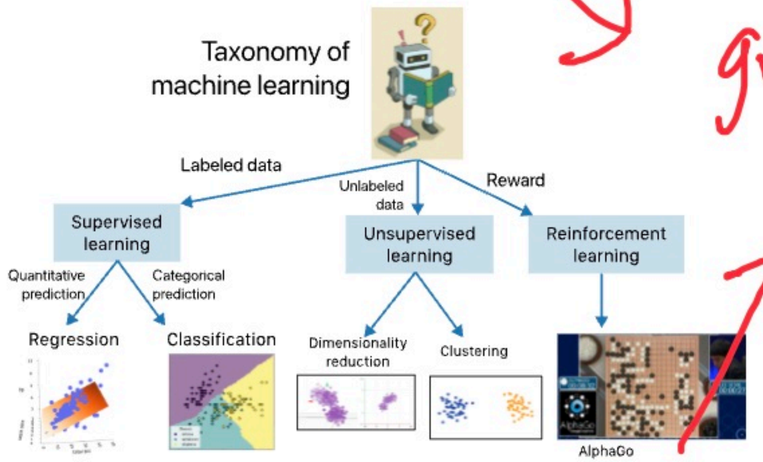
The taxonomy of machine learning

- So far, we've focused on building **regression** models.

- Regression is a form of **supervised learning**, in which the target variable (i.e., the y-values we're trying to predict) is **numerical**.

  For example, a predicted commute time could technically be any real number.



given X, predict y

- Next, we'll focus on **classification**, a form of supervised learning in which the target variable is **categorical**.

has diabetes, given other information about their health.

In [2]:
```
diabetes = pd.read_csv('data/diabetes.csv')
display_df(diabetes, cols=9)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.63	50	1
1	1	85	66	29	0	26.6	0.35	31	0
2	8	183	64	0	0	23.3	0.67	32	1
...	...	...	...	...	...	...	...	...	...
765	5	121	72	23	112	26.2	0.24	30	0
766	1	126	60	0	0	30.1	0.35	47	1
767	1	93	70	31	0	30.4	0.32	23	0

768 rows × 9 columns

In [3]:
```
# 0 means no diabetes, 1 means yes diabetes.
diabetes['Outcome'].value_counts()
```

Out[3]:
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

0: no
1: yes

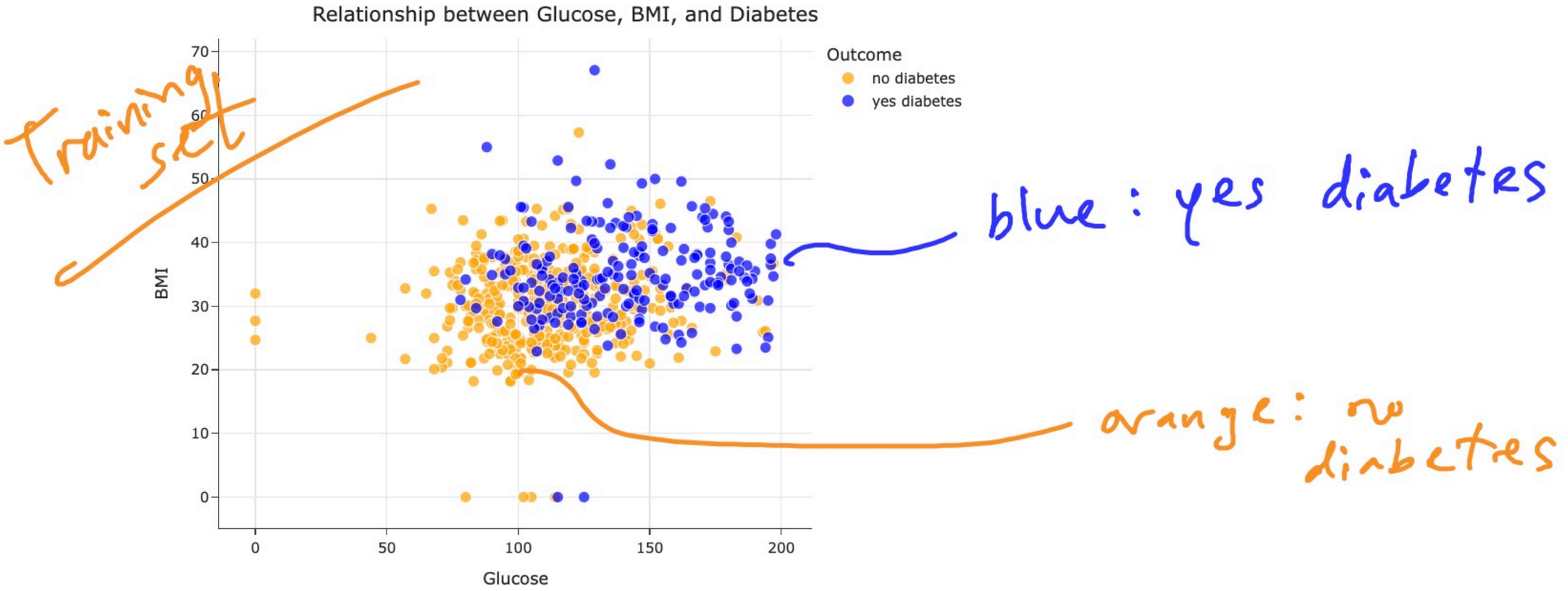
In [6]:
```
1 fig = util.create_base_scatter(X_train, y_train)
2 fig
```

Training set

blue: yes diabetes

orange: no diabetes

In [6]:
```
1 fig = util.create_base_scatter(X_train, y_train)
2 fig
```



- Using this dataset, how can we classify whether someone new (not already in the dataset) has diabetes, given their `'Glucose'` and `'BMI'`?

1. Finding the k **closest points** in the training set to $\vec{x}_{\text{new}}$.
2. Predicting that $\vec{x}_{\text{new}}$ belongs to the **most common class** among those k closest points.

In [7]:
```
1 fig
```

e.g. $\vec{x}_{\text{new}} = \begin{bmatrix} 160 \\ 40 \end{bmatrix}$

suppose we choose $k = 4$

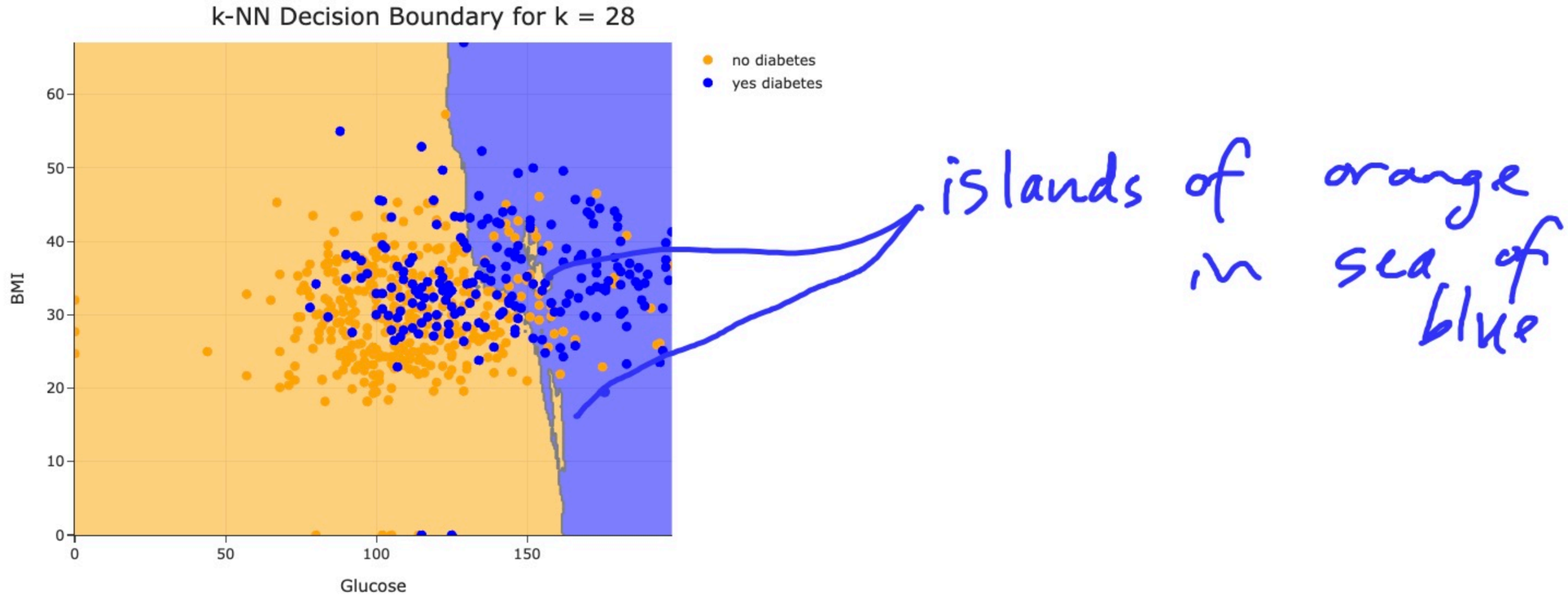
among closest 4, 3 do, 1 doesn't → predict yes diabetes

- The **decision boundaries** of a classifier visualize the regions in the feature space that separate different predicted classes.

- The decision boundaries for `model_knn` are visualized below.

  If a new person's feature vector lies in the **blue region**, we'd predict they **do have diabetes**, **otherwise**, we'd predict **they don't**.

In [13]:
```
1 util.visualize_k(28, X_train, y_train)
```

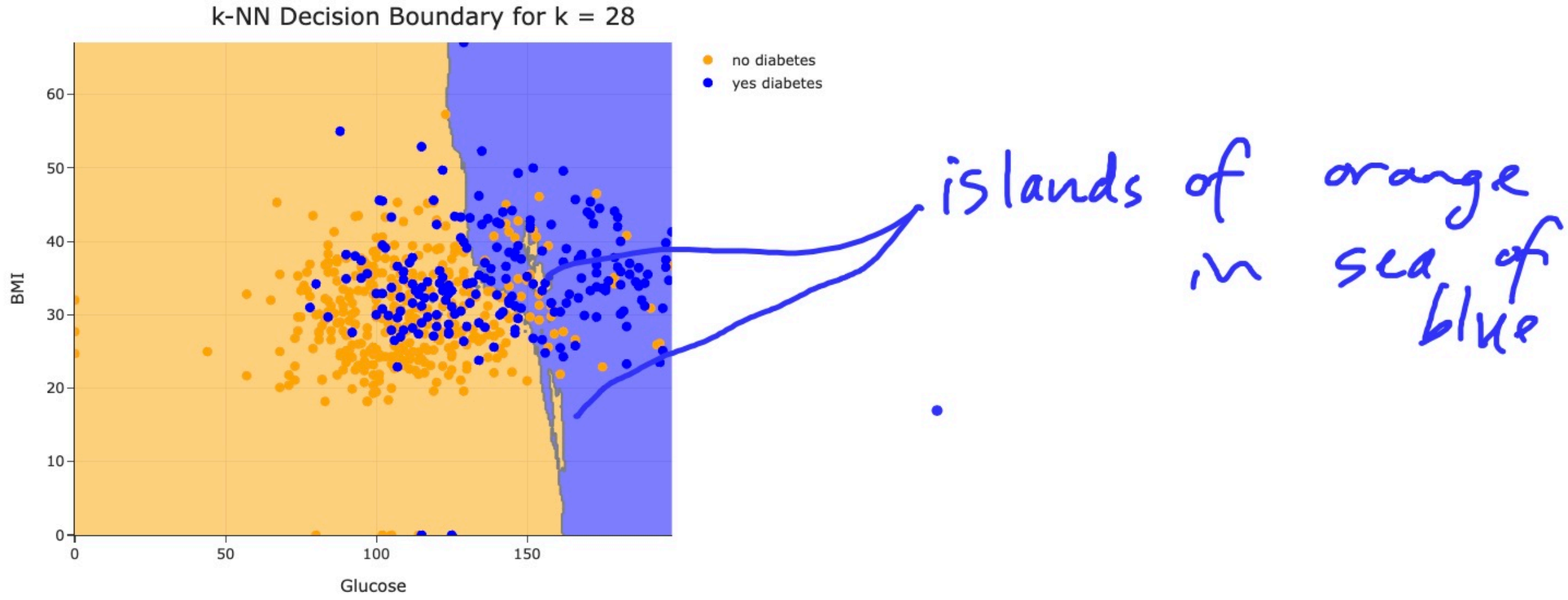


- The **decision boundaries** of a classifier visualize the regions in the feature space that separate different predicted classes.

- The decision boundaries for `model_knn` are visualized below.

  If a new person's feature vector lies in the **blue region**, we'd predict they **do have diabetes**, **otherwise**, we'd predict **they don't**.

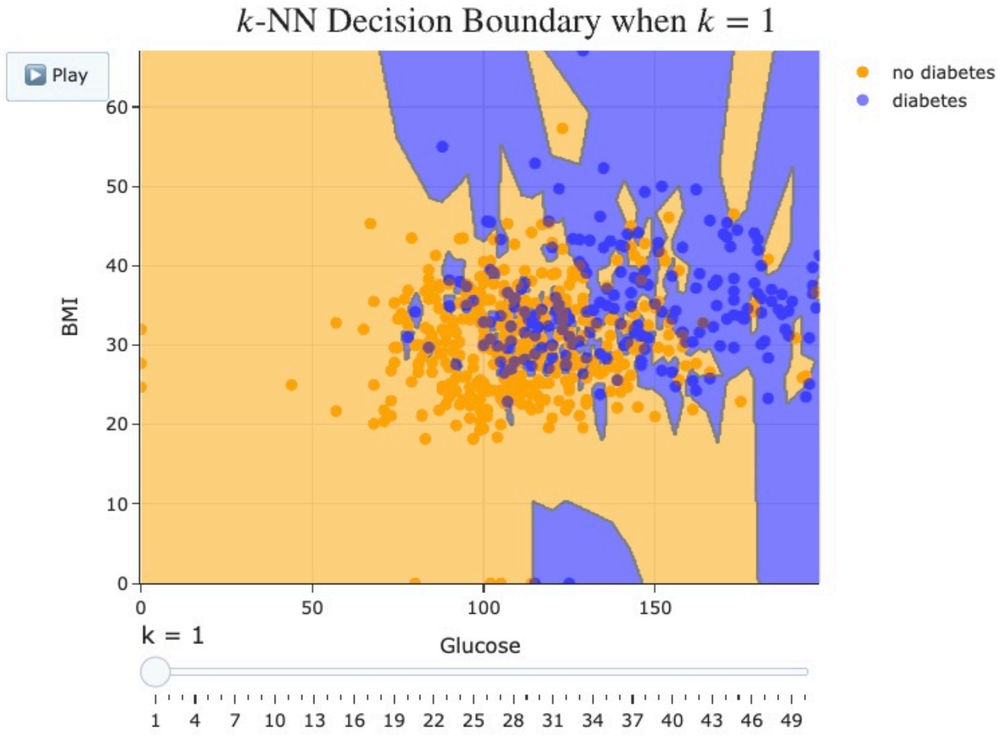
In [13]:
```
1 util.visualize_k(28, X_train, y_train)
```



What would the decision boundaries look like if k increased or decreased?

Play with the slider below to find out!

In [14]:
```
1 util.show_slider()
```

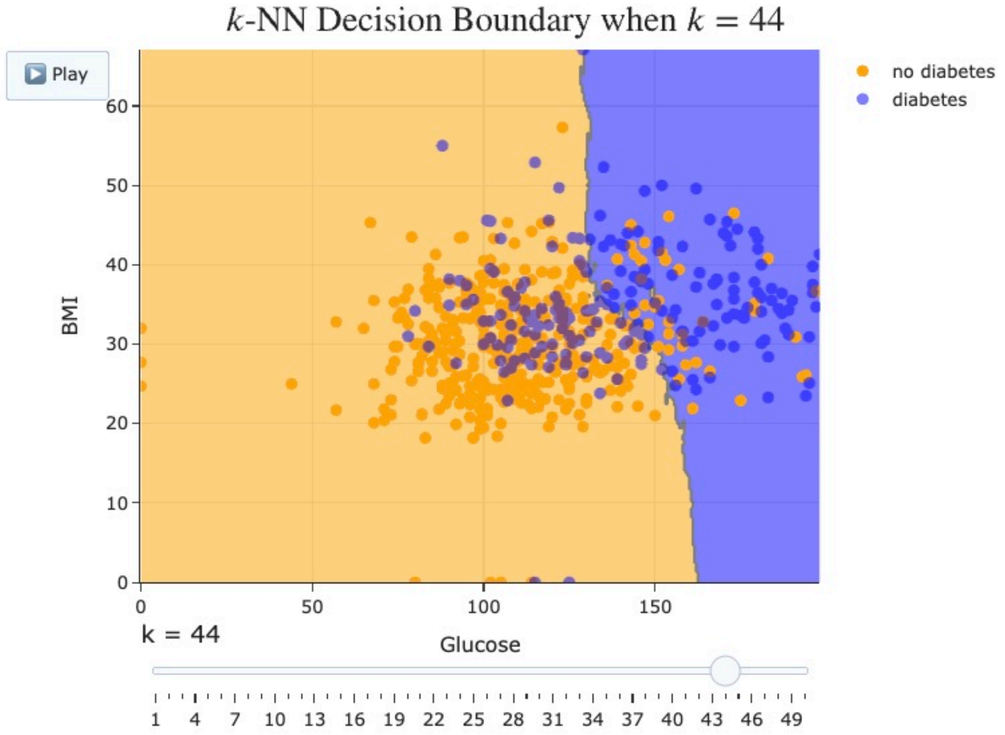


k=1: lots of islands

What would the decision boundaries look like if k increased or decreased?

Play with the slider below to find out!

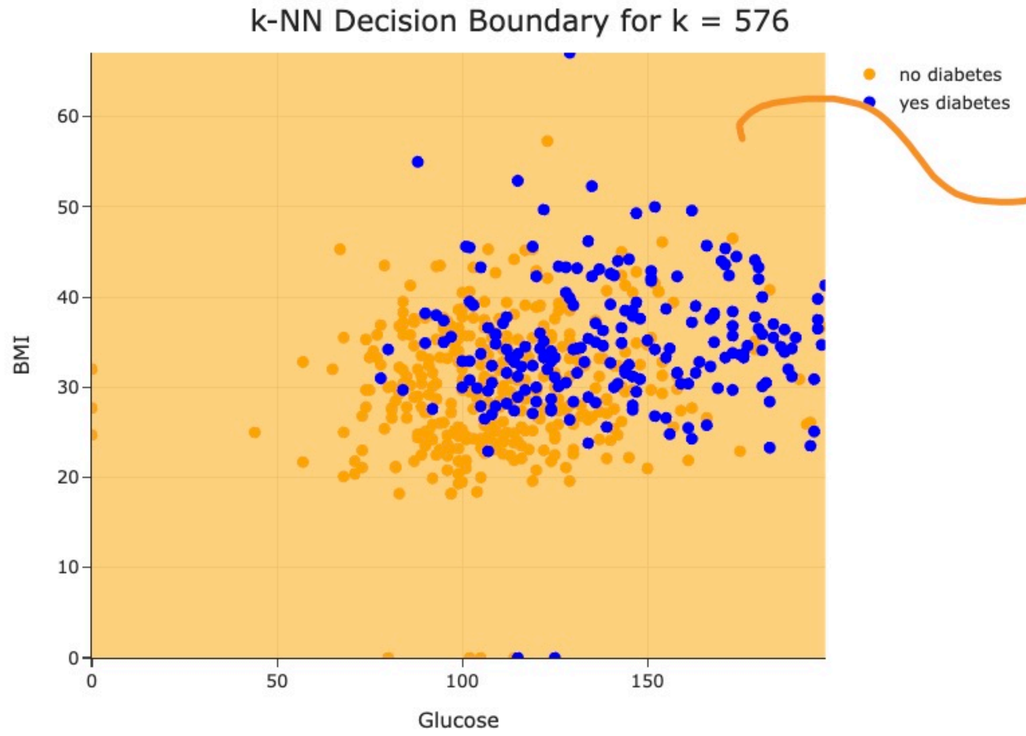
In [14]:
```
1 util.show_slider()
```



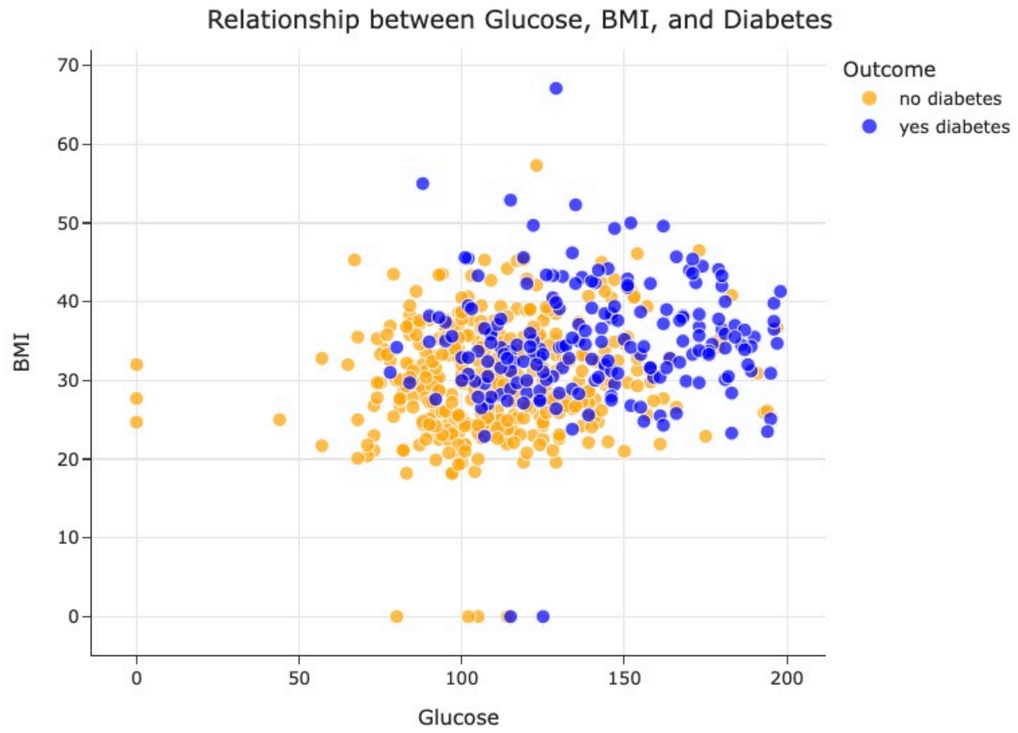
as k ↑,
decision boundary becomes more smooth!

What if $k = n$, the number of points in the training set?

```
In [15]: 1 util.visualize_k(576, X_train, y_train)
```



all orange (no), because no is more common than yes!.



why is training accuracy NOT 100% ?

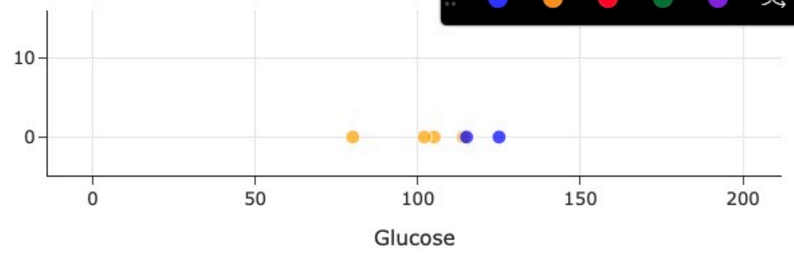
In [25]:
```
model_k1 = KNeighborsClassifier(n_neighbors=1)
model_k1.fit(X_train, y_train)
```

Out[25]:
```
KNeighborsClassifier(n_neighbors=1)
```

issue: overlapping points.

In [26]:
```
# Training accuracy – high, but not 100%.
model_k1.score(X_train, y_train)
```

Out[26]: 0.9913194444444444



In [25]:
```
model_k1 = KNeighborsClassifier(n_neighbors=1)
model_k1.fit(X_train, y_train)
```

Out[25]:
```
KNeighborsClassifier(n_neighbors=1)
```

as k ↑, model complexity ↓.

In [26]:
```
# Training accuracy — high, but not 100%.
model_k1.score(X_train, y_train)
```

Out[26]: 0.9913194444444444

In [27]:
```
# Accuracy on test set is lower than when k = 28!
model_k1.score(X_test, y_test)
```

Out[27]: 0.6822916666666666

In [28]:
```
test_scores['knn with k = 1'] = model_k1.score(X_test, y_test)
test_scores
```

Out[28]:
```
knn with k = 28    0.75
knn with k = 1     0.68
dtype: float64
```

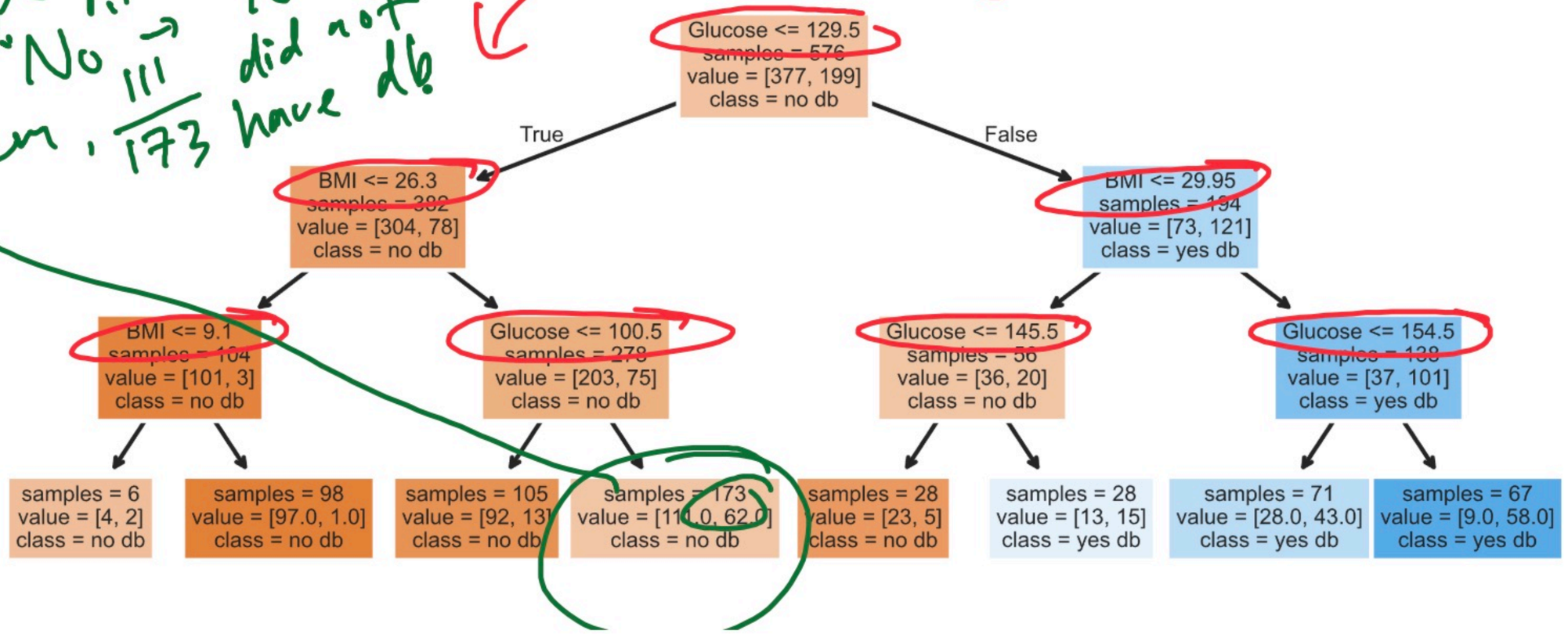
may look like:

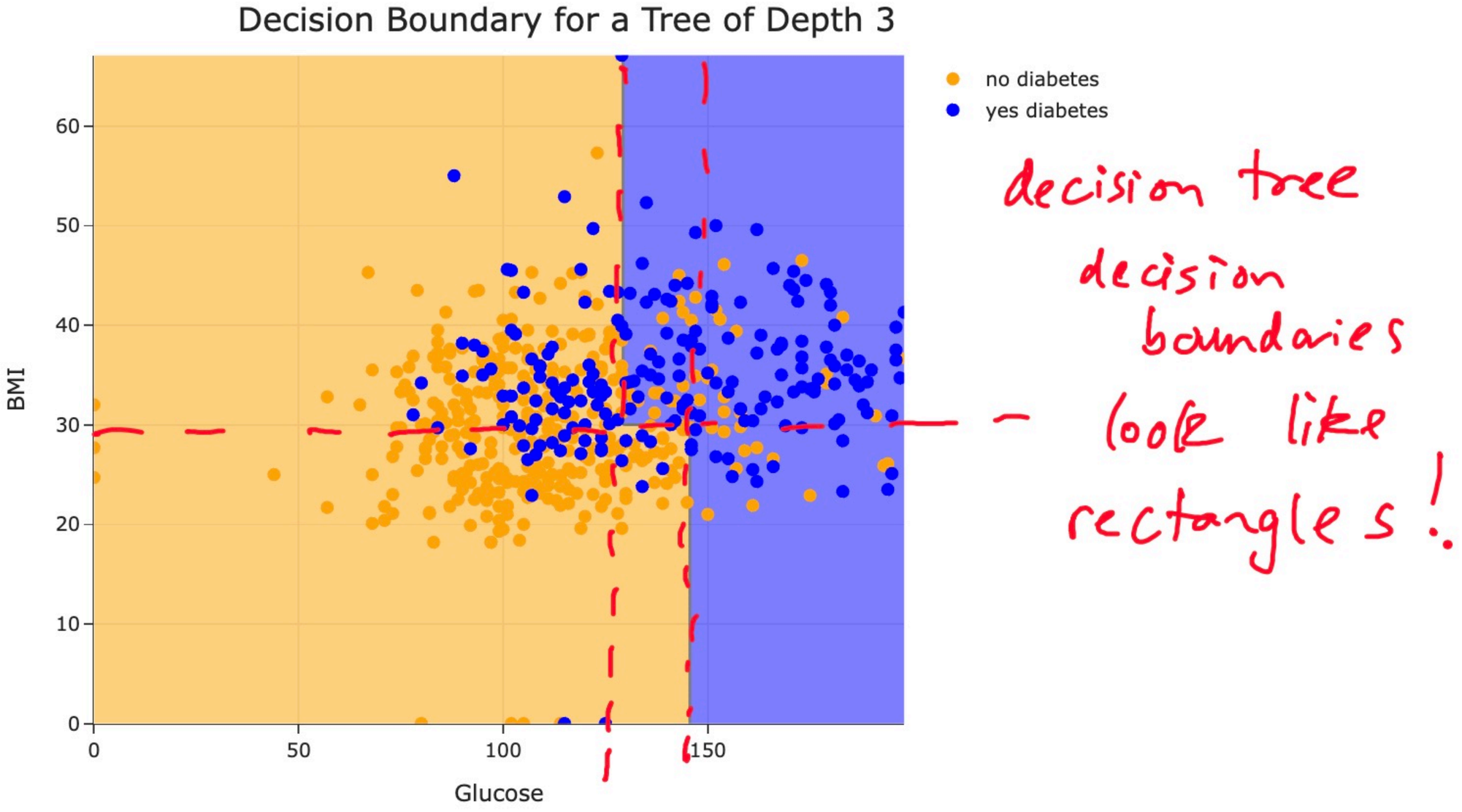
In training set, 173 people answered "Yes" → "No" !!! "No" → did not among them. 173 have db

Yes

No

each node is defined by a QUESTION




Decision Boundary for a Tree of Depth 3
no diabetes
yes diabetes
BMI
Glucose
0
10
20
30
40
50
60
0
50
100
150
decision tree
decision
boundaries
look like
rectangles!

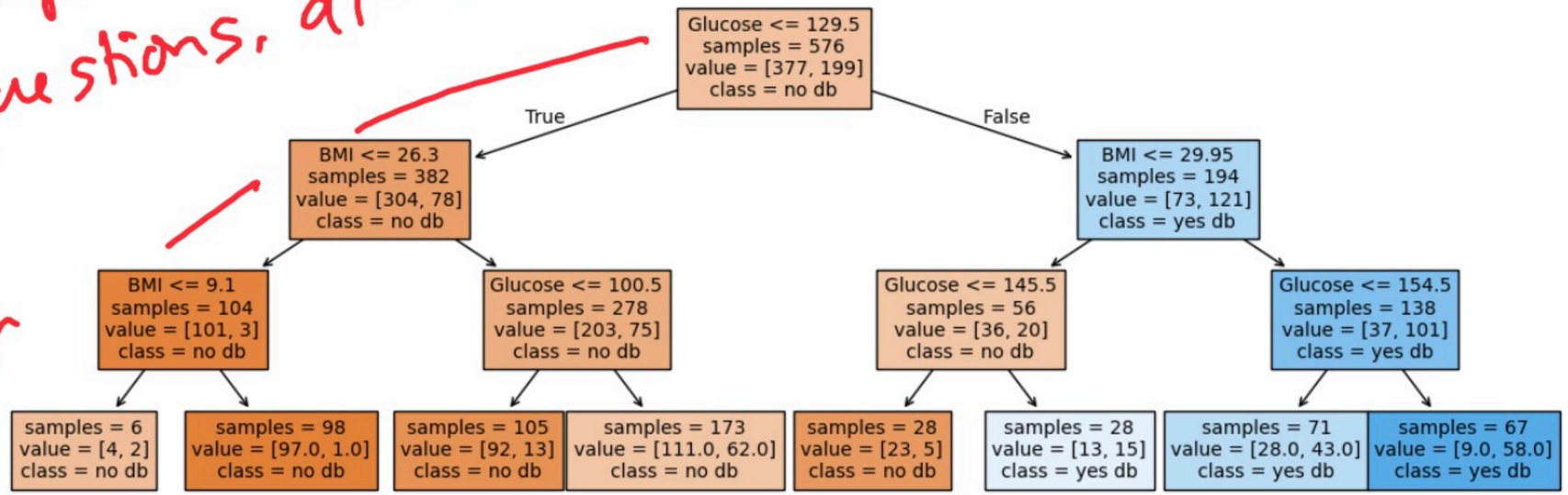
- Our fit decision tree is like a "flowchart", made up of a series of questions.

  It turns out `sklearn` provides us with a convenient way of visualizing this flowchart.

- As before, **orange is "no diabetes"** and **blue is "diabetes"**.

In [35]:
```
1 util.show_diabetes_decision_tree(model_tree, X_train);
```

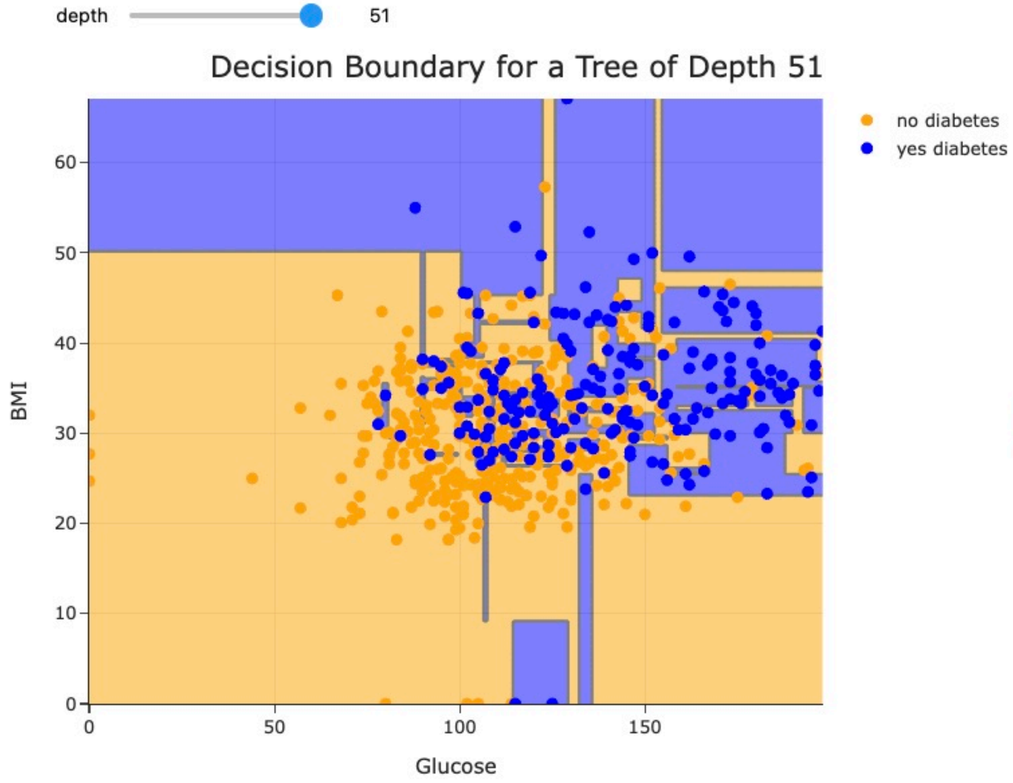
max_depth = 3
⇒ 3 questions, at most.



- One of the many hyperparameters we can tune is tree depth.

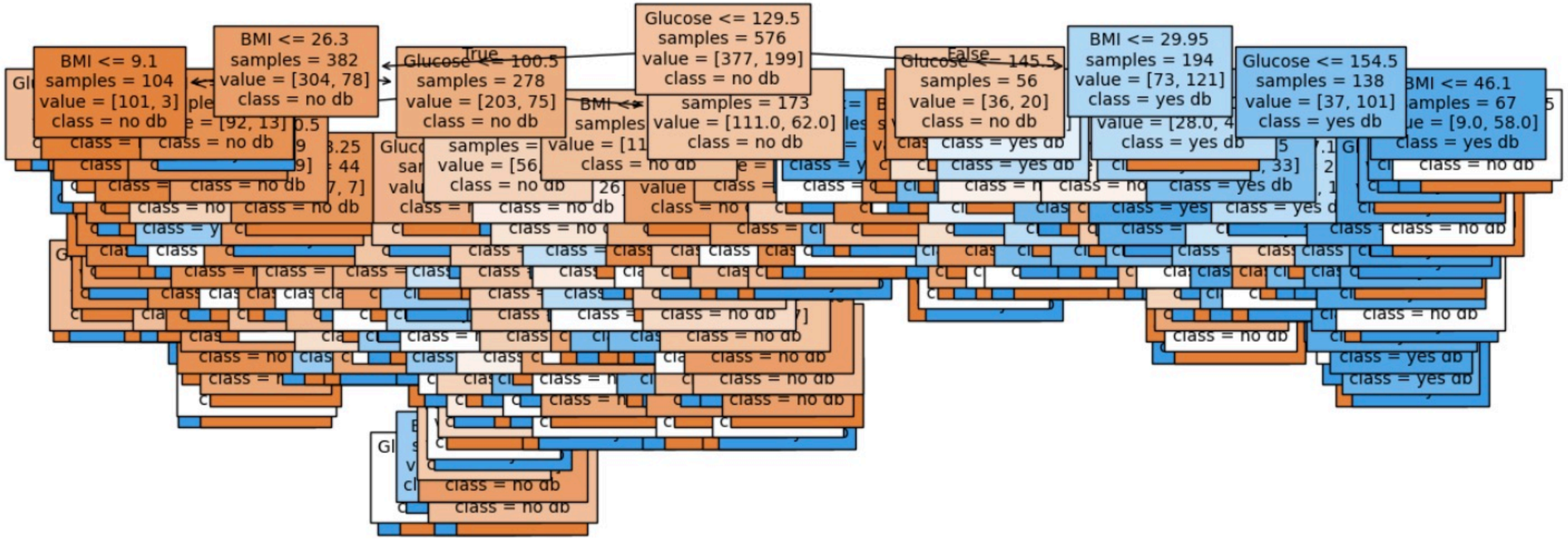
- What happens to the decision boundary of the resulting classifier if we increase `max_depth`?

In [36]:
```
1 interact(lambda depth: util.visualize_depth(depth, X_train, y_train), depth=(1, 51));
```



as depth ↑, complexity ↑.

In [42]:
```
1 util.show_diabetes_decis
```



- The tree is **extremely overfit** to the training set, and very deep!

In [43]:
```
1 # Training accuracy. This number should look familiar!
2 model_tree_no_max.score(X_train, y_train)
```

Out[43]: 0.9913194444444444

In [44]:
```
1 model_tree_no_max.tree_.max_depth
```

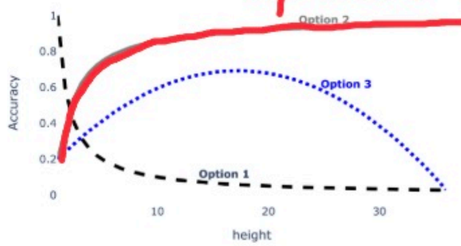
Out[44]: 21

trees are fit non-deterministically! (ties for "best questions" chosen randomly)

Activity

`ChickenClassifier`s have many hyperparameters, one of which is `height`. As we increase the value of `height`, the model variance of the resulting `ChickenClassifier` also increases.

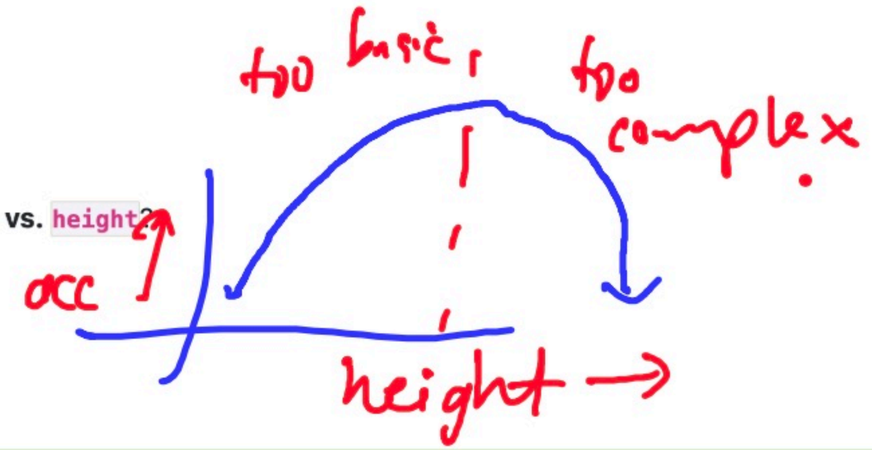
First, we consider the training and testing accuracy of a `ChickenClassifier` trained using various values of `height`. Consider the plot below.



as height ↑, overfits more to training set (more complexity)

Which of the following depicts **training accuracy vs. `height`**?

- Option 1
- Option 2
- Option 3



Which of the following depicts **testing accuracy vs. `height`**?

- Option 1
- Option 2
- Option 3

- The results of 100 Michigan Medicine COVID tests are given below.

	Predicted Negative	Predicted Positive
Actually Negative	TN = 90 ✅	FP = 1 ❌
Actually Positive	FN = 8 ❌	TP = 1 ✅

Michigan Medicine test results

= 91!

- 🤔 **Question:** What is the accuracy of the test?

$$\text{accuracy} = \frac{\text{\# points classified correctly}}{\text{\# points}}$$

- 🙋 **Answer:**

$$\text{accuracy} = \frac{TP + TN}{TP + FP + FN + TN} = \frac{1 + 90}{100} = 0.91$$

- **Followup:** At first, the test seems good. But, suppose we build a classifier that predicts that **nobody has COVID**. What would its accuracy be?

- The results of 100 Michigan Medicine COVID tests are given below.

	Predicted Negative	Predicted Positive
Actually Negative	TN = 90 ✅	FP = 1 ❌
Actually Positive	FN = 8 ❌	TP = 1 ✅

Michigan Medicine test results

= 91

- 🤔 **Question:** What is the accuracy of the test?

$$\text{accuracy} = \frac{\text{\# points classified correctly}}{\text{\# points}}$$

- 🙋 **Answer:**

$$\text{accuracy} = \frac{TP + TN}{TP + FP + FN + TN} = \frac{1 + 90}{100} = 0.91$$

- **Followup:** At first, the test seems good. But, suppose we build a classifier that predicts that **nobody has COVID**. What would its accuracy be?

Recall

	Predicted Negative	Predicted Positive
Actually Negative	TN = 90 ✅	FP = 1 ❌
Actually Positive	FN = 8 ❌	TP = 1 ✅

Michigan Medicine test results

$\frac{1}{1+8}$.

- 🤔 **Question:** What proportion of individuals who actually have COVID did the test **identify**?
- 🙋 **Answer:** $\frac{1}{1+8} = \frac{1}{9} \approx 0.11$.
- More generally, the **recall** of a binary classifier is the proportion of **actually positive instances** that are correctly classified. We'd like this number to be as close to 1 (100%) as possible.

$$\text{recall} = \frac{TP}{\text{\# actually positive}} = \frac{TP}{TP + FN}$$

Precision

precision = TP / (TP+FP)

recall = TP / (TP+FN)

	Predicted Negative	Predicted Positive
Actually Negative	TN = 0 ✅	FP = 91 ❌
Actually Positive	FN = 0 ❌	TP = 9 ✅

everyone-has-COVID classifier

- The **precision** of a binary classifier is the proportion of **predicted positive instances** that are correctly classified. We'd like this number to be as close to 1 (100%) as possible.

$$\text{precision} = \frac{TP}{\#\text{ predicted positive}} = \frac{TP}{TP + FP}$$

- To compute precision, look at the **right (positive) column** of the above confusion matrix.
  Tip: A good way to remember the difference between precision and recall is that in the denominator for 🅿recision, both terms have 🅿 in them (TP and FP).

- Note that the "everyone-has-COVID" classifier has perfect recall, but a precision of $\frac{9}{9+91} = 0.09$, which is quite low.

- 🚨 **Key idea:** There is a "tradeoff" between precision and recall. Ideally, you want both to be high. For a particular prediction task, one may be important than the other.

Precision

	Predicted Negative	Predicted Positive
Actually Negative	TN = 0 ✅	FP = 91 ❌
Actually Positive	FN = 0 ❌	TP = 9 ✅

everyone-has-COVID classifier

stop here for today

- The **precision** of a binary classifier is the proportion of **predicted positive instances** that are correctly classified. We'd like this number to be as close to 1 (100%) as possible.

$$\text{precision} = \frac{TP}{\#\text{ predicted positive}} = \frac{TP}{TP + FP}$$

- To compute precision, look at the **right (positive) column** of the above confusion matrix.
  Tip: A good way to remember the difference between precision and recall is that in the denominator for 🅿recision, both terms have 🅿 in them (TP and FP).

- Note that the "everyone-has-COVID" classifier has perfect recall, but a precision of $\frac{9}{9+91} = 0.09$, which is quite low.

- 🚨 **Key idea:** There is a "tradeoff" between precision and recall. Ideally, you want both to be high. For a particular prediction task, one may be important than the other.