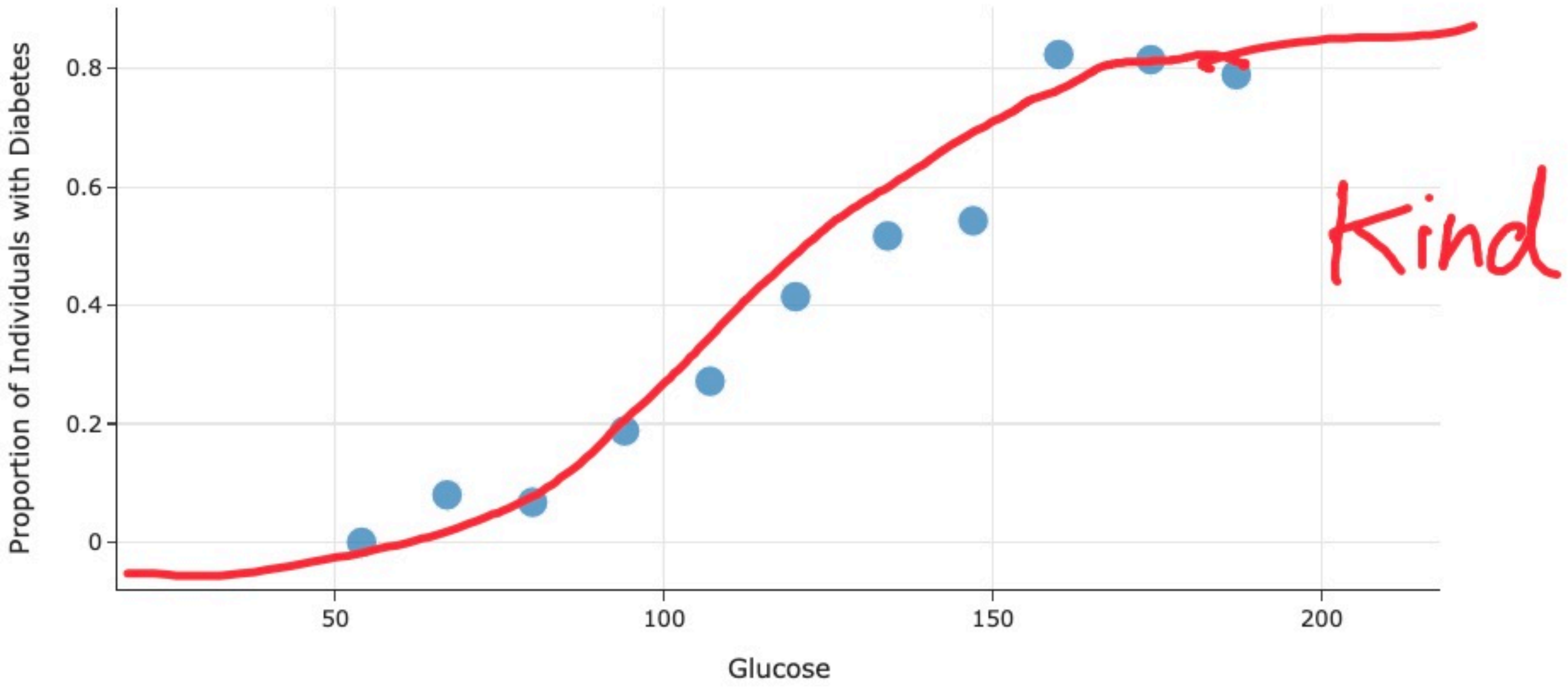
- Another approach we could try is to:
  - Place `'Glucose'` values into **bins**, e.g. 50 to 55, 55 to 60, 60 to 65, etc.
  - Within each bin, compute the proportion of patients in the training set who had diabetes.

In [6]:

```
# Take a look at the source code in lec22_util.py to see how we did this!
# We've hidden a lot of the plotting code in the notebook to make it cleaner.
util.make_prop_plot(X_train, y_train)
```



Kind of see an S

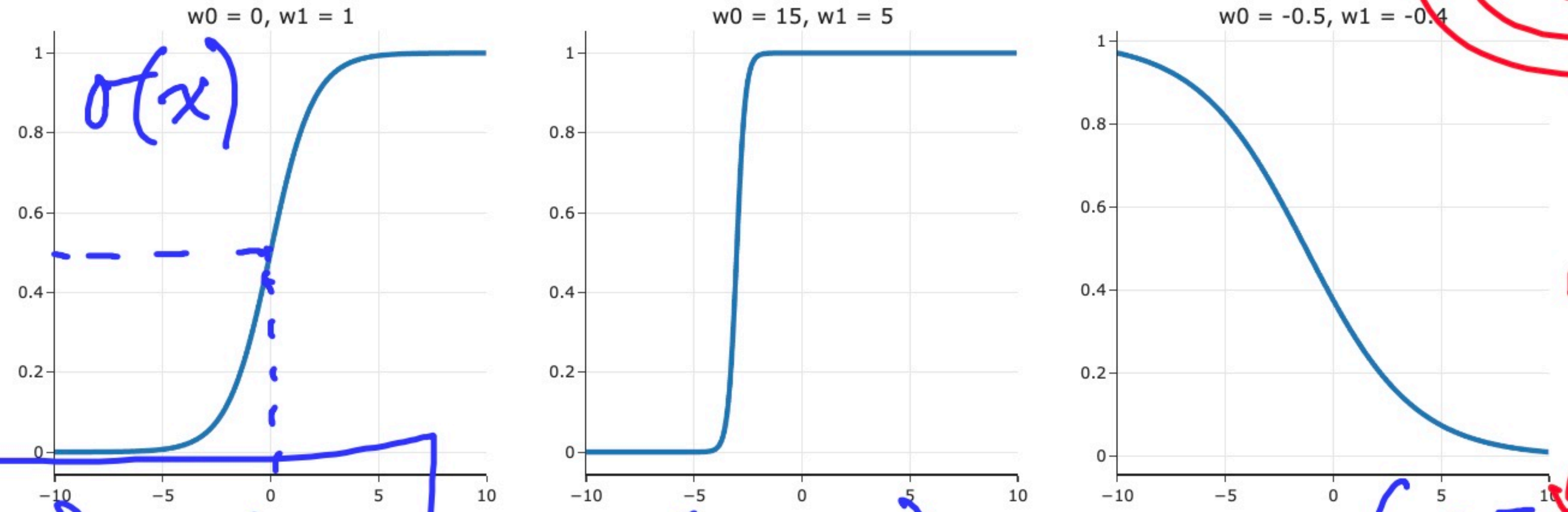
- Below, we'll look at the shape of $y = \sigma(\underset{\sim}{\overset{\uparrow}{w_0}} + \underset{\sim}{\overset{\uparrow}{w_1}} x)$ for different values of w_0 and w_1.
  - w_0 controls the position of the curve on the x-axis.
  - w_1 controls the "steepness" of the curve.

σ: sigmoid curve

sigmoid's logistic.

In [7]:

```
util.show_three_sigmoids()
```



$$\boxed{\sigma(0) = \frac{1}{2}}$$

$\sigma(15 + \underset{\uparrow}{5}x)$ steeper

$\sigma(\underset{\sim}{-0.5}\underset{\sim}{-0.4}x)$

Logistic regression

- Logistic **regression** is a linear **classification** technique that builds upon linear regression.

  It is **not** called logistic**al** regression!

- It models **the probability of belonging to class 1, given a feature vector**:

$$P(y_i = 1|\vec{x}_i) = \sigma\underbrace{\left(w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} + \ldots + w_d x_i^{(d)}\right)}_{\text{linear regression model}} = \sigma\left(\vec{w} \cdot \text{Aug}(\vec{x}_i)\right)$$

guaranteed that

$$0 < P(y_i = 1 | \vec{x}_i) < 1$$

Logistic regression

- Logistic **regression** is a linear **classification** technique that builds upon linear regression.

  It is **not** called logistic**al** regression!

- It models **the probability of belonging to class 1, given a feature vector**:

$$P(y_i = 1|\vec{x}_i) = \sigma\underbrace{\left(w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} + \ldots + w_d x_i^{(d)}\right)}_{\text{linear regression model}} = \sigma\left(\vec{w} \cdot \text{Aug}(\vec{x}_i)\right)$$

parameters

guaranteed that

$$0 < P(y_i = 1 | \vec{x}_i) < 1$$

```
In [9]: from sklearn.linear_model import LogisticRegression
```

- Let's fit a `LogisticRegression` classifier. Specifically, this means we're asking `sklearn` to learn the optimal parameters w_0^* and w_1^* in:

$$P(y_i = 1 | \text{Glucose}_i) = \sigma\left(w_0 + w_1 \cdot \text{Glucose}_i\right)$$

```
In [11]: model_logistic = LogisticRegression()
         model_logistic.fit(X_train[['Glucose']], y_train)
```

```
Out[11]: ▾ LogisticRegression
         LogisticRegression()
```

training data

- We get a test accuracy that's roughly in line with the test accuracies of the two models we saw last class.

75% test accuracy

```
In [12]: model_logistic.score(X_test[['Glucose']], y_test)

Out[12]: 0.75
```

Attempting to use squared loss

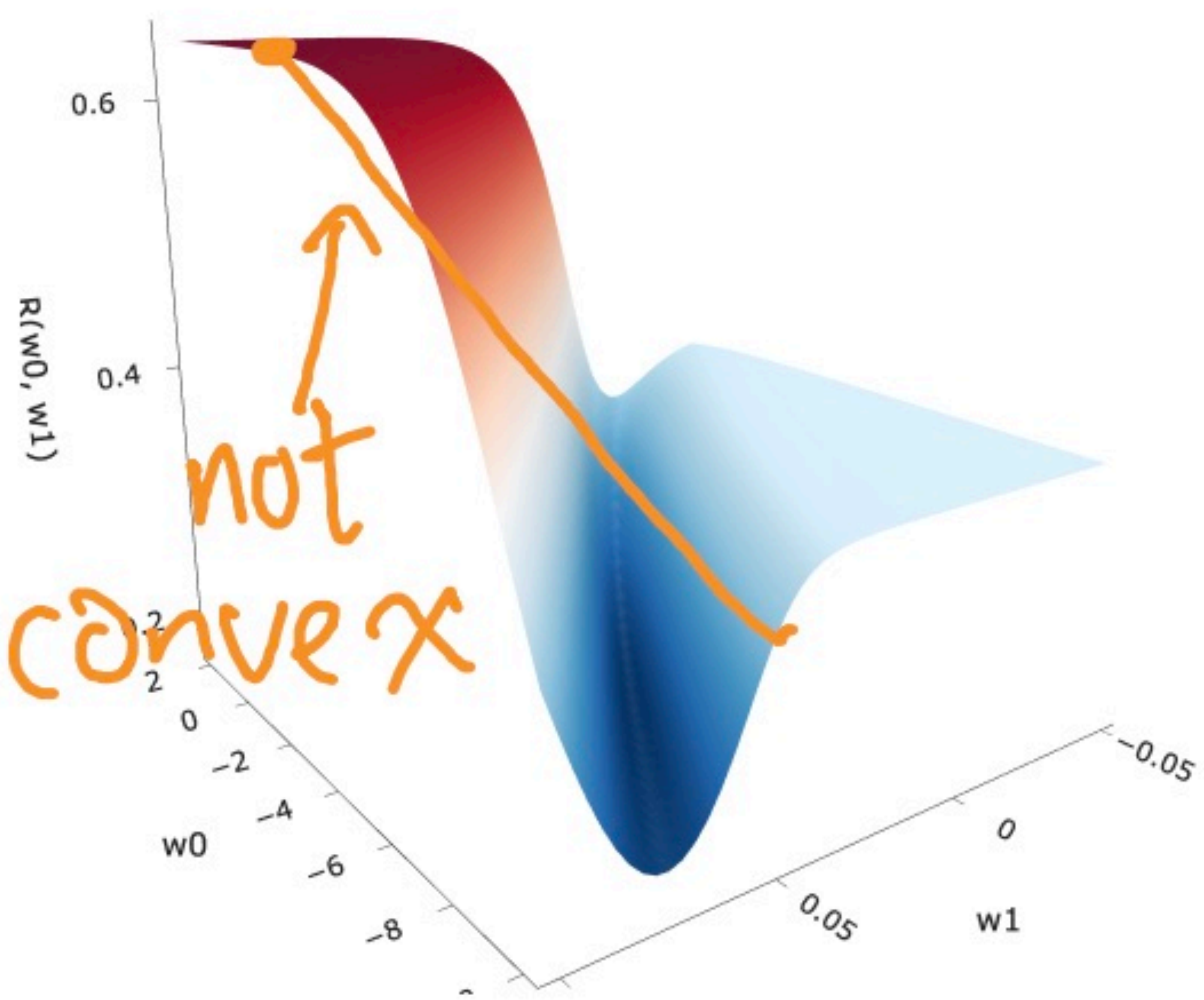
- Our default loss function has always been squared loss, so we could try and use it here.

