0	nontypical	130	236	0	2	174	0	...
302	303	38	1	nonanginal	138	175	0	0	173	0	...

303 rows × 15 columns

```
In [36]: heart = heart.dropna()
heart
```

Out[36]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpe
0	1	63	1	typical	145	233	1	2	150	0	...
1	2	67	1	asymptomatic	160	286	0	2	108	1	...
2	3	67	1	asymptomatic	120	229	0	2	129	1	...
3	4	37	1	nonanginal	130	250	0	0	187	0	...
4	5	41	0	nontypical	130	204	0	2	172	0	...
...
297	298	57	0	asymptomatic	140	241	0	0	123	1	...
298	299	45	1	typical	110	264	0	0	132	0	...
299	300	68	1	asymptomatic	144	193	1	0	141	0	...
300	301	57	1	asymptomatic	130	131	0	0	115	1	...
301	302	57	0	nontypical	130	236	0	2	174	0	...

297 rows × 15 columns

```
In [37]: from sklearn.model_selection import train_test_split
```

```
In [38]: train, test = train_test_split(heart, test_size=.20)
heart.shape, train.shape, test.shape
```

Out[38]: ((297, 15), (237, 15), (60, 15))

```
In [39]: from sklearn import preprocessing
```

```
In [40]: enc = preprocessing.OrdinalEncoder()
```

```
In [47]: enc.fit(train[["ChestPain"]])
enc.categories_
```

Out[47]: [array(['asymptomatic', 'nonanginal', 'nontypical', 'typical'],
dtype=object)]

```
In [48]: train2 = train.copy()

train2["ChestPain"] = enc.fit_transform(train[["ChestPain"]])
train2.head()
```

Out[48]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak
274	275	59	1	3.0	134	204	0	0	162	0	0.8
80	81	45	1	0.0	104	208	0	2	148	1	3.0
140	141	59	1	2.0	140	221	0	0	164	1	0.0
21	22	58	0	3.0	150	283	1	2	162	0	1.0
64	65	54	1	0.0	120	188	0	0	113	0	1.4

```
In [49]: enc.fit(train[["Thal"]])
enc.categories_
```

Out[49]: [array(['fixed', 'normal', 'reversible'], dtype=object)]

```
In [50]: train2["Thal"] = enc.fit_transform(train[["Thal"]])
train2.head()
```

Out[50]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak
274	275	59	1	3.0	134	204	0	0	162	0	0.8
80	81	45	1	0.0	104	208	0	2	148	1	3.0
140	141	59	1	2.0	140	221	0	0	164	1	0.0
21	22	58	0	3.0	150	283	1	2	162	0	1.0
64	65	54	1	0.0	120	188	0	0	113	0	1.4

```
In [51]: enc.fit(train[["AHD"]])
enc.categories_
```

Out[51]: [array(['No', 'Yes'], dtype=object)]

```
In [52]: train2["AHD"] = enc.fit_transform(train[["AHD"]])
train2.head()
```

Out[52]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak
274	275	59	1	3.0	134	204	0	0	162	0	0.8
80	81	45	1	0.0	104	208	0	2	148	1	3.0
140	141	59	1	2.0	140	221	0	0	164	1	0.0
21	22	58	0	3.0	150	283	1	2	162	0	1.0
64	65	54	1	0.0	120	188	0	0	113	0	1.4

```
In [53]: test2 = test.copy()
```

```
In [54]: enc.fit(test[["ChestPain"]])
test2["ChestPain"] = enc.fit_transform(test[["ChestPain"]])
enc.fit(test[["Thal"]])
test2["Thal"] = enc.fit_transform(test[["Thal"]])
enc.fit(test[["AHD"]])
test2["AHD"] = enc.fit_transform(test[["AHD"]])
test2.head()
```

Out[54]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak
218	219	64	0	0.0	130	303	0	0	122	0	2.0
48	49	65	0	1.0	140	417	1	2	157	0	0.8
110	111	61	0	0.0	145	307	0	2	146	1	1.0
138	139	35	1	0.0	120	198	0	0	130	1	1.6
251	252	58	1	0.0	146	218	0	0	105	0	2.0

```
In [56]: test2 = test2.dropna()
test2
```