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \underbrace{\sigma\left(\vec{w}\cdot \text{Aug}(\vec{x}_i)\right)}_{\text{logistic regression model's predictions}}\right)^2$$

$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \sigma(w_0 + w_1 \underbrace{x_i}_{\text{Glucose}_i})\right)$$

In [18]:
```
util.show_logistic_mse_surface(X_train, y_train)
```

Mean Squared Error Loss Surface
for Logistic Regression



localhost
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
$(1-p_i)^2$
p_i

- Suppose $y_i = 1$. Then, the graph of the squared loss of the prediction p_i is below.

In [19]:
```
util.show_squared_loss_individual()
```

Squared loss is bounded to (0, 1) when predicting probabilities!

p_i

$(1-p_i)^2$

pretend

$p_i = 0.02$

squared loss:

$(1-0.02)^2.$

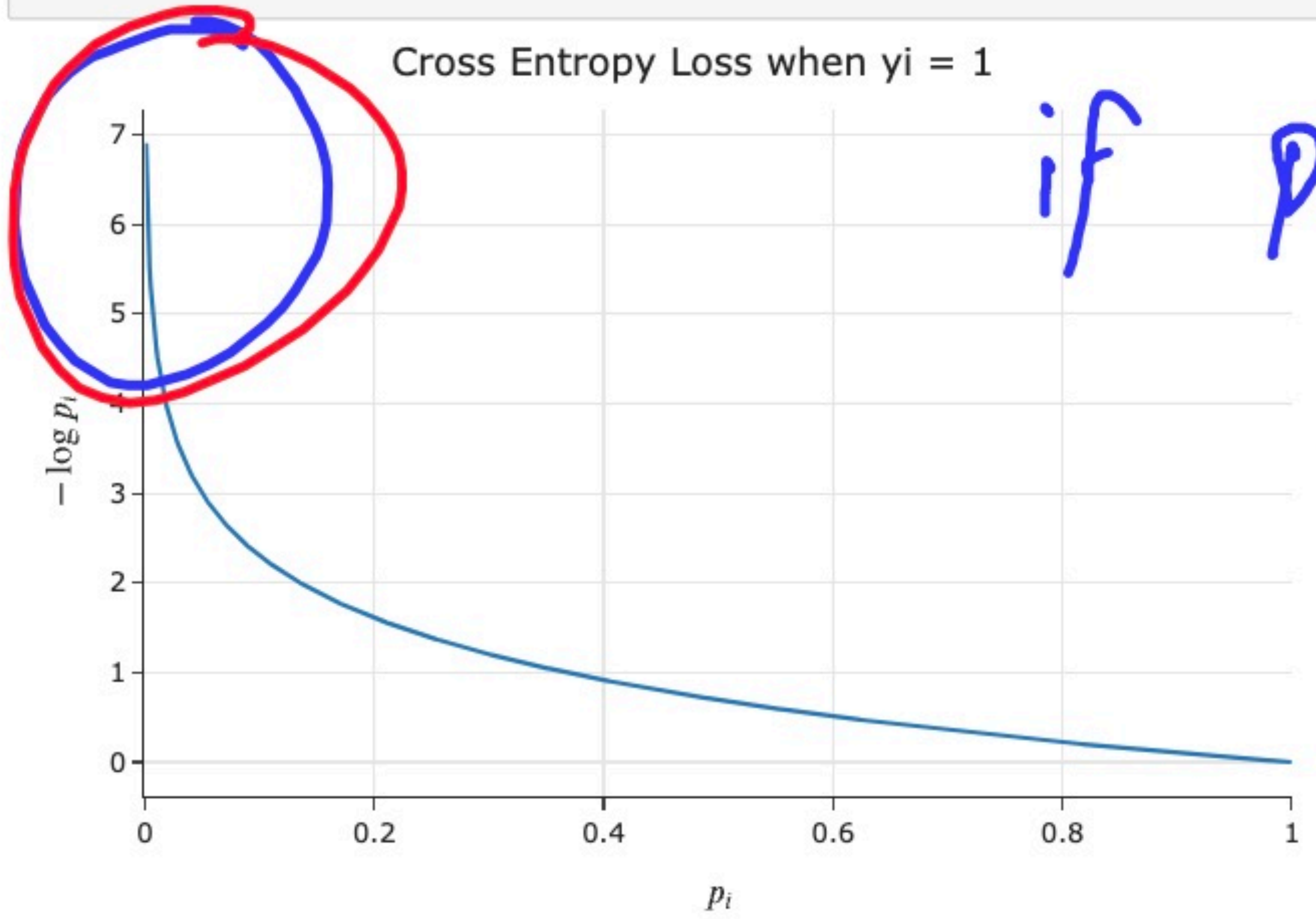
predicted **probability**, then:

$$L_{\text{ce}}(y_i, p_i) = \begin{cases} -\log(p_i) & \text{if } y_i = 1 \\ -\log(1-p_i) & \text{if } y_i = 0 \end{cases}$$

- Note that in the two cases – $y_i = 1$ and $y_i = 0$ – the cross-entropy loss function resembles square loss, but is unbounded when the predicted probabilities p_i are far from y_i.

In [20]:

```
util.show_ce_loss_individual_1()
```



if patient i has diabetes

but we predict

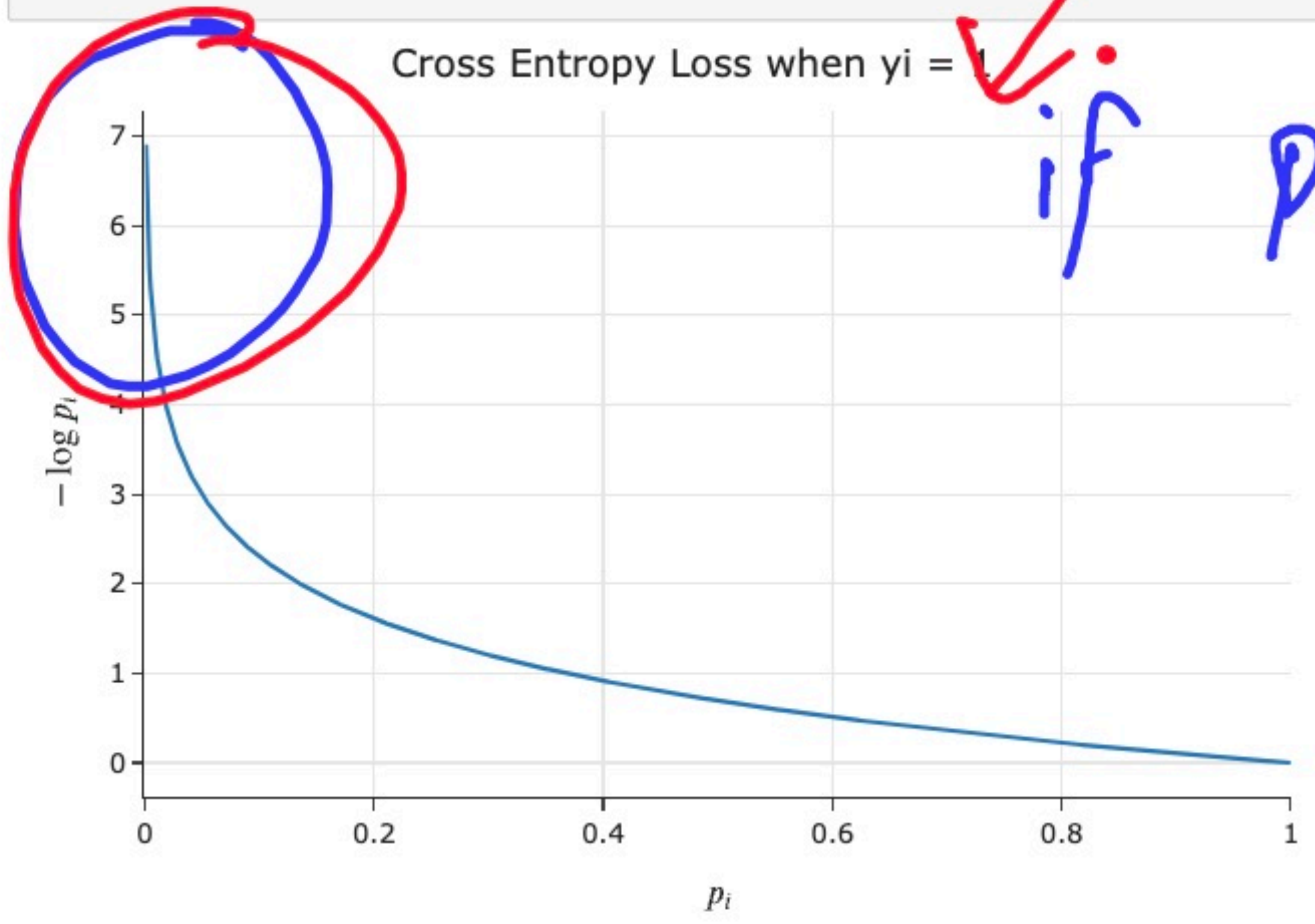
p_i = Probability of diabetes to be small loss is very high!

predicted **probability**, then:

$$L_{\text{ce}}(y_i, p_i) = \begin{cases} -\log(p_i) & \text{if } y_i = 1 \\ -\log(1 - p_i) & \text{if } y_i = 0 \end{cases}$$

- Note that in the two cases – $y_i = 1$ and $y_i = 0$ – the cross-entropy loss function resembles square loss, but is unbounded when the predicted probabilities p_i are far from y_i.

```
In [20]: util.show_ce_loss_individual_1()
```



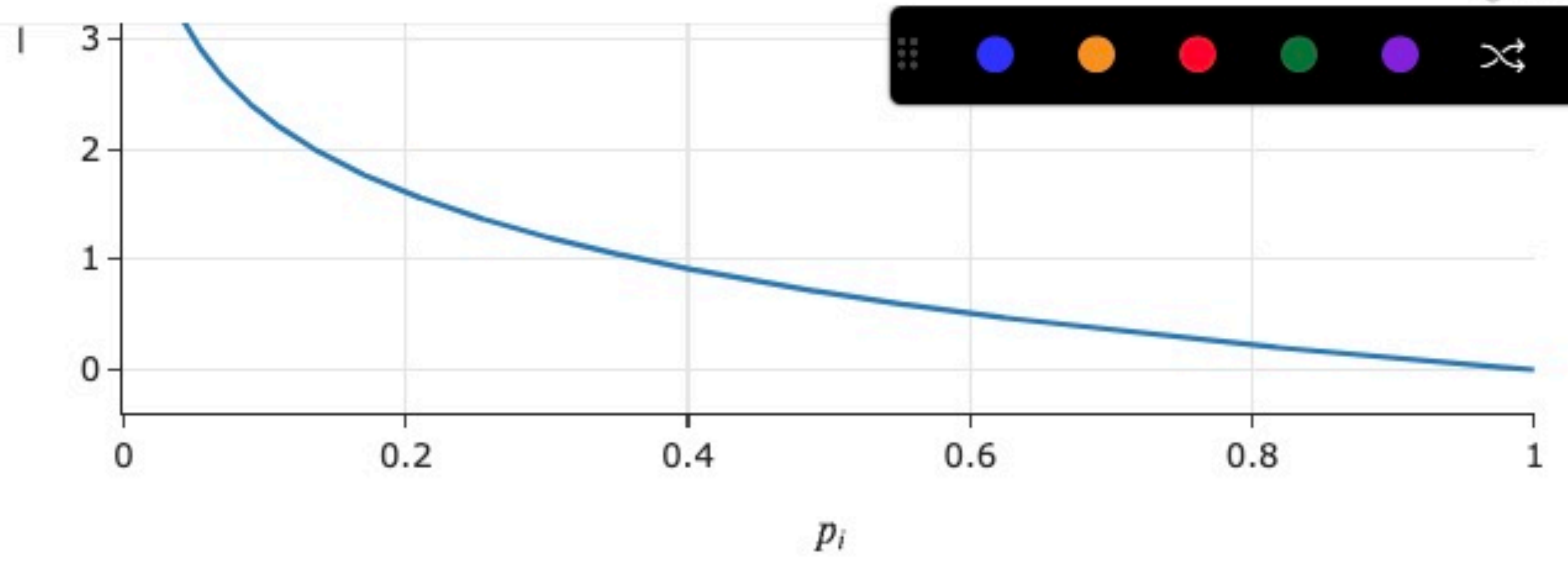
if patient i has diabetes

but we predict

p_i = Probability of diabetes

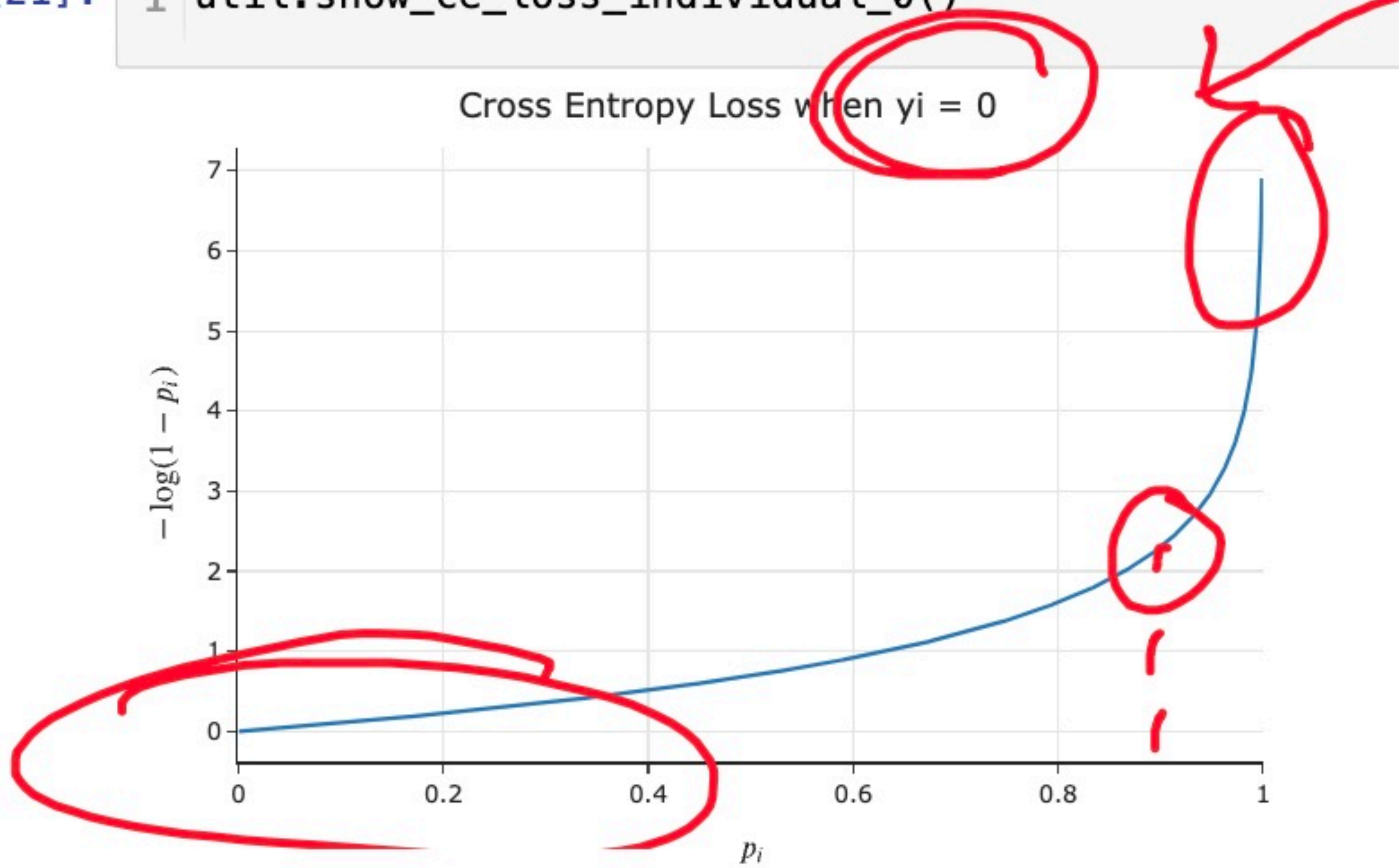
to be small

loss is very high!



$$L_{\text{ce}}(y_i, p_i) = \begin{cases} -\log(p_i) & \text{if } y_i = 1 \\ -\log(1 - p_i) & \text{if } y_i = 0 \end{cases}$$

```
In [21]: util.show_ce_loss_individual_0()
```



mirror image of each other .

A non-piecewise definition of cross-entropy loss

- We can define the cross-entropy loss function piecewise. If y_i is an observed value and p_i is a predicted **probability**, then:

$$L_{\text{ce}}(y_i, p_i) = \begin{cases} -\log(p_i) & \text{if } y_i = 1 \\ -\log(1 - p_i) & \text{if } y_i = 0 \end{cases}$$

- An equivalent formulation of L_{ce} that isn't piecewise is:

$$L_{\text{ce}}(y_i, p_i) = -(y_i \log p_i + \cancel{(\overset{1-1}{1 - y_i}) \log(1 - p_i)})$$

if $y_i = 1$:

localhost
T
22 . 1
?

A non-piecewise definition of cross-entropy loss

- We can define the cross-entropy loss function piecewise. If y_i is an observed value and p_i is a predicted **probability**, then:

$$L_{\text{ce}}(y_i, p_i) = \begin{cases} -\log(p_i) & \text{if } y_i = 1 \\ -\log(1 - p_i) & \text{if } y_i = 0 \end{cases}$$

- An equivalent formulation of L_{ce} that isn't piecewise is:

$$L_{\text{ce}}(y_i, p_i) = -\big(\cancel{\overset{0}{y_i}\log p_i} + (1 - y_i)\log(1 - p_i)\big)$$

if $y_i = 0$

Average cross-entropy loss

- **Cross-entropy loss** for an observed value y_i and predicted **probability** $p_i = P(y=1|\vec{x}_i) = \sigma\left(\vec{w} \cdot \text{Aug}(\vec{x}_i)\right)$ is:

$$L_{\text{ce}}(y_i, p_i) = -\left(y_i \log p_i + (1-y_i)\log(1-p_i)\right)$$

$$-\Big(y_i \log \sigma(\vec{w} \cdot \text{Aug}(\vec{x}_i))\Big) + (1-y_i) - - ..$$

$$P(y_i = 1|\vec{x}_i) = \sigma(w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} + \ldots + w_d x_i^{(d)}) = \sigma\left(\vec{w} \cdot \text{Aug}(\vec{x}_i)\right)$$

2. Choose a loss function.

cross-entropy loss

$$L_{\text{ce}}(y_i, p_i) = -\left(y_i \log p_i + (1 - y_i)\log(1 - p_i)\right)$$

$$\text{where } p_i = P(y = 1|\vec{x}_i) = \sigma\left(\vec{w} \cdot \text{Aug}(\vec{x}_i)\right)$$

3. Minimize average loss to find optimal model parameters.

As we've now seen, average loss could also be regularized!

$$\begin{aligned} R_{\text{ce}}(\vec{w}) &= -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \log p_i + (1 - y_i)\log(1 - p_i)\right) \\ &= -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log\left(\sigma\left(\vec{w} \cdot \text{Aug}(\vec{x}_i)\right)\right) + (1 - y_i)\log\left(1 - \sigma\left(\vec{w} \cdot \text{Aug}(\vec{x}_i)\right)\right)\right] \end{aligned}$$

The actual minimization here is done using numerical methods, through `sklearn`.

localhost
26 . 1

`LogisticRegression` in `sklearn`, revisited

- The `LogisticRegression` class in `sklearn` has a lot of hidden, default hyperparameters.

```
In [24]: LogisticRegression?
```

- It performs L_2 regularization ("ridge logistic regression") **by default**. The hyperparameter for regularization strength, C, is the **inverse** of λ; by default, it sets $C = 1$.

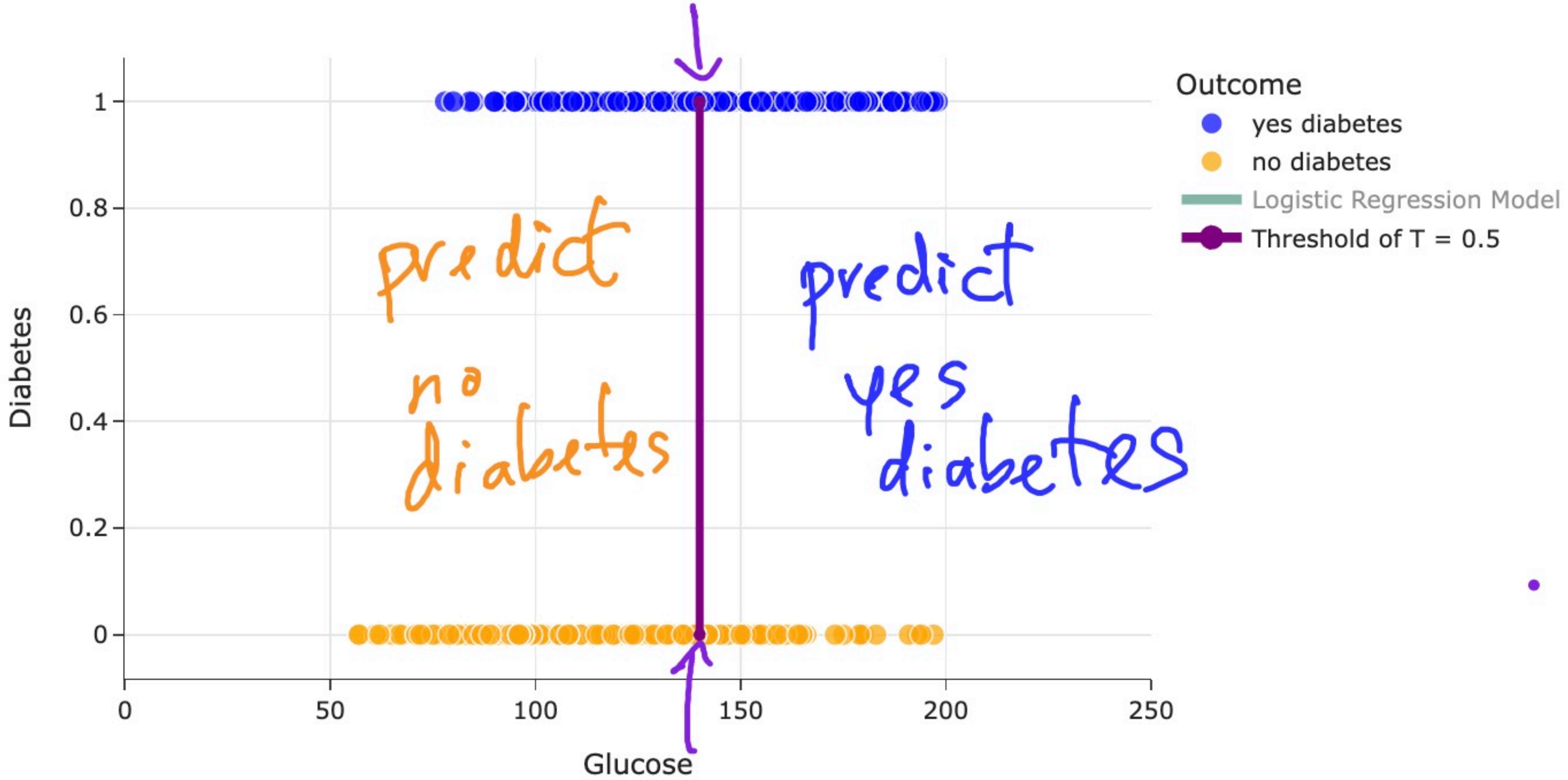
$$C = \frac{1}{\lambda}$$

- So, for a given value of C, it minimizes:

$$R_{\text{ce-reg}}(\vec{w}) \underset{=}{-} \underbrace{\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log\left(\sigma\left(\vec{w}\cdot \text{Aug}(\vec{x}_i)\right)\right) + (1-y_i)\log\left(1-\sigma\left(\vec{w}\cdot\text{Aug}(\vec{x}_i)\right)\right)\right]}_{\text{average CE loss}} + \underbrace{\frac{1}{C}\sum_{j=1}^{d} w_j^2}_{\text{by default, uses } L_2 \text{ reg.}}$$

In [29]:

```
util.show_one_feature_plot_with_logistic_and_x_threshold(X_train, y_train, 0.5
```



- **How do we find the exact x-axis position of the decision bound**

If we can, then we'd be able to predict whether someone has diabetes just by looking at their `'Glucose'` valu

- In our single feature model that predicts `Outcome` given just `Glucose`, our predicted probabilities are of the form:

$$P(y_i = 1|\text{Glucose}_i) = \sigma\left(w_0^* + w_1^* \cdot \text{Glucose}_i\right)$$

$$f^{-1}(f(x)) = x$$

- Suppose we fix a threshold, T. Then, our **decision boundary** is of the form:

$$\sigma^{-1}\Big(\sigma\left(w_0^* + w_1^* \cdot \text{Glucose}_T\right)\Big) = T \quad \sigma^{-1}(T)$$

- If we can invert $\sigma(t)$, then we can re-arrange the above to solve for the `'Glucose'` value at the threshold:

$$\text{Glucose}_\text{T} = \frac{\sigma^{-1}(T) - w_0^*}{w_1^*}$$

$$w_0^* + w_1^* \cdot \boxed{\text{Glucose}_T} = \sigma^{-1}(T)$$

- In our single feature model that predicts 'Outcome' given just 'Glucose', our predicted probabilities are of the form:

$$P(y_i = 1|\text{Glucose}_i) = \sigma\left(w_0^* + w_1^* \cdot \text{Glucose}_i\right)$$

- Suppose we fix a threshold, T. Then, our **decision boundary** is of the form:

$$\sigma\left(w_0^* + w_1^* \cdot \text{Glucose}_T\right) = T$$

- If we can invert $\sigma(t)$, then we can re-arrange the above to solve for the `'Glucose'` value at the threshold:

$$\log\left(\frac{T}{1-T}\right).$$

$$\text{Glucose}_\text{T} = \frac{\boxed{\sigma^{-1}(T)} - w_0^*}{w_1^*}$$

- **Important**: If $p = \sigma(t)$, then $\sigma^{-1}(p) = \log\left(\frac{p}{1-p}\right)$ is the inverse of $\sigma(t)$.

$\sigma^{-1}(p)$ is called the **logit** function.

- Suppose an event occurs with probability p.

- The **odds** of that event are:

$$\text{odds}(p) = \frac{p}{1-p}$$

- For instance, if there's a $p = \frac{3}{4}$ chance that Michigan wins this week, then the **odds** that Michigan wins this week are:

$$\text{odds}\left(\frac{3}{4}\right) = \frac{\frac{3}{4}}{\frac{1}{4}} = 3$$

- Interpretation: it's 3 times more likely that Michigan wins than loses.

- **We can interpret $\sigma^{-1}(p) = \overset{\text{log}}{\log}\left(\overset{\text{odds.}}{\frac{p}{1-p}}\right)$ as the "log odds" of p!**

See the reference slides for more details.

localhost

```
T = 0.5
glucose_threshold = (np.log(T / (1 - T)) - w0_star) / w1_star
glucose_threshold
```

Out[30]:

```
140.0083983057046
```

In [31]:

```
util.show_one_feature_plot_with_logistic_and_x_threshold(X_train, y_train, 0.5)
```

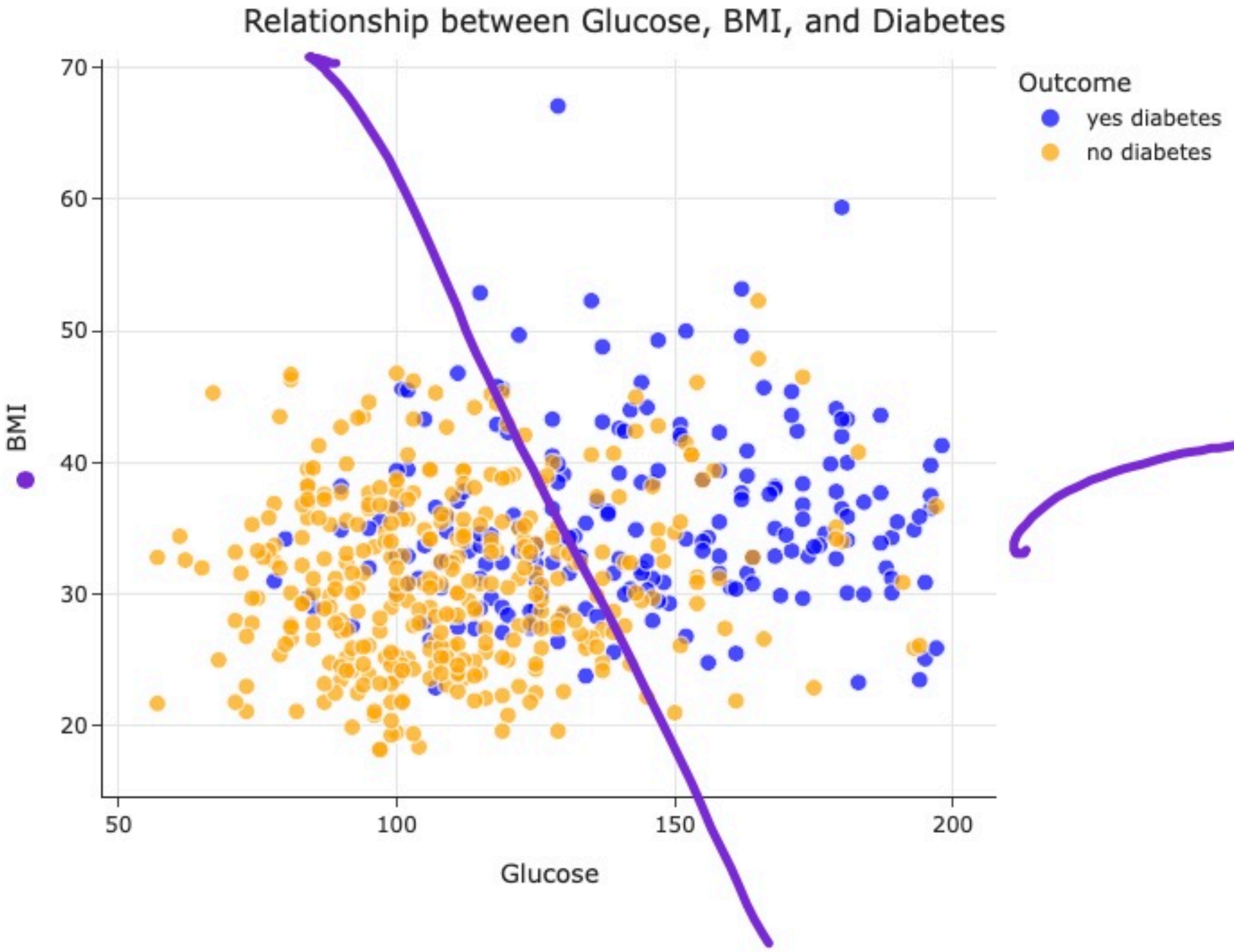
Diabetes
1
0.8
0.6
0.4
0.2
0
0
50
100
150
200
250
Glucose
Outcome
● yes diabetes
● no diabetes
Logistic Regression Model
Threshold of T = 0.5
← - - 50% probability
x = 140.00839...

Logistic regression with multiple features

d=2 features

- Now, as we did last class, let's use both `'Glucose'` and `'BMI'` to predict diabetes.

```
In [33]: util.create_base_scatter(X_train, y_train)
```



2D graph

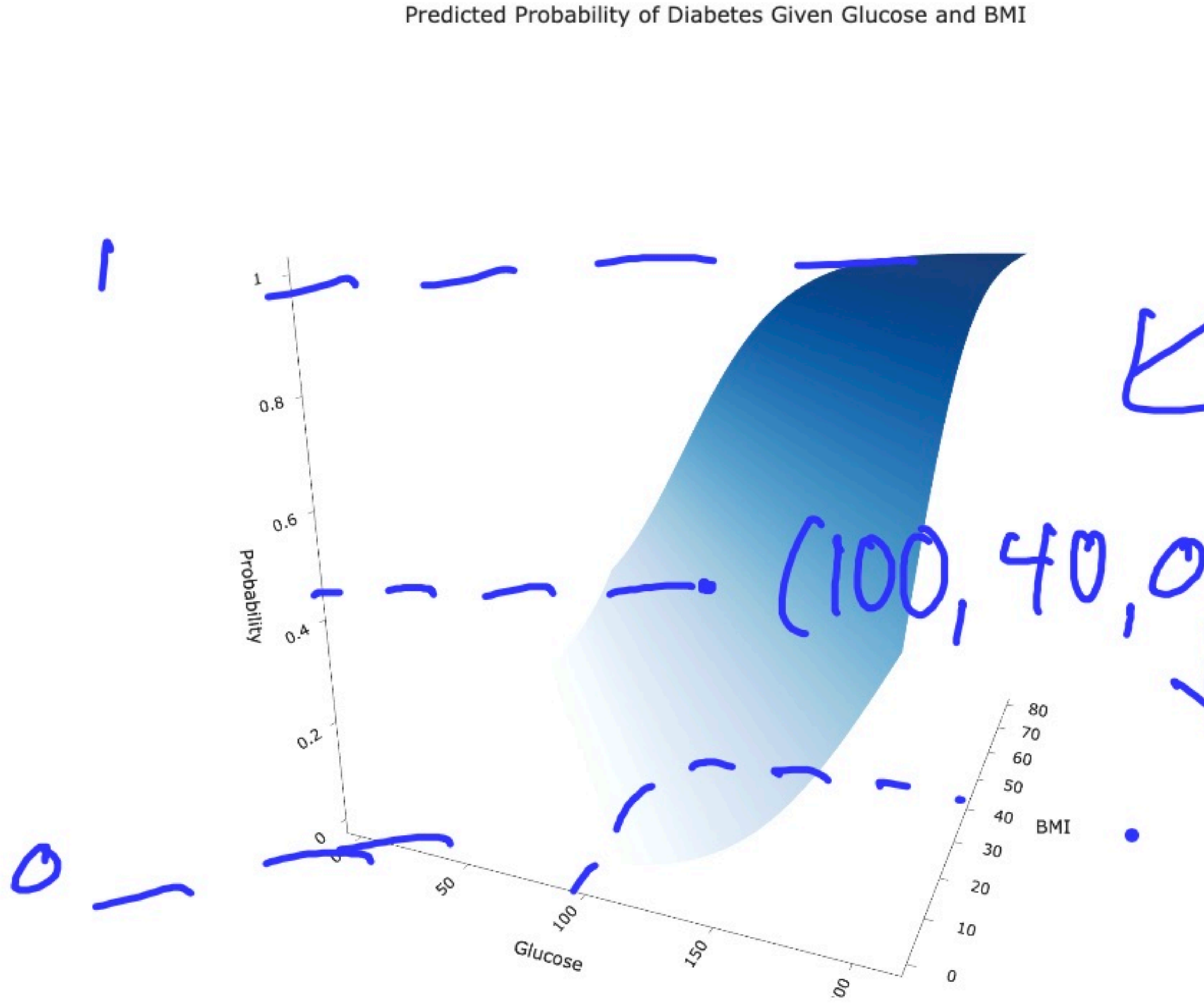
linear decision boundary IN THE FEATURE SPACE

- Recall, the logistic regression mo lass 1 (diabetes).

$$P(y_i = 1|\text{Glucose}_i, \text{BMI}_i) = \sigma(-7.85 + 0.04 \cdot \text{Glucose}_i + 0.08 \cdot \text{BMI}_i)$$

- The graph below shows the predicted probabilities of **class 1 (diabetes)** for different combinations of features.

```
In [36]: util.show_logistic(model_logistic_multiple, X_train, y_train)
```



Predicted Probability of Diabetes Given Glucose and BMI

(100, 40, 0.42)

↓ 42% predicted probability given Glucose=100, BMI=40