Out[56]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak
218	219	64	0	0.0	130	303	0	0	122	0	2.0
48	49	65	0	1.0	140	417	1	2	157	0	0.8
110	111	61	0	0.0	145	307	0	2	146	1	1.0
138	139	35	1	0.0	120	198	0	0	130	1	1.6
251	252	58	1	0.0	146	218	0	0	105	0	2.0
246	247	58	1	0.0	100	234	0	0	156	0	0.1
79	80	58	1	0.0	150	270	0	2	111	1	0.8
23	24	58	1	1.0	132	224	0	2	173	0	3.2
172	173	59	0	0.0	174	249	0	0	143	1	0.0
102	103	57	0	0.0	128	303	0	2	159	0	0.0
97	98	60	0	0.0	150	258	0	2	157	0	2.6
59	60	51	1	3.0	125	213	0	2	125	1	1.4
282	283	55	0	0.0	128	205	0	1	130	1	2.0
296	297	59	1	0.0	164	176	1	2	90	0	1.0
238	239	49	0	2.0	134	271	0	0	162	0	0.0
241	242	41	0	2.0	126	306	0	0	163	0	0.0
286	287	58	0	0.0	170	225	1	2	146	1	2.8
78	79	48	1	2.0	130	245	0	2	180	0	0.2
28	29	43	1	0.0	150	247	0	0	171	0	1.5
175	176	57	1	0.0	152	274	0	0	88	1	1.2
207	208	50	1	0.0	144	200	0	2	126	1	0.9
50	51	41	0	2.0	105	198	0	0	168	0	0.0
91	92	62	0	0.0	160	164	0	2	145	0	6.2
108	109	61	1	0.0	120	260	0	0	140	1	3.6
42	43	71	0	2.0	160	302	0	0	162	0	0.4
1	2	67	1	0.0	160	286	0	2	108	1	1.5
41	42	40	1	3.0	140	199	0	0	178	1	1.4
174	175	64	1	0.0	145	212	0	2	132	0	2.0
129	130	62	0	0.0	124	209	0	0	163	0	0.0
96	97	59	1	0.0	110	239	0	2	142	1	1.2
94	95	63	0	1.0	135	252	0	2	172	0	0.0

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak
285	286	58	1	0.0	114	318	0	1	140	0	4.4
22	23	58	1	2.0	120	284	0	2	160	0	1.8
103	104	71	0	1.0	110	265	1	2	130	0	0.0
120	121	48	1	0.0	130	256	1	2	150	1	0.0
107	108	57	1	1.0	128	229	0	2	150	0	0.4
0	1	63	1	3.0	145	233	1	2	150	0	2.3
147	148	41	1	1.0	112	250	0	0	179	0	0.0
163	164	58	0	0.0	100	248	0	2	122	0	1.0
26	27	58	0	1.0	120	340	0	0	172	0	0.0
62	63	58	1	0.0	128	216	0	2	131	1	2.2
226	227	47	1	0.0	112	204	0	0	143	0	0.1
119	120	65	1	0.0	135	254	0	2	127	0	2.8
86	87	47	1	1.0	138	257	0	2	156	0	0.0
118	119	63	1	0.0	130	330	1	2	132	1	1.8
117	118	35	0	0.0	138	183	0	0	182	0	1.4
292	293	44	1	0.0	120	169	0	0	144	1	2.8
281	282	47	1	1.0	130	253	0	0	179	0	0.0
247	248	47	1	0.0	110	275	0	2	118	1	1.0
44	45	61	0	0.0	130	330	0	2	169	0	0.0
55	56	54	1	0.0	124	266	0	2	109	1	2.2
88	89	53	0	0.0	138	234	0	2	160	0	0.0
106	107	59	1	0.0	140	177	0	0	162	1	0.0
243	244	61	1	3.0	134	234	0	0	145	0	2.6
300	301	57	1	0.0	130	131	0	0	115	1	1.2
264	265	61	1	0.0	138	166	0	2	125	1	3.6
98	99	52	1	2.0	134	201	0	0	158	0	0.8
177	178	56	1	0.0	132	184	0	2	105	1	2.1

In [57]: `model = LogisticRegression()`

In [60]: `model.fit(preprocessing.scale(train2.drop(['AHD'], axis=1)), train2.AH`

Out[60]:

▼ LogisticRegression

LogisticRegression()

```
In [61]: pred_train = model.predict(preprocessing.scale(train2.drop(['AHD'], axis=1), axis=0)
pred_test = model.predict(preprocessing.scale(test2.drop(['AHD'], axis=1), axis=0))
```

```
In [64]: train2.AHD
```

```
Out[64]: 274      1.0
          80      0.0
          140     0.0
          21      0.0
          64      1.0
          ...
          260     0.0
          29      1.0
          101     0.0
          132     0.0
          178     0.0
          Name: AHD, Length: 237, dtype: float64
```

```
In [65]: accuracy_score(train2.AHD, pred_train)
```

```
Out[65]: 0.8396624472573839
```

```
In [66]: confusion_matrix(train2.AHD, pred_train)
```

```
Out[66]: array([[122, 13],
                [ 25, 77]])
```

```
In [67]: print(classification_report(train2.AHD, pred_train))
```

	precision	recall	f1-score	support
0.0	0.83	0.90	0.87	135
1.0	0.86	0.75	0.80	102
accuracy			0.84	237
macro avg	0.84	0.83	0.83	237
weighted avg	0.84	0.84	0.84	237

```
In [68]: print(classification_report(test2.AHD, pred_test))
```

	precision	recall	f1-score	support
0.0	0.80	0.96	0.87	25
1.0	0.97	0.83	0.89	35
accuracy			0.88	60
macro avg	0.88	0.89	0.88	60
weighted avg	0.90	0.88	0.88	60

```
In [84]: from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion='entropy', max_depth=2)
```

```
In [85]: model.fit(train2.drop(['AHD'], axis=1), train2.AHD)
```

```
Out[85]: DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=2)
```

```
In [86]: pred_train = model.predict(train2.drop(['AHD'], axis=1))
pred_test = model.predict(test2.drop(['AHD'], axis=1))
```

```
In [87]: # tree depth 2
accuracy_score(train2.AHD, pred_train)
```

```
Out[87]: 0.7848101265822784
```

```
In [88]: # tree depth 2
confusion_matrix(train2.AHD, pred_train)
```

```
Out[88]: array([[112, 23],
               [ 28, 74]])
```

```
In [89]: # tree depth 2
print(classification_report(train2.AHD, pred_train))
```

	precision	recall	f1-score	support
0.0	0.80	0.83	0.81	135
1.0	0.76	0.73	0.74	102
accuracy			0.78	237
macro avg	0.78	0.78	0.78	237
weighted avg	0.78	0.78	0.78	237

```
In [90]: # tree depth 2
print(classification_report(test2.AHD, pred_test))
```

	precision	recall	f1-score	support
0.0	0.76	0.88	0.81	25
1.0	0.90	0.80	0.85	35
accuracy			0.83	60
macro avg	0.83	0.84	0.83	60
weighted avg	0.84	0.83	0.83	60

In this case, the algorithm that appears to be optimal is the logistical regression. I also ran the tree classificaiton at a depth of 4, which made the accuracy almost comparable to logistical but still slightly lower. When applied to the test data, the accuracy was higher than the train data, but the logistical regression had an accuracy of 0.88 while tree depth of 2 had an accuracy of 0.83.

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 [77]: from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion='entropy', max_depth=8)
```

```
In [78]: model.fit(train2.drop(['AHD'], axis=1), train2.AHD)
```

```
Out[78]: ▼ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=8)
```

```
In [79]: pred_train = model.predict(train2.drop(['AHD'], axis=1))
pred_test = model.predict(test2.drop(['AHD'], axis=1))
```

```
In [80]: # tree depth 8
accuracy_score(train2.AHD, pred_train)
```

```
Out[80]: 0.9789029535864979
```

```
In [81]: # tree depth 8
confusion_matrix(train2.AHD, pred_train)
```

```
Out[81]: array([[133,  2],
               [ 3,  99]])
```

```
In [82]: # tree depth 8
print(classification_report(train2.AHD, pred_train))
```

	precision	recall	f1-score	support
0.0	0.98	0.99	0.98	135
1.0	0.98	0.97	0.98	102
accuracy			0.98	237
macro avg	0.98	0.98	0.98	237
weighted avg	0.98	0.98	0.98	237

```
In [83]: # tree depth 8
print(classification_report(test2.AHD, pred_test))
```

	precision	recall	f1-score	support
0.0	0.79	0.92	0.85	25
1.0	0.94	0.83	0.88	35
accuracy			0.87	60
macro avg	0.86	0.87	0.87	60
weighted avg	0.88	0.87	0.87	60

Compared to the previous runs for decision trees, with a depth of 8, the train data has the best accuracy scores over the other algorithms. However, when run against the test data, still only reaches an accuracy of 0.87 which is comparable to the logistic regression case but did not get above 0.88. For these scenarios, the logistic regression may still be optimal.

In []: