

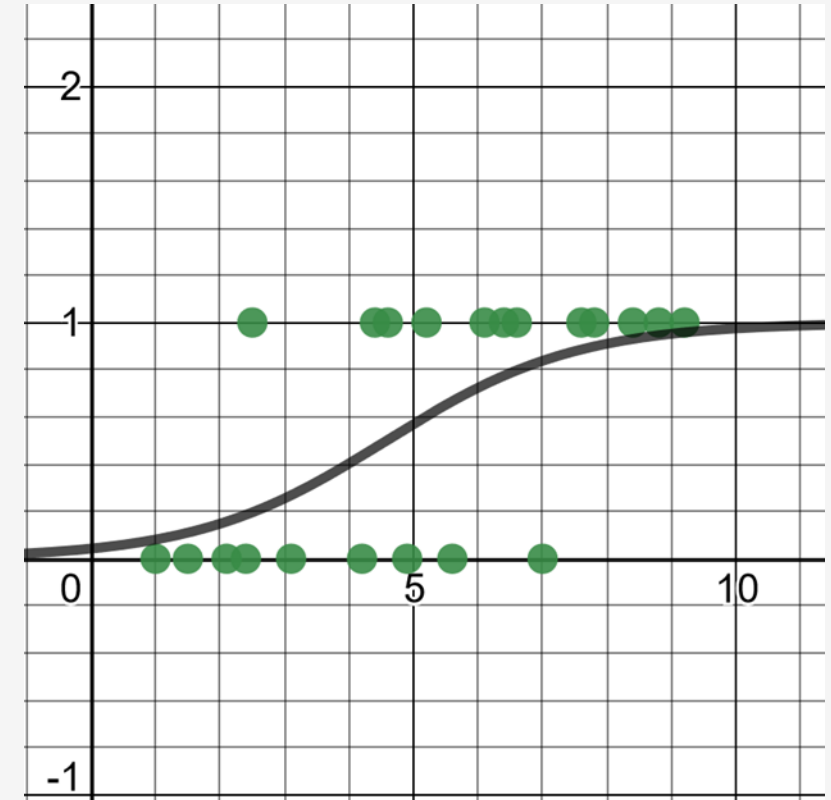
Logistic Regression and Classification

Overview

Agenda

Here is what we will do for the next 2 hours:

- 1 Logistic Regression Fundamentals
- 2 DISCUSSION: Space Shuttle Challenger
- 3 Preparing and Rescaling Data
- 4 ML Classification Validation



Section I

Logistic Regression Fundamentals

Classifying Things

So far we just did regressions via linear regression.

However, linear regressions are awkward to use for classification.

- Lines extend in a straight direction for positive infinity and negative infinity, well outside a range of acceptable values.
- Lines do not do a good job representing a probability and staying within the limits of 0.0 and 1.0.
- When doing classification, probability is a critical tool.



Hot dog: 0.92

Classifying Things

Classification tasks are pretty common in machine learning:

- How do I classify images of *dogs* versus *cats*?
- Will a shipment be most likely be *late*, *early*, or *on-time*?
- Is this email *spam* or *not spam*?
- Will this movie get *1 star*, *2 stars*... or *5 stars*?

There are several machine learning algorithms that work well for classification, and we are going to learn first about Logistic Regression.

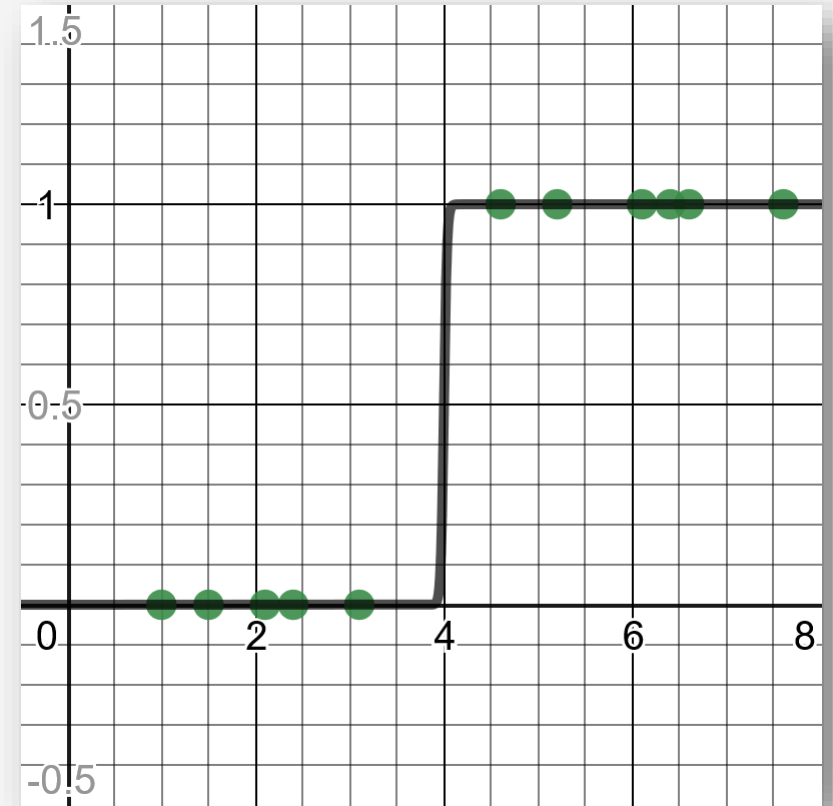


Hot dog: 0.92

Logistic Regression Intuition

Imagine you have 11 patients exposed to a chemical for x hours, and you plot whether they exhibited symptoms (1) or not (0).

Plotting our patient data (right), we can easily eyeball a clear cutoff at 4 hours where patients transition from ***not showing symptoms (0)*** to ***showing symptoms (1)***.



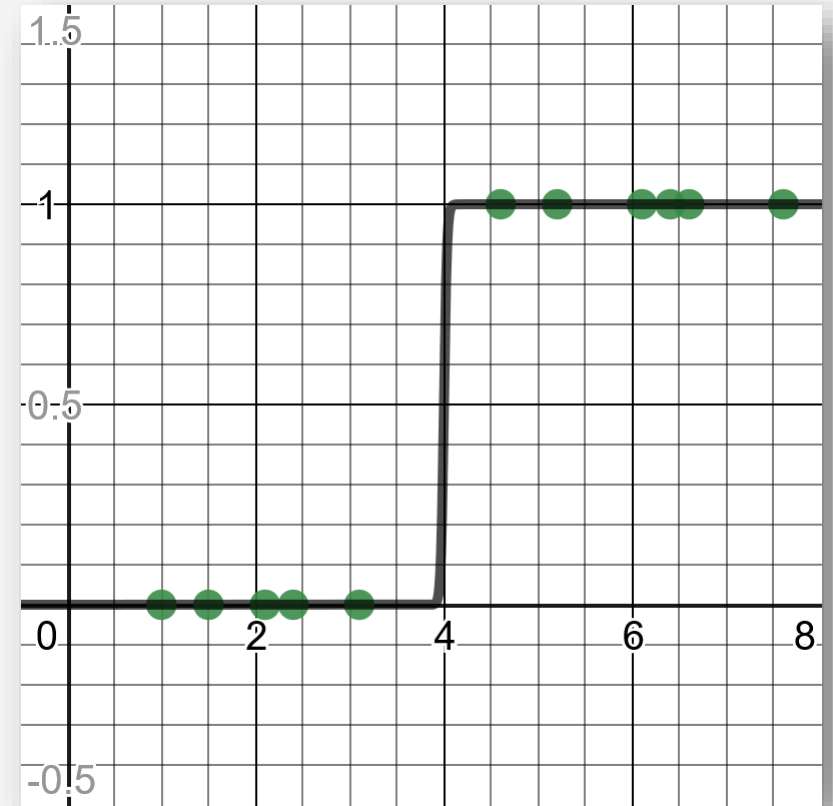
<https://www.desmos.com/calculator/prs2p0sofc>

Logistic Regression Intuition

This indicates any patient exposed for less than 4 hours will have a 0% chance of showing symptoms, but greater than 4 will have a 100% chance of showing symptoms.

Because there is a distinct separation at 4 hours, a logistic regression is going to “jump” from 0% to 100% at that boundary.

Of course, real life rarely works out this way...



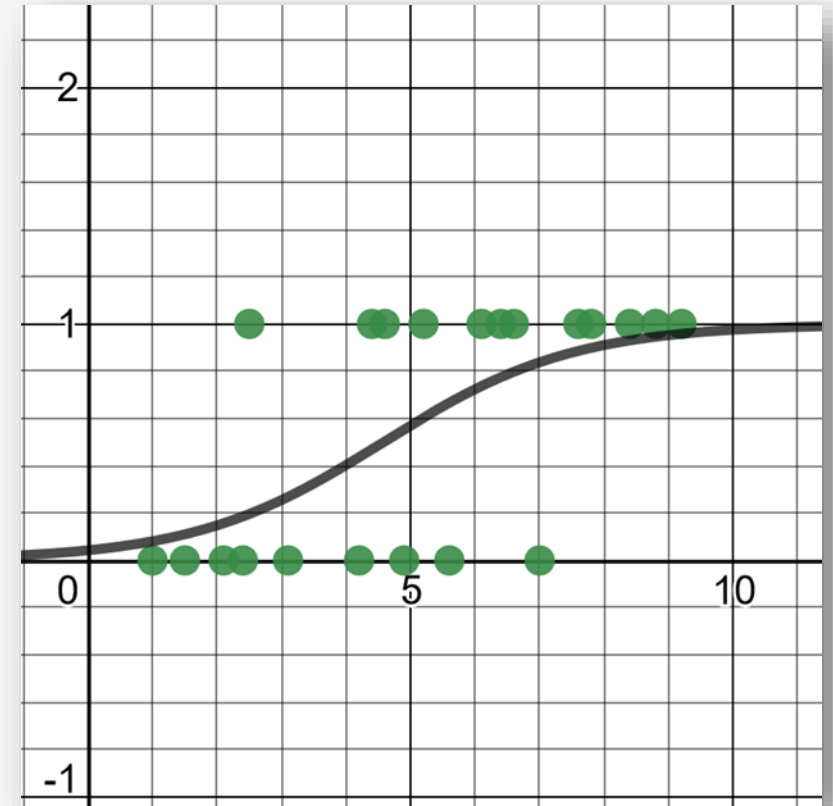
<https://www.desmos.com/calculator/prs2p0sofc>

Logistic Regression Intuition

Now let's say you gathered more data and got a realistic picture, where the middle of the range has a mix of patients showing symptoms and not showing symptoms.

The way to interpret this is the probability of patients showing symptoms gradually increases with each hour of exposure.

Because of this overlap of points in the middle, there is no distinct cutoff when patients show symptoms, but rather a gradual transition from 0% probability to 100% probability ("0" and "1").

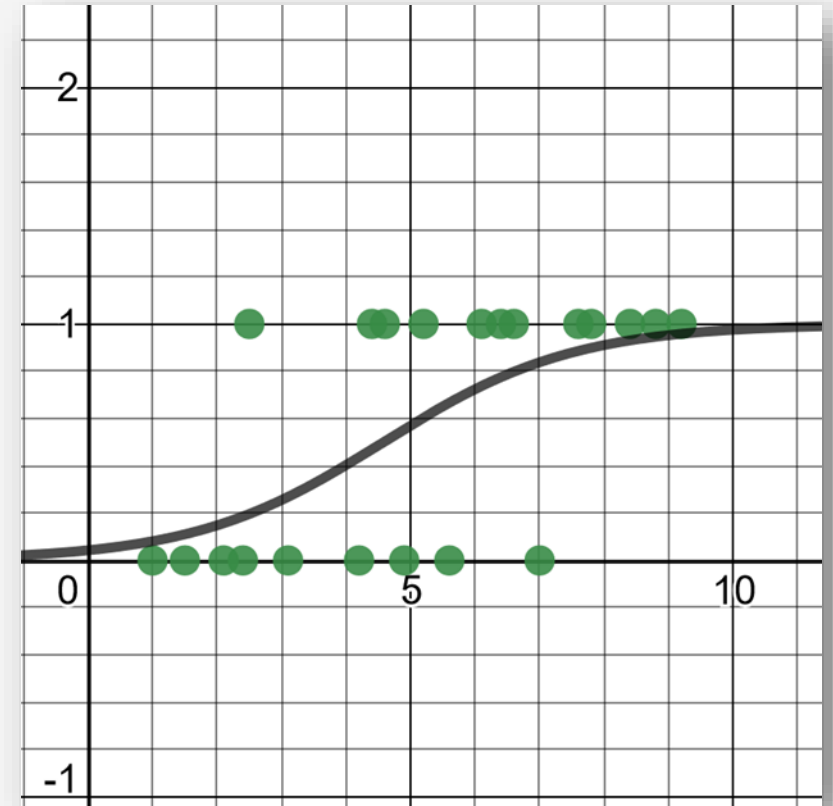


<https://www.desmos.com/calculator/bsqtqfians>

Logistic Regression Intuition

More technically, a logistic regression results in a curve indicating a probability of belonging to the **true** (1) category, which in this case means ***a patient showed symptoms***.

As the hours of chemical exposure increases, the number of patients showing symptoms increases, and thus the probability of showing symptoms increases.



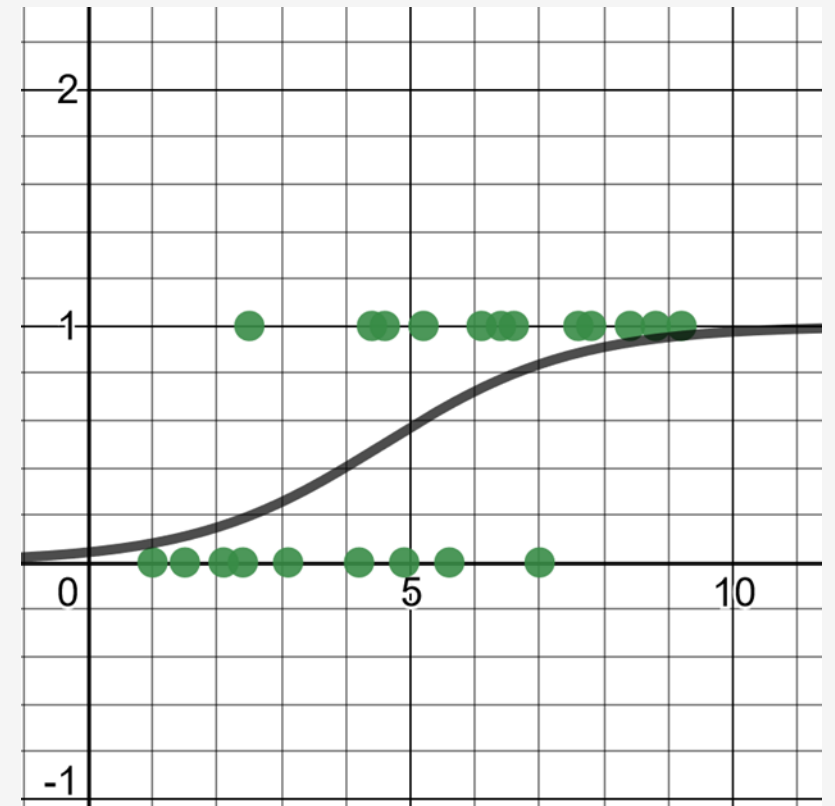
<https://www.desmos.com/calculator/bsqtqfians>

Logistic Regression

Logistic Regression is a classification tool that predicts a **true** or **false** value for one or more variables.

Training data must have outcomes of 0 (false) or 1 (true), but the regression will output a probability value between 0 and 1.

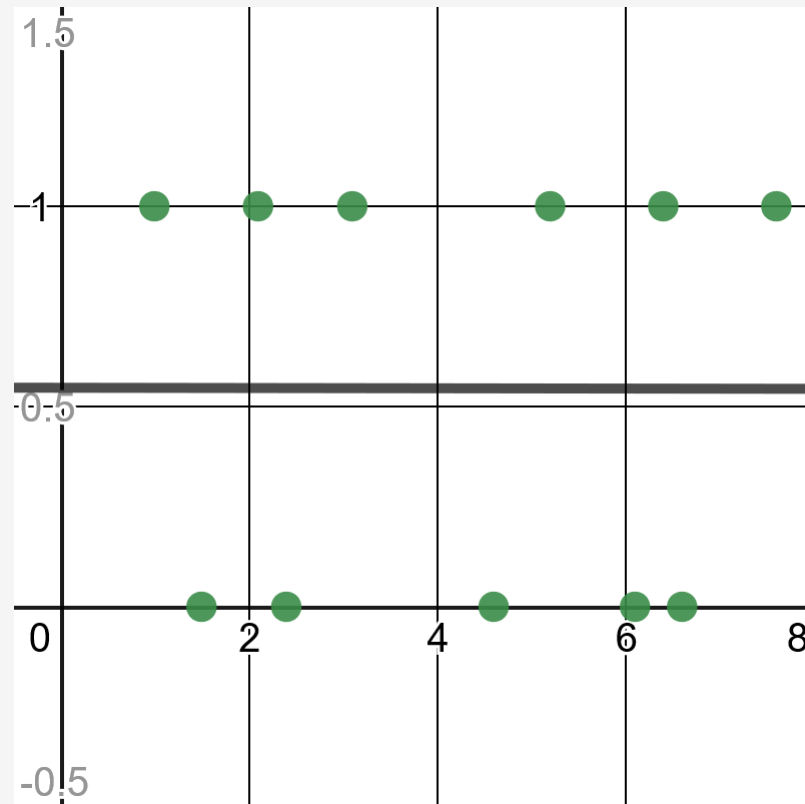
- An S-shaped curve (a **logistic function**) is fit to the points and then used to predict probability.
- If a predicted value (the y-axis) is less than .5 it is typically categorized as false (0), and if the predicted value is greater than/equal to .5 it is typically categorized as true (1).



<https://www.desmos.com/calculator/bsqtqfians>

When Not To Use Logistic Regression

Of course, if there is no transitional trend in your data then you should not use logistic regression.



<https://www.desmos.com/calculator/cmeksxk5rk>

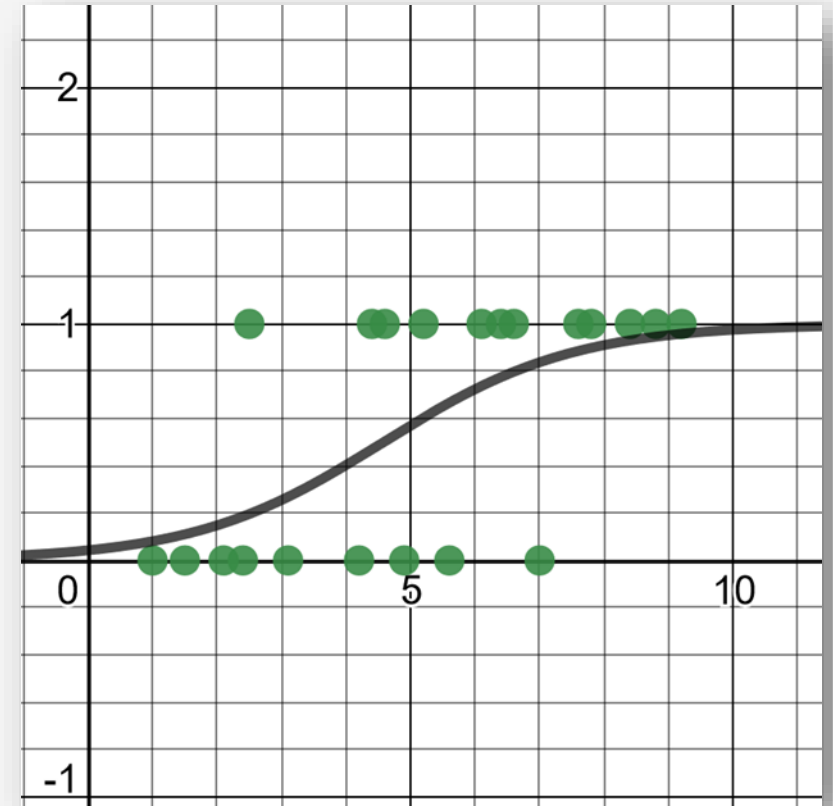
Just Show Me the Math!

For a single independent variable x to predict a dependent probability variable y , you need to fit the data to the logistic function:

$$y = \frac{1.0}{1.0 + e^{-(\beta_0 + \beta_1 x)}}$$

You can express this in Python as:

```
def predict_probability(x):  
    p = 1.0 / (1.0 + math.exp(-(b0 + b1 * x)))  
    return p
```



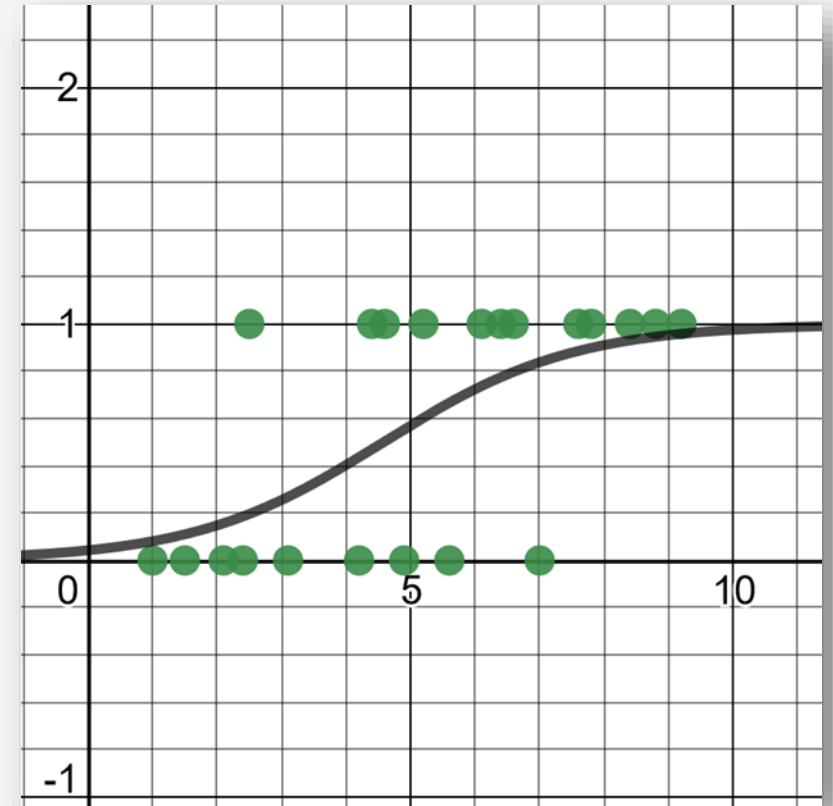
<https://www.desmos.com/calculator/blfcwrlnuw>

Just Show Me the Math!

You may also see this logistic function expressed as:

$$y = \frac{e^{\beta_0 + \beta_1 x}}{1.0 + e^{\beta_0 + \beta_1 x}}$$

However it is the same and just algebraically expressed differently.



<https://www.desmos.com/calculator/blfcwrlnuw>

Just Show Me the Math!

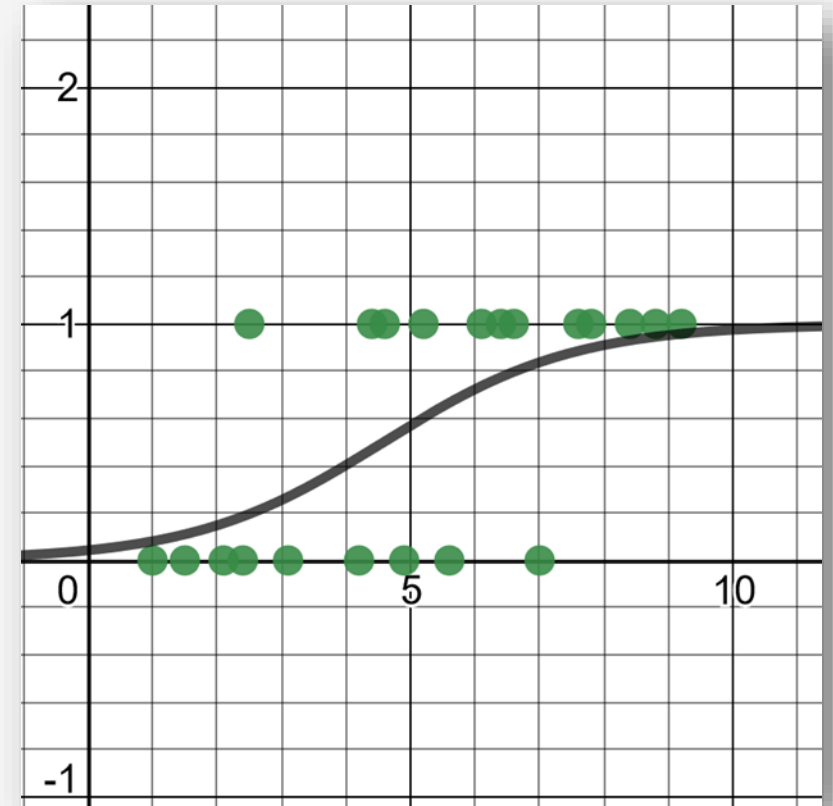
$$y = \frac{1.0}{1.0 + e^{-(\beta_0 + \beta_1 x)}}$$

```
def predict_probability(x):  
    p = 1.0 / (1.0 + math.exp(-(b0 + b1 * x)))  
    return p
```

Notice the expression $\beta_0 + \beta_1 x$ is linear, and this known as the **log odds function** which is translated logarithmically into a probability.

In the interest of time, we will avoid going into proofs and mathematical details about how this function works.

Just know it produces this S-shaped curve we need to output a probability between 0 and 1.



<https://www.desmos.com/calculator/blfcwrlnuw>

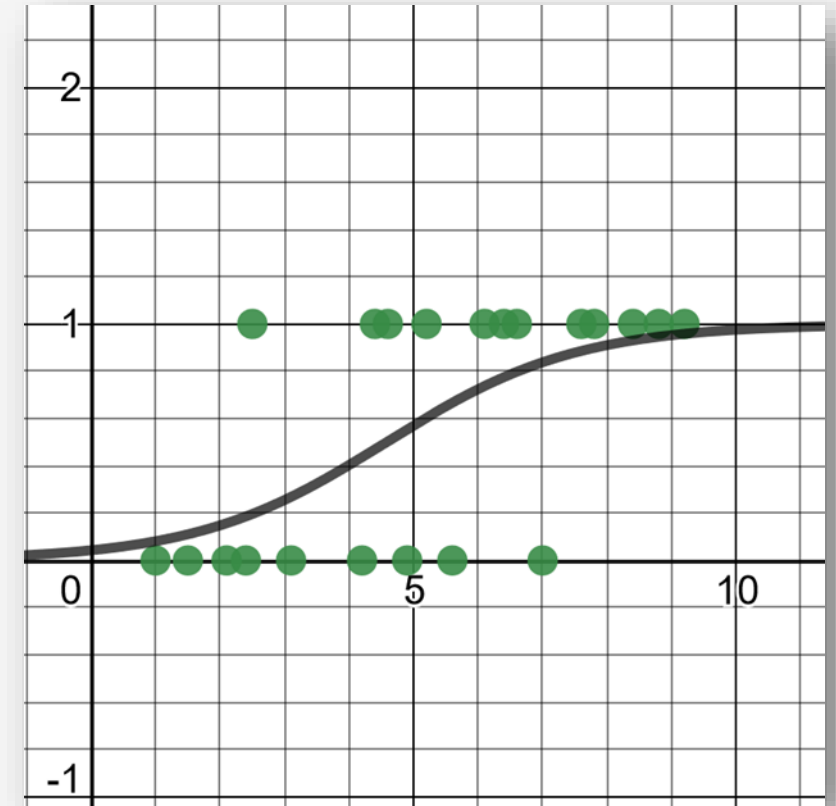
Just Show Me the Math!

$$y = \frac{1.0}{1.0 + e^{-(\beta_0 + \beta_1 x)}}$$

```
def predict_probability(x):  
    p = 1.0 / (1.0 + math.exp(-(b0 + b1 * x)))  
    return p
```

We need to solve for β_0 and β_1 , but we cannot use least squares like in linear regression.

- We are trying to maximize probability of the curve predicting correctly, not finding the best fit.
- Using hill climbing, we need to find β_0 and β_1 that produces the maximum likelihood.



<https://www.desmos.com/calculator/blfcwrlnuw>

Maximum Likelihood

Maximum likelihood is a technique to estimate parameters that have the highest probability of outputting the observed data.

In our case, we need to find values for β_0 and β_1 that will yield the highest likelihood of outputting the correct true/false values.

Remember that our logistic function outputs a probability y for a given value x .

$$y = \frac{1.0}{1.0 + e^{-(\beta_0 + \beta_1 x)}}$$

Maximum Likelihood

Let's say during our hill-climbing, we test parameters $\beta_0 = -3.17$ and $\beta_1 = 0.69$

$$y = \frac{1.0}{1.0 + e^{-(-3.17 + 0.69x)}}$$

To calculate the total likelihood for these parameters...

1. Get the true (1) data, calculate the probability y for each x value, and multiply them together.
2. Get the false (0) data, calculate the probability $(1.0 - y)$ for each x value, and multiply them together.
3. Multiply the two products above together, and that is your total likelihood.

Maximum Likelihood – Avoiding Floating Point Underflow

However, multiplying this many decimals together can cause floating point underflow.

With a clever mathematical hack, we can remedy this by using logarithmic addition instead of multiplication.

$$y = \log\left(\frac{1.0}{1.0 + e^{-(-3.17+0.69x)}}\right)$$

If you need to learn about logarithms, YouTube is the best place to get crash coursed.

PatrickJMT: <https://youtu.be/AAW7WRFBKdw>

Don't Memorize: <https://youtu.be/4UNkQcBrLaQ>

Maximum Likelihood – Avoiding Floating Point Underflow

$$y = \log\left(\frac{1.0}{1.0 + e^{-(-3.17 + 0.69x)}}\right)$$

To calculate the total likelihood for these parameters, but avoid floating point underflow...

1. Get the true (1) data, calculate the probability y for each x value, pass it to a ***log()*** function, then sum the values.
2. Get the false (0) data, calculate the probability $(1.0 - y)$ for each x value, pass it to a ***log()*** function, then sum the values.
3. Sum the two values above together, pass it to the ***exp()*** function to undo the logarithm, and that is your total likelihood.

Maximum Likelihood

We now know how to calculate the likelihood for a given set of parameters.

To calculate the maximum likelihood...

1. Randomly adjust the β_0 and β_1 values (using hill-climbing, gradient descent, or other optimization)
2. Calculate the likelihood (as shown in the previous slide)
3. If the likelihood improves, keep the changes to β_0 and β_1 , otherwise revert.
4. Do this for as many iterations as necessary, until the likelihood stops improving.

Logistic Regression from Scratch

https://github.com/thomasniel/oreilly_math_fundamentals_data_science/blob/main/logistic_regression/1_logistic_regression_from_scratch.py

Hands-On: Logistic Regression with Scikit-Learn

```
import pandas as pd
from sklearn.linear_model import LogisticRegression

data = pd.read_csv("https://tinyurl.com/y2coco07")

# grab independent variable column
inputs = data.iloc[:, :-1]

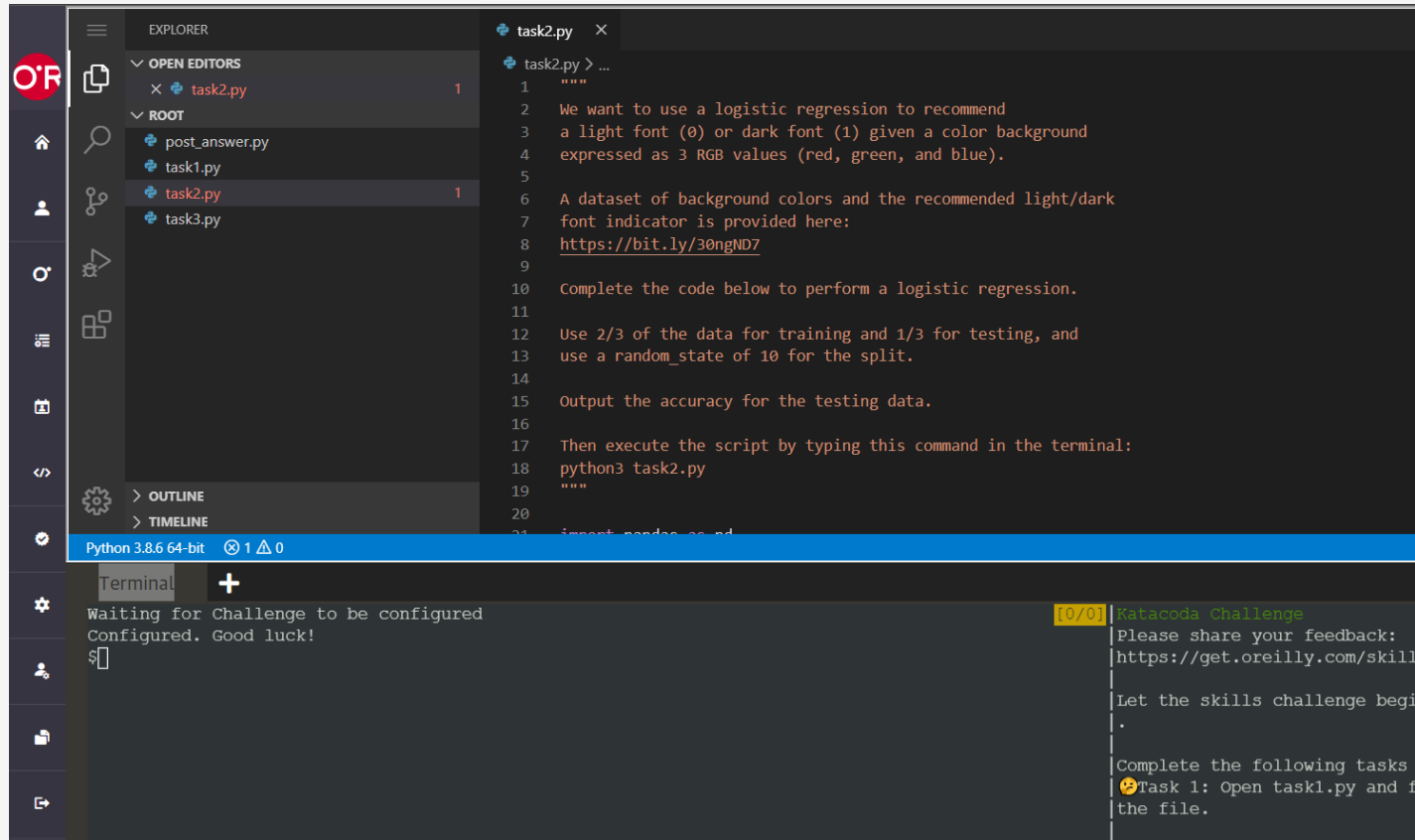
# grab dependent variable column
output = data.iloc[:, -1]

# build logistic regression, note CVLogisticRegression is also recommended to use cross-validation
fit = LogisticRegression().fit(inputs, output)

# Print coefficients:
print("COEFFICIENTS: {0}".format(fit.coef_.flatten()))
print("INTERCEPT: {0}".format(fit.intercept_.flatten()))

# Test a prediction
print("x=1.5, y={0}".format(fit.predict([[1.5]])))
print("x=18.5, y={0}".format(fit.predict([[18.5]])))
```

Katacoda – Task 1



The screenshot displays the Katacoda IDE interface. On the left, the Explorer panel shows the file structure with 'task2.py' selected. The main editor area displays the content of 'task2.py', which includes instructions for a logistic regression task. The terminal at the bottom shows the initial setup and a challenge message.

```
task2.py
1 """
2 We want to use a logistic regression to recommend
3 a light font (0) or dark font (1) given a color background
4 expressed as 3 RGB values (red, green, and blue).
5
6 A dataset of background colors and the recommended light/dark
7 font indicator is provided here:
8 https://bit.ly/30ngND7
9
10 Complete the code below to perform a logistic regression.
11
12 Use 2/3 of the data for training and 1/3 for testing, and
13 use a random_state of 10 for the split.
14
15 Output the accuracy for the testing data.
16
17 Then execute the script by typing this command in the terminal:
18 python3 task2.py
19 """
20
21 import pandas as pd
```

Terminal:

```
Waiting for Challenge to be configured
Configured. Good luck!
$
```

Katacoda Challenge

```
[0/0] Please share your feedback:
https://get.oreilly.com/skill

Let the skills challenge begin.

Complete the following tasks
🤖 Task 1: Open task1.py and f
the file.
```

<https://learning.oreilly.com/scenarios/data-science-math/9781098108601/>

Multivariable Logistic Regression

We can easily extend logistic regression to handle multiple independent variables, simply by adding more β_x variables for each additional variable.

We then solve for those β_x variables the same way as before.

$$y = \frac{1.0}{1.0 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots + \beta_n x_n)}}$$

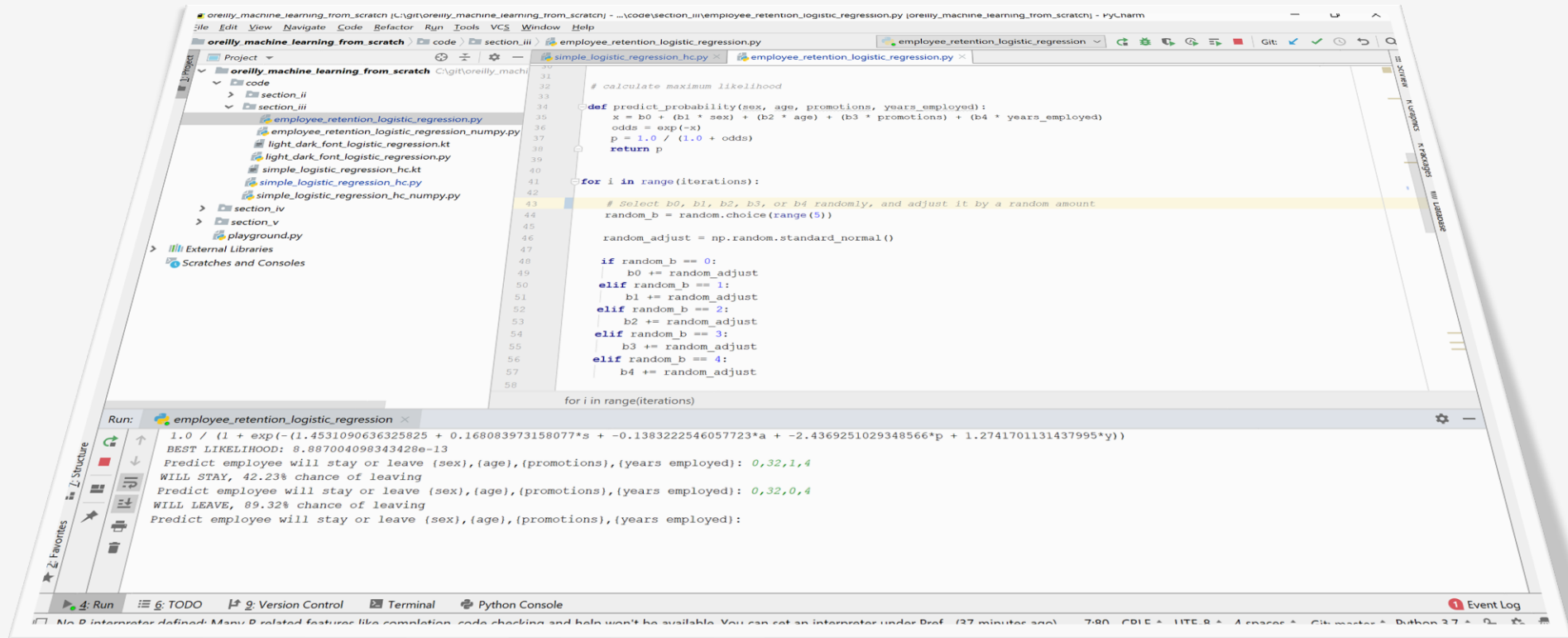
Multivariable Logistic Regression

We have some historical employee data (<https://tinyurl.com/y6r7qjrp>) and want to use it to predict whether an employee will quit or not.

SEX, *AGE*, *PROMOTIONS*, and *YEARS_EMPLOYED* are the predictor variables, and *DID_QUIT* is the outcome variable where 1 = true and 0 = false.

SEX	AGE	PROMOTIONS	YEARS_EMPLOYED	DID_QUIT
1	43	4	10	0
1	38	3	8	0
1	44	4	11	1
0	41	2	6	0
1	45	2	6	1
0	36	3	9	0
1	33	1	3	0
1	44	3	10	0
...				

Hands-On: Multivariable Logistic Regression



Using Logistic Regression for Classification

Logistic regression may seem limited in that it only supports two categories: true (1) or false (0).

But you can make it support any number of categories!

To use logistic regression for more than two categories:

1. Build a separate logistic regression for each category, where a given item belongs to that category (1) or doesn't belong to that category (0).
2. To predict which category an item belongs to, pass it through each category's logistic regression, and choose the one with the highest probability.

Quiz Time!

Logistic regression will only output a probability between 0 and 1 that an event will happen.

- 1) True
- 2) False

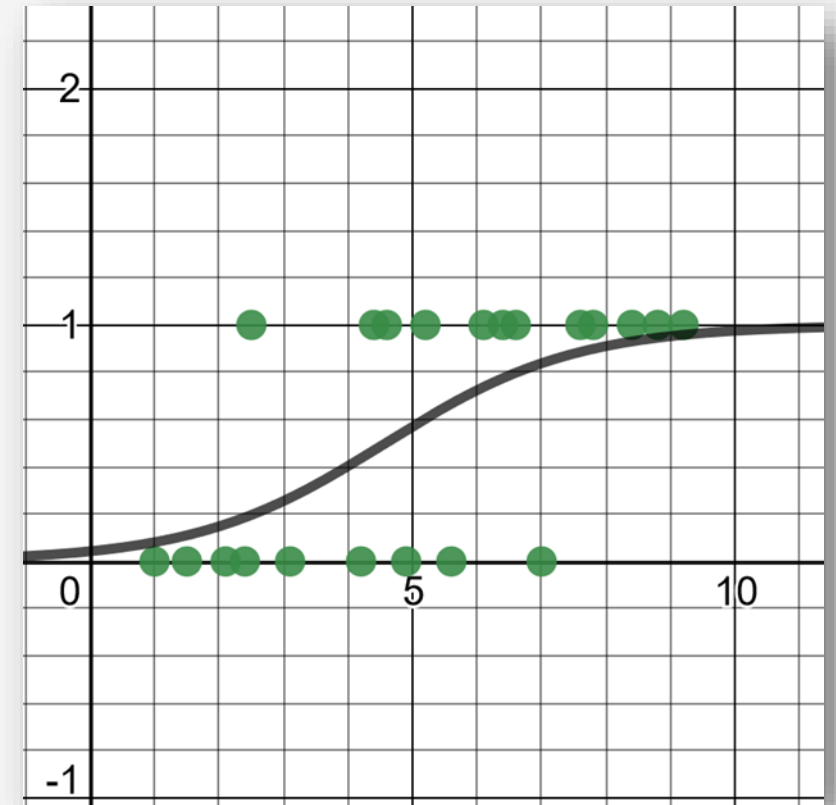
Quiz Time!

Logistic regression will only output a probability between 0 and 1 that an event will happen.

1) True

Like any probability model, logistic regression will output a percentage between 0 and 1

2) False



Quiz Time!

Logistic regression cannot be used for more than two categories.

- A) True
- B) False

Quiz Time!

Logistic regression cannot be used for more than two categories.

A) True

Logistic regression can support more than one category by doing a separate logistic regression for each category, and predicting the one that yields the highest probability.

☒ B) False

Section II

Space Shuttle Challenger

Exercise: "There's No Correlation"

It is one day from a manned space launch.

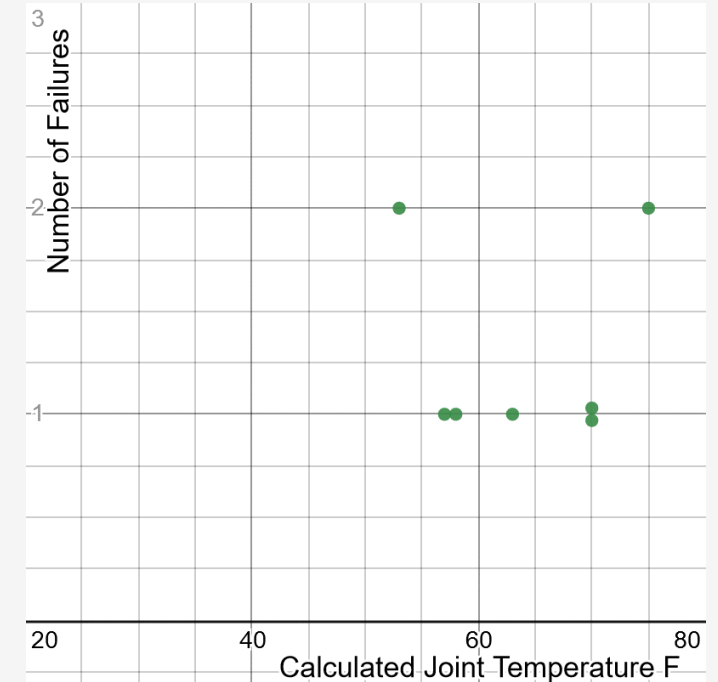
However, your engineering team has been expressing concern about the O-Rings that seal rocket gases from releasing, and whether they perform in colder temperatures.

You share this concern with other parties and are provided data of all 7 O-Ring failures from 24 launches and the temperature (shown to the right).

The consensus from other parties is there is no correlation between number of failures and temperature.

Is this assessment correct?

temperature	o_ring_failures
53	2
57	1
58	1
63	1
70	1
70	1
75	2



Exercise: "There's No Correlation"

It is one day from a manned space launch.

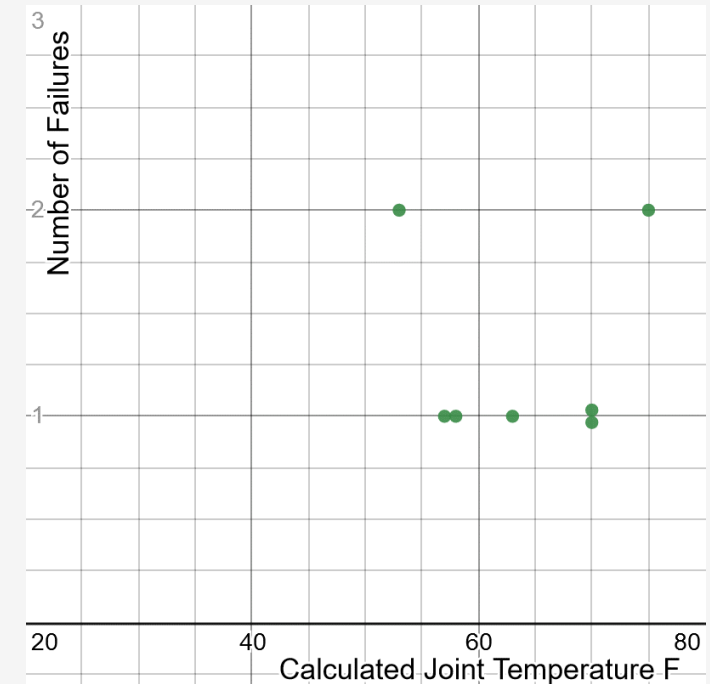
However, your engineering team has been expressing concern about the O-Rings that seal rocket gases from releasing, and whether they perform in colder temperatures.

You share this concern with other parties and are provided data of all **7 O-Ring failures** from **24 launches** and the temperature (shown to the right).

The consensus from other parties is there is no correlation between number of failures and temperature.

Is this assessment correct? **Is anything missing?**

temperature	o_ring_failures
53	2
57	1
58	1
63	1
70	1
70	1
75	2

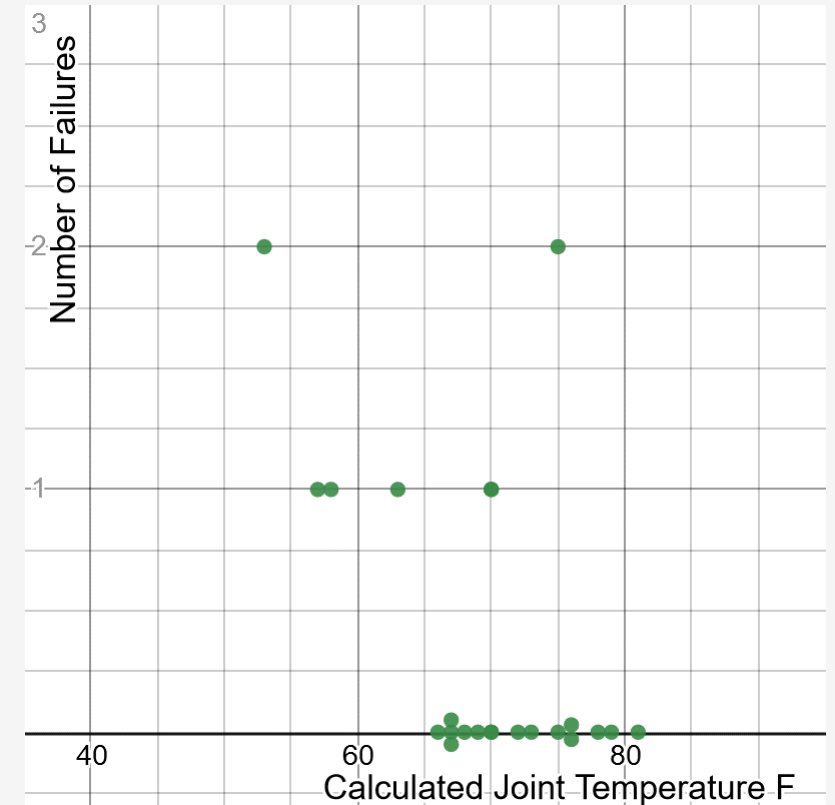


Exercise: "There's No Correlation"

Notice that data from successful launches were not included, which may tell an entirely different story.

What are your thoughts now? Is there a correlation? Is there a model that can be used to predict risk?

temperature	o_ring_failures
53	2
57	1
58	1
63	1
66	0
67	0
67	0
67	0
68	0
69	0
70	1
70	0
70	1
70	0
72	0
73	0
75	0
75	2
76	0
76	0
78	0
79	0
81	0

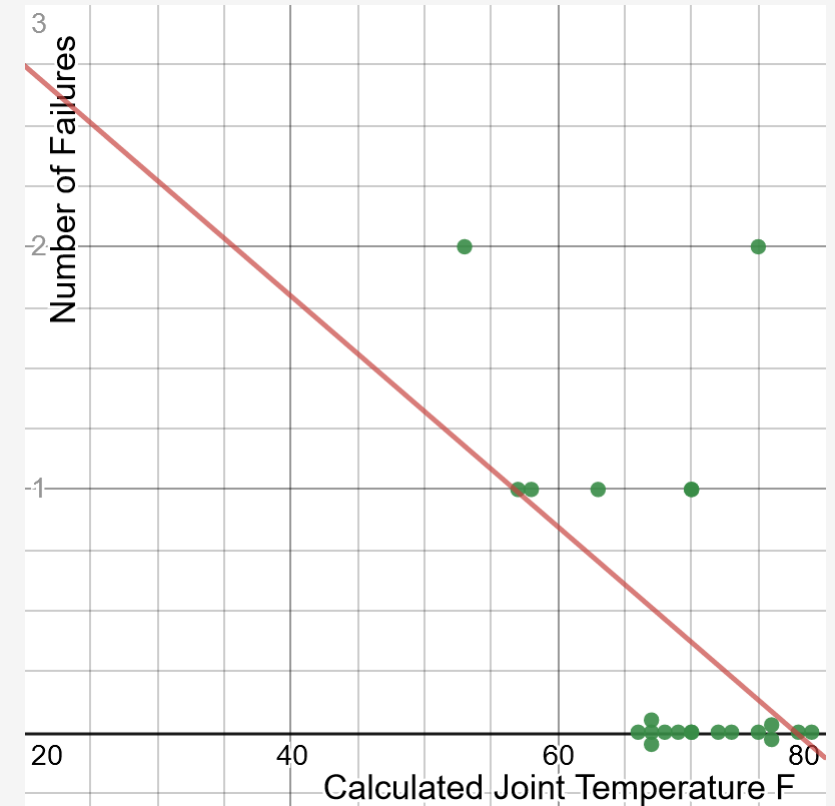


Exercise: "There's No Correlation"

You can try to apply a linear regression here, and while it does show a trend it is a little awkward especially since our data is sparse.

Should we transform our data somehow?
Are there any other models we can try?

temperature	o_ring_failures
53	2
57	1
58	1
63	1
66	0
67	0
67	0
67	0
68	0
69	0
70	1
70	0
70	1
70	0
72	0
73	0
75	0
75	2
76	0
76	0
78	0
79	0
81	0



Linear Regression Source Code

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Learn more: https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LinearRegression.html

# Import points

df = pd.read_csv('https://bit.ly/2DgjTk5', delimiter=",")

# Extract input variables (all rows, all columns but last column)
X = df.values[:, :-1]

# Extract output column (all rows, last column)
Y = df.values[:, -1]

# Plain ordinary Least squares
fit = LinearRegression().fit(X, Y)

# Print "m" and "b" coefficients
print("m = {}".format(fit.coef_.flatten()))
print("b = {}".format(fit.intercept_.flatten()))
```

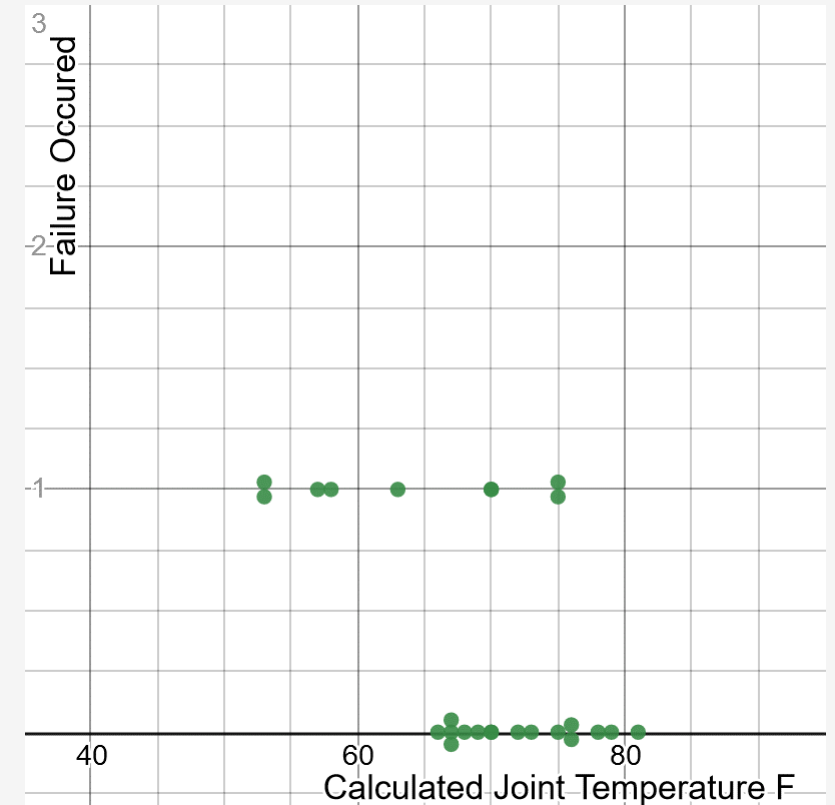
Exercise: "There's No Correlation"

What if we converted the data to be binary, showing whether a failure occurred or not occurred, by separating each instance into its own record?

This reduces the domain of output variables to "0" and "1" creating a binary model.

Is a story now becoming clear? What model can we use to predict probability of failure at a given temperature?

temperature	o_ring_failures
53	1
53	1
57	1
58	1
63	1
66	0
67	0
67	0
67	0
68	0
69	0
70	1
70	0
70	1
70	0
72	0
73	0
75	0
75	1
75	1
76	0
76	0
78	0
79	0
81	0



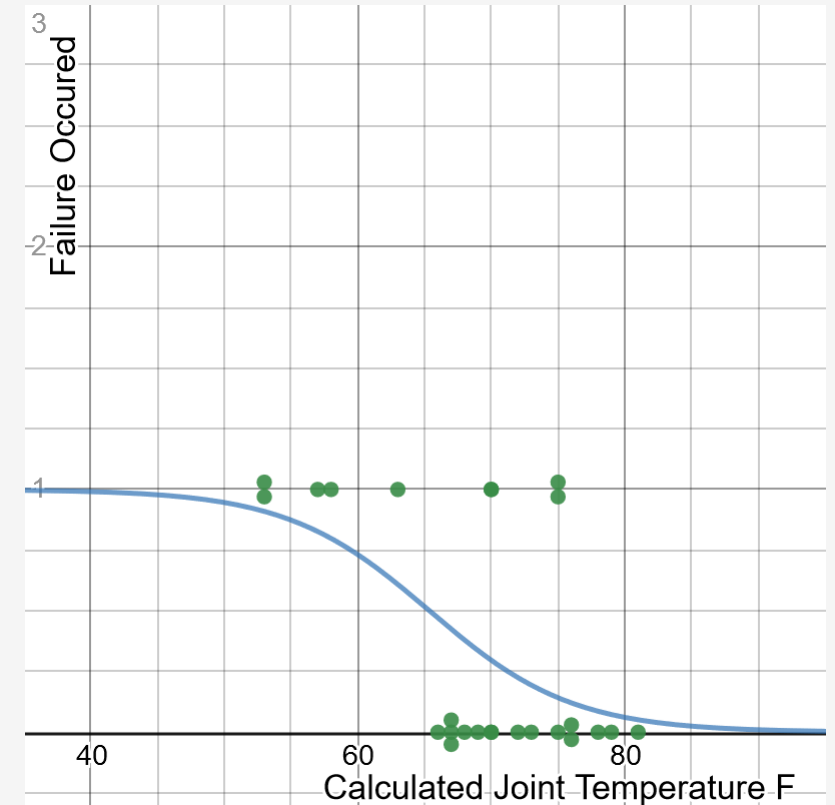
Exercise: "There's No Correlation"

Logistic regression might be the best way to model this risk.

Even though we lack freezing temperature data, the logistic regression points to a high probability of risk for O-ring failure.

If our launch is going to happen in freezing temperatures, this does not bode well.

temperature	o_ring_failures
53	1
53	1
57	1
58	1
63	1
66	0
67	0
67	0
67	0
68	0
69	0
70	1
70	0
70	1
70	0
72	0
73	0
75	0
75	1
75	1
76	0
76	0
78	0
79	0
81	0



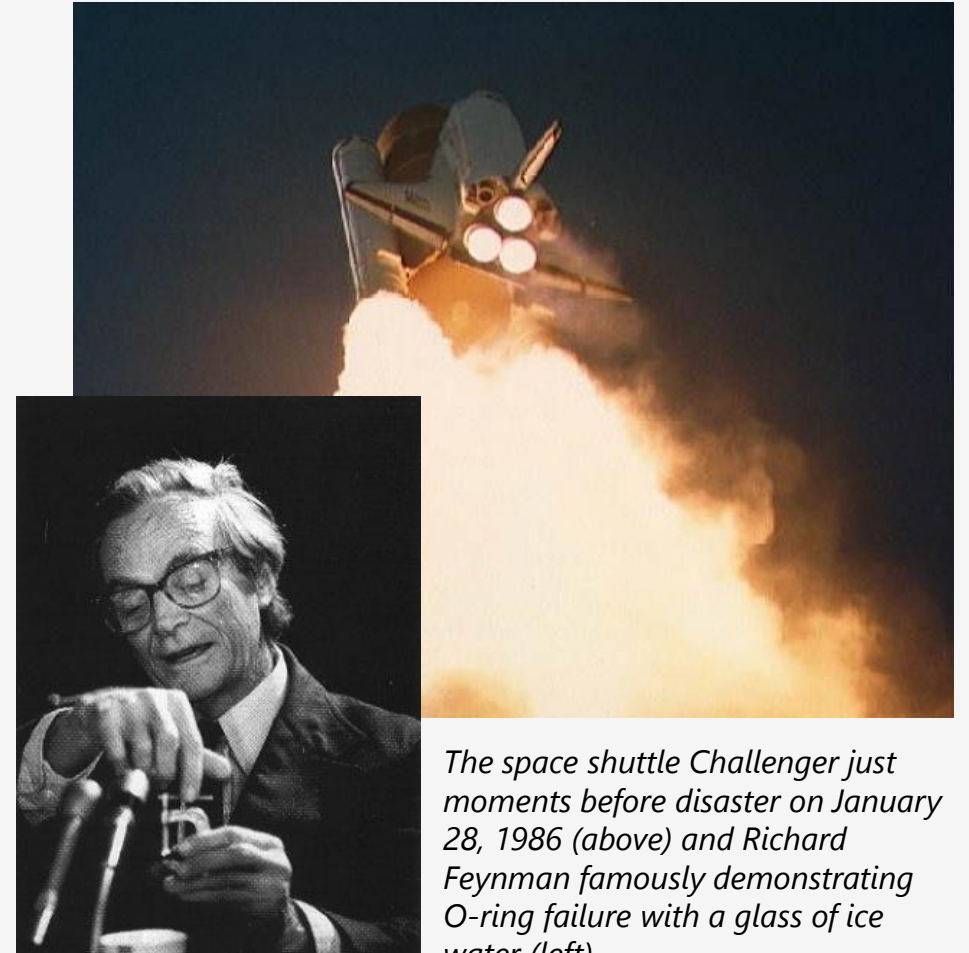
Exercise: "There's No Correlation"

This is exactly what happened to the space shuttle Challenger on January 28, 1986, and if you already have not figured out already, we are doing the analysis.

Through a series of unfortunate events, only partial data was accessible and omitted non-failure data, which showed a correlation with temperature and O-ring failure.

The analysis we just did should have happened before the accident, but unfortunately it occurred afterwards.

We will learn later about how biases, class imbalance, and broken data can derail machine learning and statistical models.



The space shuttle Challenger just moments before disaster on January 28, 1986 (above) and Richard Feynman famously demonstrating O-ring failure with a glass of ice water (left).

Space Shuttle Challenger: Logistic Regression Source Code

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go
from sklearn.linear_model import LogisticRegression

# Load the data
df = pd.read_csv("https://bit.ly/2DgjTk5", delimiter=",")

# Duplicate and convert records to have binary outcomes
df = df.loc[df.index.repeat(df["o_ring_failures"])]
df.loc[(df["o_ring_failures"] > 0), 'o_ring_failures'] = 1

# Extract input variables (all rows, all columns but last column)
X = df.values[:, :-1]

# Extract output column (all rows, last column)\
Y = df.values[:, -1]

model = LogisticRegression()
model.fit(X, Y)

# Print "m" and "b" coefficients
print("m = {}".format(model.coef_.flatten()))
print("b = {}".format(model.intercept_.flatten()))

# Plot results
fig = go.Figure()
fig.add_trace(go.Scatter(x=df["temperature"], y = df["o_ring_failures"], mode='markers', name="Observations"))
fig.add_trace(go.Scatter(x=np.arange(0.0,90.0,.1),
                        y= model.predict_proba(np.arange(0.0,90.0,.1).reshape(-1, 1))[:, -1],
                        mode='lines',
                        name='Logistic Regression'))

fig.show()
```

Section III

Preparing and Rescaling Data

Normalization

Normalization is scaling and converting the values of each variable, so they are relatively close together.

- Imagine you had a variable **age** whose values typically range in 0-99.
- But you had another variable **income** that typically ranges from 30,000 to 1,000,000.
- Because these ranges are so drastically different, fitting a model is not going to be productive until you transform them somehow.

There are a variety of techniques you can employ from linear scaling to fitting to a standard normal distribution.



Rescaling

When you have different fields with varying scales, it can be helpful to **rescale, or normalize, the data by compressing it between 0.0 and 1.0.**

This can be helpful for optimization algorithms that perform training, making it easier to iterate parameters to fit the data more productively.

Note you do not always have to rescale, but sometimes it can give a better result especially for data with fields that vary in scale.

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv('https://bit.ly/33iTfS9', delimiter=",")

# Extract input variables (all rows, all columns but last column)
X = df.values[:, :-1]

# Extract output column (all rows, 5th column)
Y = df.values[:, 4]

# Rescale all the input variables to be between 0 and 1
scaler = MinMaxScaler(feature_range=(0.0, 1.0))
rescaled_X = scaler.fit_transform(X)

print(rescaled_X)
```

Doing a simple rescale between 0.0 and 1.0 above in scikit-learn

[[0	25	2	3]		[0.	0.	0.5	0.2]	
	[0	30	2	3]		[0.	0.20833333	0.5	0.2]
	[0	26	2	3]		[0.	0.04166667	0.5	0.2]
	[0	25	1	2]		[0.	0.	0.25	0.1]
	[0	28	1	2]		[0.	0.125	0.25	0.1]
	[0	30	2	4]		[0.	0.20833333	0.5	0.3]
	[0	49	4	8]		[0.	1.	1.	0.7]

Standardization

Standardization is transforming numeric variables to fit a standard normal distribution (with a mean of 0 and standard deviation of 1) and express each value in standard deviations.

It can be preferable to rescaling, if the data can assume a normal distribution for each variable.

In other words, standardization is finding the z-scores for each variable.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('https://bit.ly/3flZJSR', delimiter=";")

# Extract input variables (all rows, all columns but last column)
X = df.values[:, :-1]

# Extract output column (all rows, last column)
Y = df.values[:, -1]

# Rescale all the input variables to be between 0 and 1
scaler = StandardScaler().fit(X)
rescaled_X = scaler.fit_transform(X)

print(rescaled_X)
```

Doing a standardization on all data fields

[6.	148.	72.	...	33.6	0.627	50.]
[1.	85.	66.	...	26.6	0.351	31.]
[8.	183.	64.	...	23.3	0.672	32.]



[0.63994726	0.84832379	0.14964075	...	0.20401277	0.46849198	1.4259954]
[-0.84488505	-1.12339636	-0.16054575	...	-0.68442195	-0.36506078	-0.19067191]
[1.23388019	1.94372388	-0.26394125	...	-1.10325546	0.60439732	-0.10558415]

Unit Vector Normalization

Not to be confused with standardization, unit vector normalization is rescaling each data record to have a vector length of 1.0.

This is probably the most complicated transformation, as it involves some understanding of linear algebra.

It works well for sparse data sets and it compresses the data into a smaller space, which is especially helpful for high-dimension datasets.

It is often used for neural networks, k-nearest neighbors, and a few other ML algorithms.

```
import pandas as pd
from sklearn.preprocessing import Normalizer

df = pd.read_csv('https://bit.ly/3flZJSR', delimiter=",")

# Extract input variables (all rows, all columns but last column)
X = df.values[:, :-1]

# Extract output column (all rows, last column)
Y = df.values[:, -1]

# Rescale all the input variables to be normalized
scaler = Normalizer().fit(X)
rescaled_X = scaler.fit_transform(X)

print(rescaled_X)
```

Doing a normalization on all data fields

[6.	148.	72.	...	33.6	0.627	50.]
[1.	85.	66.	...	26.6	0.351	31.]
[8.	183.	64.	...	23.3	0.672	32.]



[0.03355237	0.82762513	0.40262844	...	0.18789327	0.00350622	0.27960308]
[0.008424	0.71604034	0.55598426	...	0.22407851	0.00295683	0.26114412]
[0.04039768	0.92409698	0.32318146	...	0.11765825	0.00339341	0.16159073]

Section IV

Classification Validation

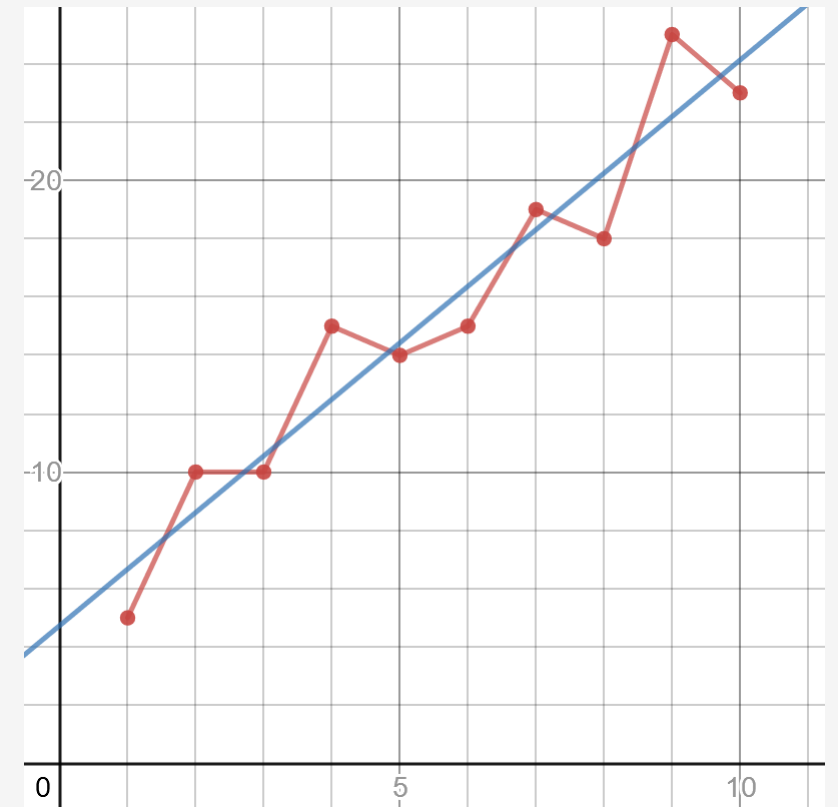
A Note About Overfitting

We are not going to focus too much on validating and analyzing machine learning models, but we should at least mention overfitting.

Overfitting means that our ML model works well with the data it was trained on but fails to predict correctly with new data.

- This can be due to many factors, but a common cause is the sampled data does not represent the larger population and more data is needed.
- The red line has high **variance**, meaning its predictions are sensitive to outliers and therefore can vary greatly.
- The blue line has high **bias**, meaning the model is less sensitive to outliers because it prioritizes a method (maintaining a straight line) rather than bend and respond to variance.

The red line to the right is likely overfitted (high variance, low bias), but the blue linear regression line (low variance, high bias) is less likely to be overfit.



<https://www.desmos.com/calculator/wmwfolbvdk>

ML Classification is Not Perfect

Machine learning algorithms are not perfect and easily error-prone especially with outliers.

Generally, the more “advanced” and flexible a machine learning method is the more prone it is to overfitting.

To the right are examples of a well-trained neural network unable to recognize images correctly due to objects in abnormal positions

SOURCE: <https://arxiv.org/abs/1811.11553>



parachute 0.54



snowplow 0.92



bobsled 0.79

A Note About Overfitting

Linear and logistic regression are highly biased methods and are resilient to overfitting.

There are other remedies to mitigate overfitting, the most basic being separating **training data and **test data**.**

- The model is fit to the training data, and then is tested with the test data.
- If the test data performs poorly compared to the training data, there is a possibility of overfit (or just no correlation altogether).

You can also try to train with more data as well as utilize cross-validation, regularization, bagging, boosting, and other techniques.

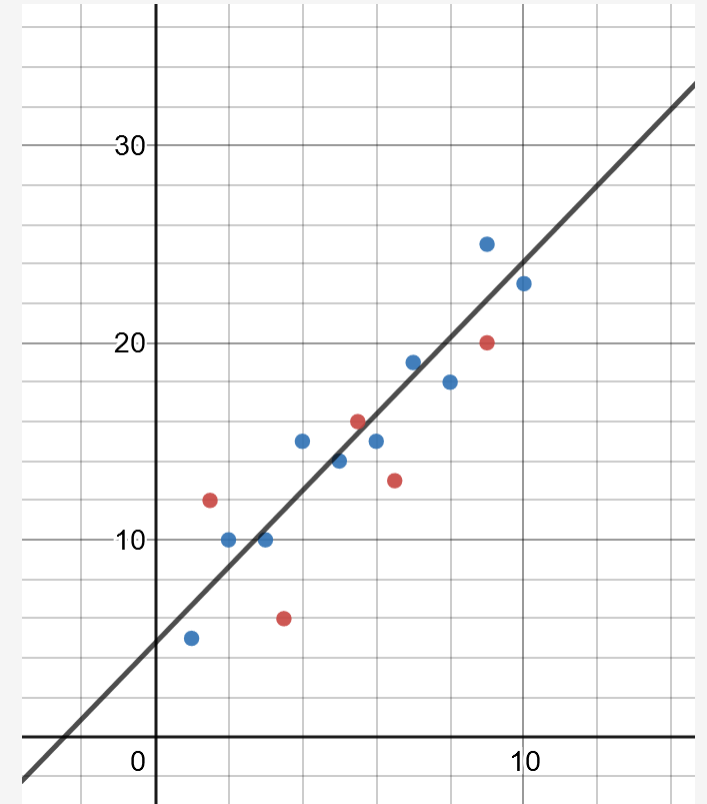
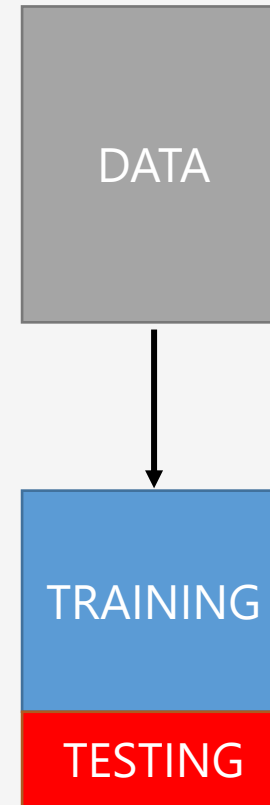


Training and Testing Data

A common practice to proactively prevent overfitting in machine learning is to separate training data and testing data.

- **Training data** is data used to fit a model and is typically 2/3 of the data.
- **Test data** is used to test the model and is the remaining 1/3 of the data.

By omitting the testing data from training, we see how well the model works on data it has not seen before and change our parameters accordingly.



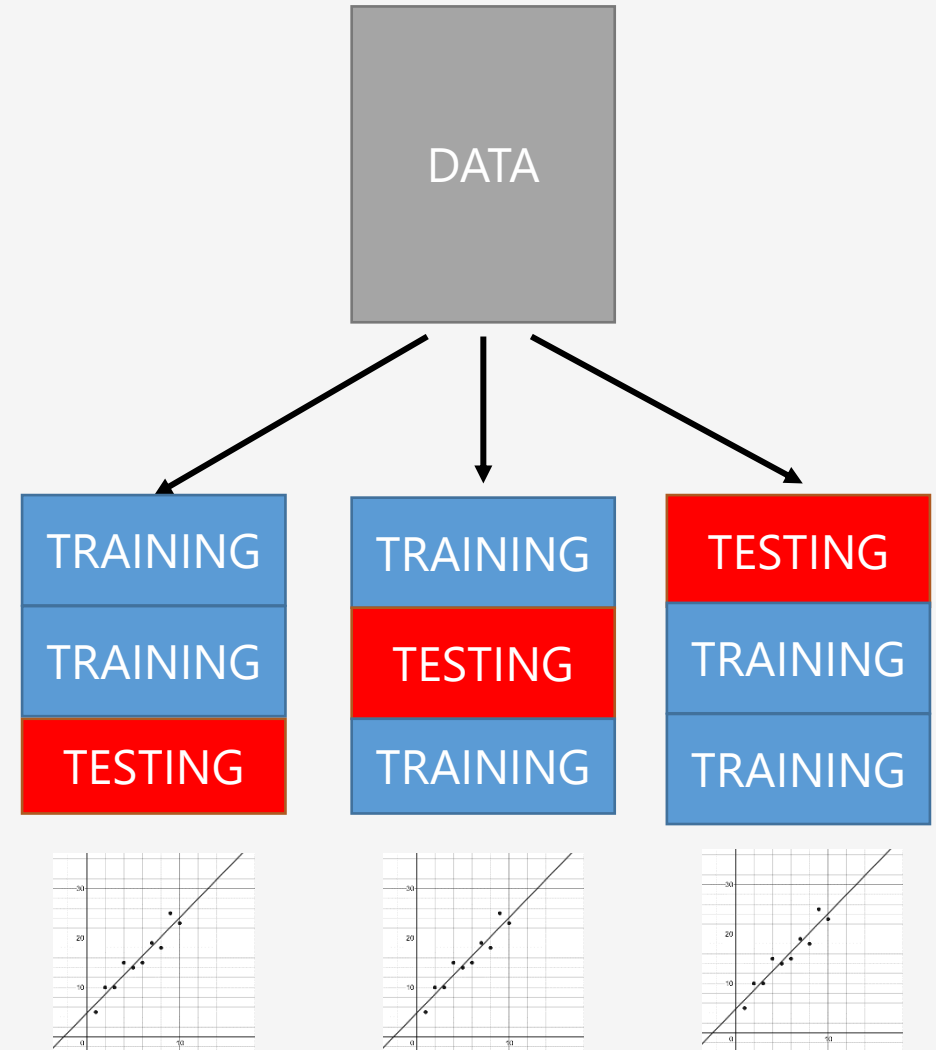
Cross-Validation

We can take this concept of training/testing data a step further, and test different combinations of training and testing data.

This is known as **cross-validation**, the gold standard of validation techniques.

To the right we have **3-fold cross validation** which breaks the data into thirds and uses one of the pieces for testing.

We can then evaluate how well each of these perform, being able to compare different parameters and models (e.g. linear regression vs decision trees) and see which setup produces the best performance.



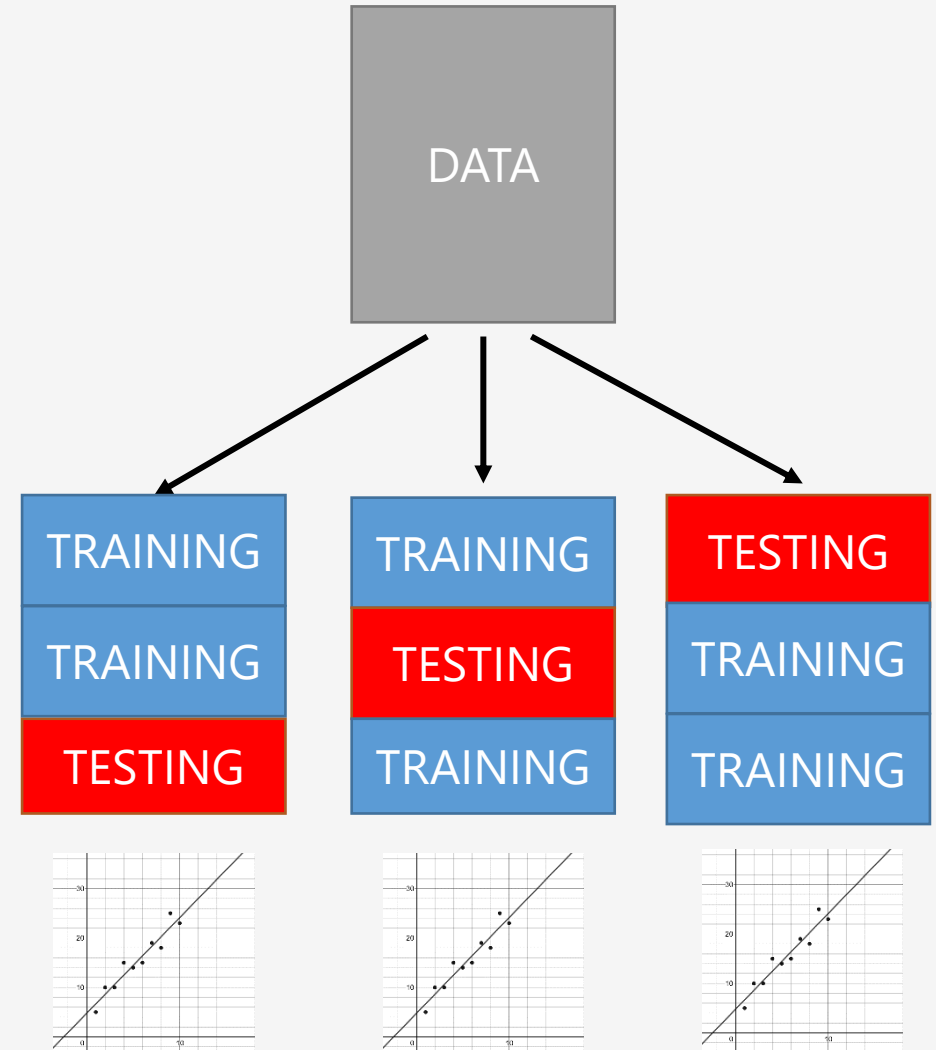
Cross-Validation

We can take this concept of training/testing data a step further, and test different combinations of training and testing data.

This is known as **cross-validation**, the gold standard of validation techniques.

To the right we have **3-fold cross validation** which breaks the data into thirds and uses one of the pieces for testing.

We can then evaluate how well each of these perform, being able to compare different parameters and models (e.g. linear regression vs decision trees) and see which setup produces the best performance.



Cross-Validation

Note that **k-fold cross validation** allows us to slice our data into any number and not just 3 (typically 3, 5, or 10).

For example we can do 10-fold cross validation and validate 10 different combinations of training/test data.

The most extreme form of folding is **leave-one-out cross validation**, which omits one data record for testing and uses the remaining records for training, and this is done repeatedly.

TRAINING

TRAINING

TRAINING

TRAINING

TRAINING

TRAINING

TRAINING

TRAINING

TRAINING

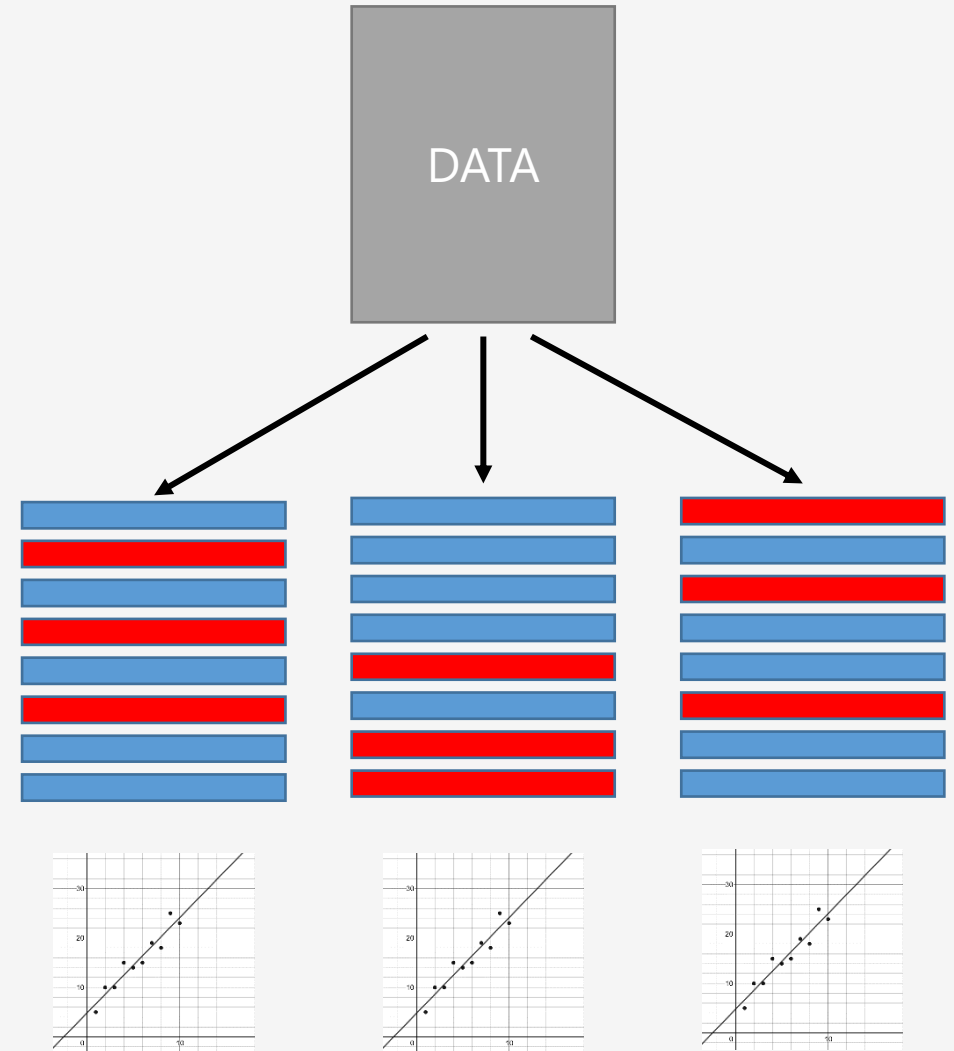
TESTING

Random Fold Validation

As you may be noticing, machine learning often tries to overcome data variance with randomness.

A variant with fold validation is **repeated random fold validation**, where we randomly shuffle the data and create random train/test folds as many times as desired.

This is helpful when we need to mitigate variance in the model.



Which Validation to Use?

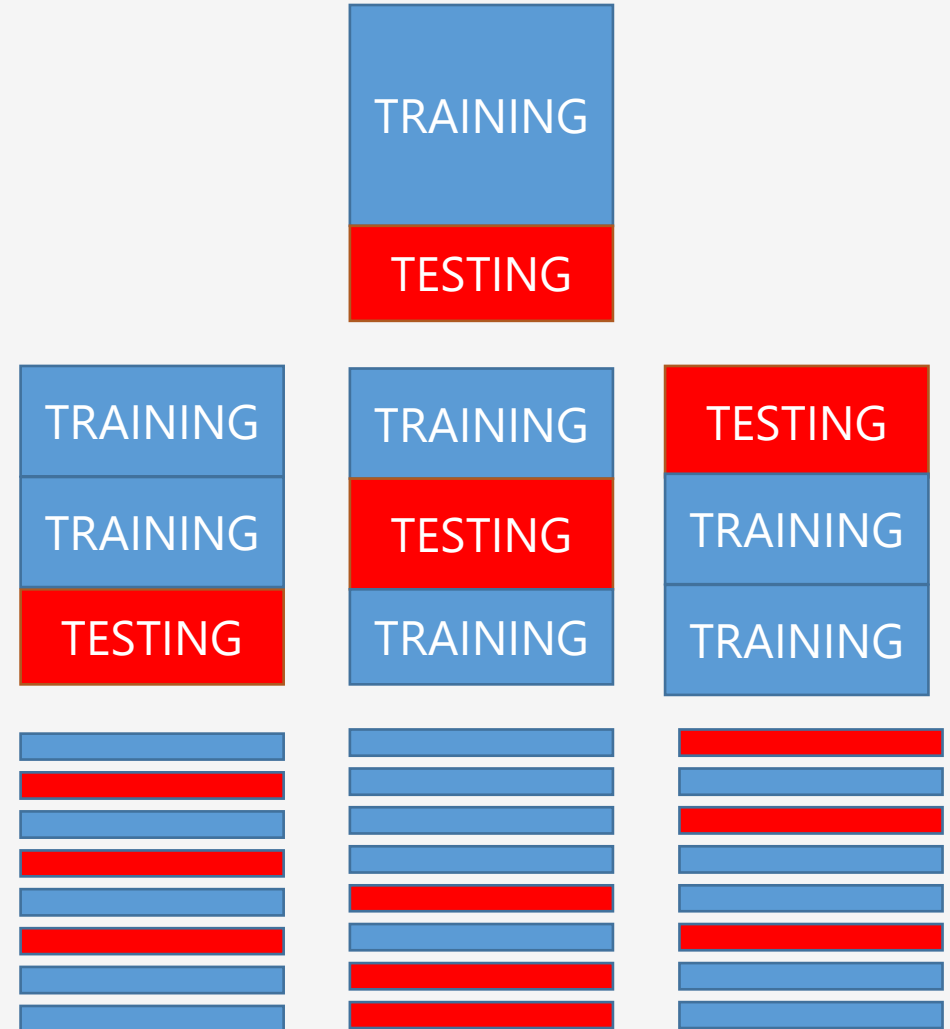
Generally, you will want to prefer k-fold validation as it is the gold standard.

A single train/test split might be warranted if performance of machine learning algorithm is slow and has enough data with lower bias.

Use the random fold split to mitigate variance in the model while balancing training speed and dataset sizes.

We will talk about class imbalance later, but if you do not have an equal number of samples for each class, you might want to consider using stratification in your k-fold validation.

Stratification means an equal proportion of data for each class is sampled for training and testing data (even it is sampled repeatedly), so that no class is neglected.



A Stratified Validation in Scikit-Learn

```
import numpy as np
import pandas as pd
# Load data
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

df = pd.read_csv("https://tinyurl.com/y6r7qjrp", delimiter=",")

# Extract input variables (all rows, all columns but last column)
# Note we should do some linear scaling here
X = (df.values[:, :-1] / 255.0) + .01

# Extract output column (all rows, last column)
Y = df.values[:, -1]

# Get a count of each group to ensure samples are equitably balanced
print(df.groupby(["class"]).agg({"class" : [np.size]}))

# Separate training and testing data
# Note that I use the 'stratify' parameter to ensure
# each class is proportionally represented in both sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.33, stratify=Y)

nn = MLPClassifier(solver='sgd', hidden_layer_sizes=(100, ),
                  max_iter=480, learning_rate_init=.1)

nn.fit(X_train, Y_train)

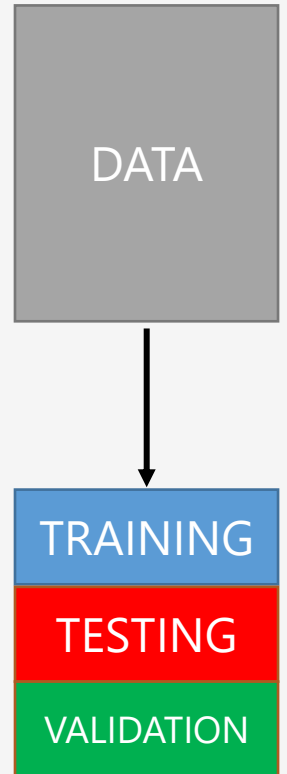
print("Training set score: %f" % nn.score(X_train, Y_train))
print("Test set score: %f" % nn.score(X_test, Y_test))
```

Validation Data

Validation is a separate type of testing data used to compare performance of different models.

When you are comparing two or more models (e.g. logistic regression versus decision trees), you may hold back one more chunk of data for validation.

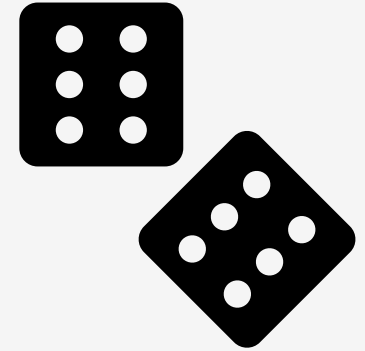
It is different than the testing data, which is used to tune parameters of the individual model, not compare different models altogether.



Random Seeding

Machine learning deals with a lot of random-based operations:

- Randomly sampled training/testing data
- Random-based optimization and training (e.g. stochastic gradient descent, simulated annealing)
- Randomly generated data from simulations



Testing can be challenging as the above operations make outputs nondeterministic, so setting a **random seed is common practice to make them deterministic.**

This allows tests to be conveniently reproducible, where sequences of randomly generated values are the same.

Be careful to not rely on random seeds too much! They can give the illusion of determinism so test many random seeds, not just one, to ensure they converge towards the same outcome.

Random Seeding a 3-Fold Split for Reproducibility

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, cross_val_score

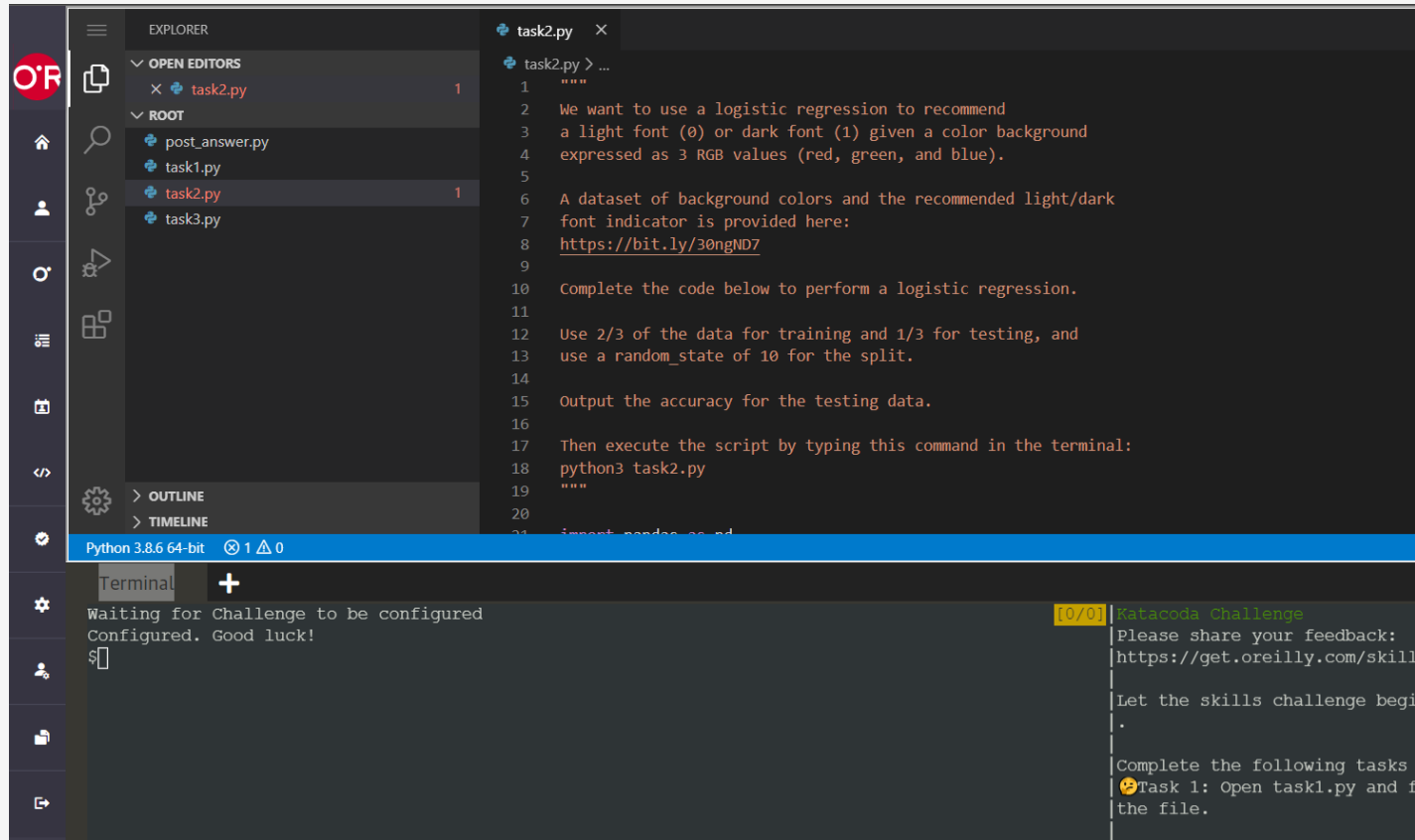
# Load the data
df = pd.read_csv("https://tinyurl.com/y6r7qjrp", delimiter=",")

X = df.values[:, :-1]
Y = df.values[:, -1]

# "random_state" is the random seed, which we fix to 7
kfold = KFold(n_splits=3, random_state=7, shuffle=True)
model = LogisticRegression(solver='liblinear')
results = cross_val_score(model, X, Y, cv=kfold)

print("Accuracy Mean: %.3f (stdev=%.3f)" % (results.mean(), results.std()))
```

Katacoda – Task 2



The screenshot displays the Katacoda IDE interface. On the left, the Explorer panel shows the file structure with 'task2.py' selected. The main editor area displays the content of 'task2.py', which includes instructions for a logistic regression task. The terminal at the bottom shows the initial setup message and a challenge notification.

```
task2.py
1 """
2 We want to use a logistic regression to recommend
3 a light font (0) or dark font (1) given a color background
4 expressed as 3 RGB values (red, green, and blue).
5
6 A dataset of background colors and the recommended light/dark
7 font indicator is provided here:
8 https://bit.ly/30ngND7
9
10 Complete the code below to perform a logistic regression.
11
12 Use 2/3 of the data for training and 1/3 for testing, and
13 use a random_state of 10 for the split.
14
15 Output the accuracy for the testing data.
16
17 Then execute the script by typing this command in the terminal:
18 python3 task2.py
19 """
20
21 import pandas as pd
```

Terminal

```
Waiting for Challenge to be configured
Configured. Good luck!
$
```

[0/0] Katacoda Challenge

```
Please share your feedback:
https://get.oreilly.com/skill

Let the skills challenge begin.

Complete the following tasks
🤖 Task 1: Open task1.py and f
the file.
```

<https://learning.oreilly.com/scenarios/data-science-math/9781098108601/>

Why "Accuracy" is a Bad Measure for Classification

Suppose your machine learning model observed people with the name "Michael" quit their job.

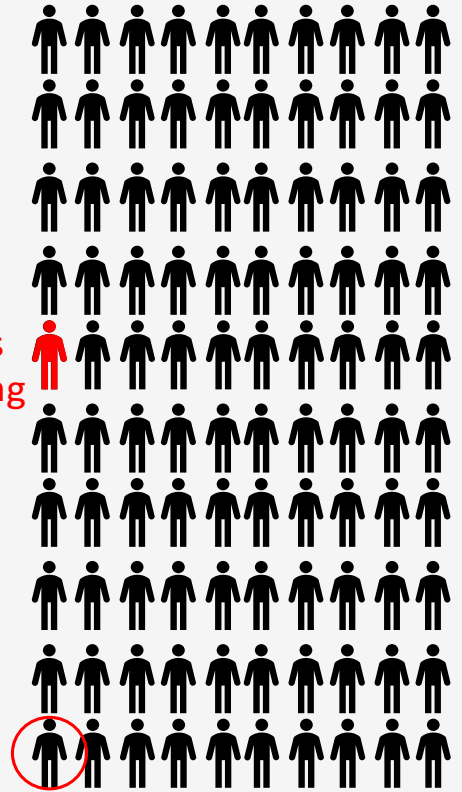
Therefore it simply predicts any employee with the name "Michael" will quit, and everyone else will stay.

If there was one Michael in a company of 100 employees, but a different employee ended up quitting, the machine could still boast 98% accuracy.

Why stop there? If 99% of employees do not quit, you might as well predict every employee *will not quit* and get 99% accuracy.

Machine learning makes shortcuts like this all the time especially when the event of interest is rare (e.g. diseases, security breaches, employee attrition, rare road/highway events).

This employee quits
Prediction was wrong
But I'm still 98% accurate!



Prediction
This employee is named "Michael"
This employee will quit

Confusion Matrix

As seen in the previous example, **accuracy** (the rate of correct labeling) can be a misleading measure of performance, especially in classification problems with class imbalance.

A better way to evaluate classification models is the **confusion matrix**, which keeps track of **false positives** (Type I Error) and **false negatives** (Type II Error).

We want to evaluate how many employees who were predicted to quit actually did quit (**true positives**).

Conversely, we also want to evaluate how many employees who were predicted to stay actually did stay (**true negatives**).

	Actually Quits (True)	Actually Stays (False)
Predicted will quit (True)	0	1
Predicted will stay (False)	1	98

Confusion Matrix

From the confusion matrix, we can derive all sorts of useful metrics beyond just accuracy.

We can easily see that **precision** (how accurate positive predictions were) and **sensitivity** (rate of identified positives) are 0, meaning this machine learning model fails entirely at positive predictions.

	Actually Quits	Actually Stays	
Predicted will quit	0 (TP)	1 (FN)	Sensitivity $\frac{TP}{TP+FN} = \frac{0}{0+1} = 0$
Predicted will stay	1 (FP)	98 (TN)	Specificity $\frac{TN}{TN+FP} = \frac{98}{98+1} = .989$
	Precision $\frac{TP}{TP+FP} = \frac{0}{0+1} = 0$	Negative Predicted Value $\frac{TN}{TN+FN} = \frac{98}{98+1} = .989$	Accuracy $\frac{TP+TN}{TP+TN+FP+FN} = \frac{98+0}{0+98+1+1} = .98$

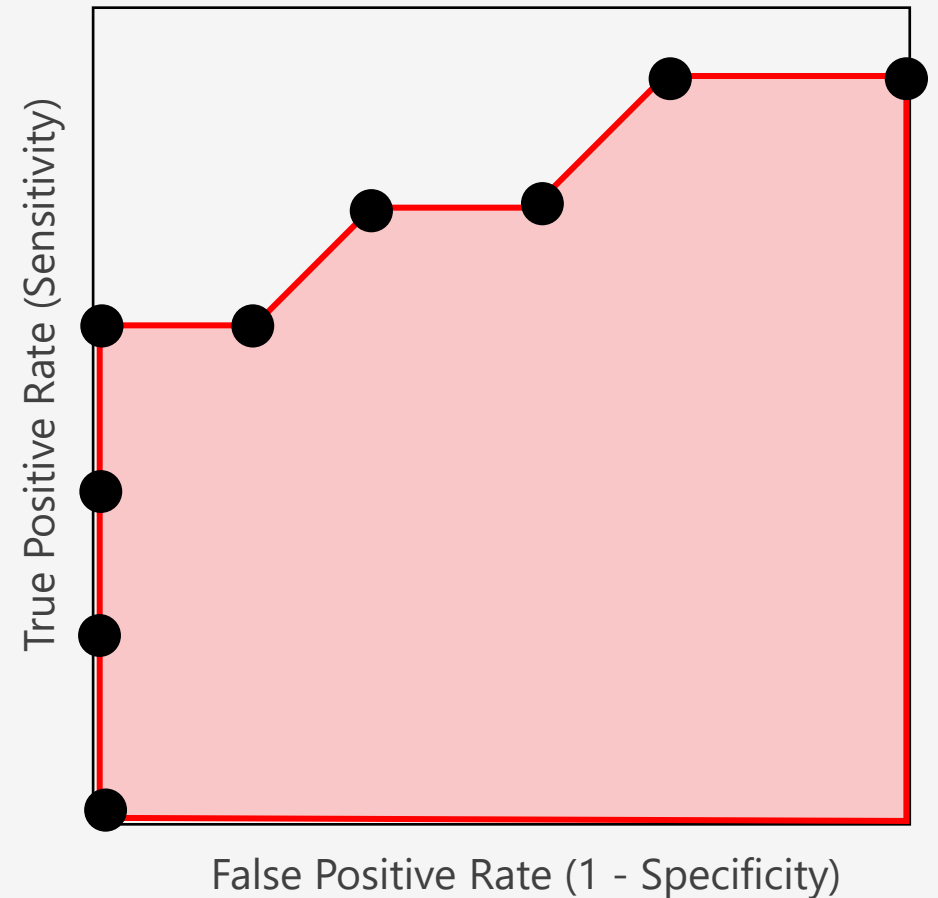
F1 Score $\frac{2 * Precision * Recall}{Precision + Recall} = \text{Undefined}$

ROC and AUC

When we are evaluating different machine learning configurations, we may end up with dozens, hundreds, or thousands of confusion matrices.

These can be tedious to review, so we can summarize all of them with a **receiver operator characteristic (ROC)** curve as shown to the right.

This allows us to see each testing instance (each represented by a black dot) and find an agreeable balance between true positives and false positives.

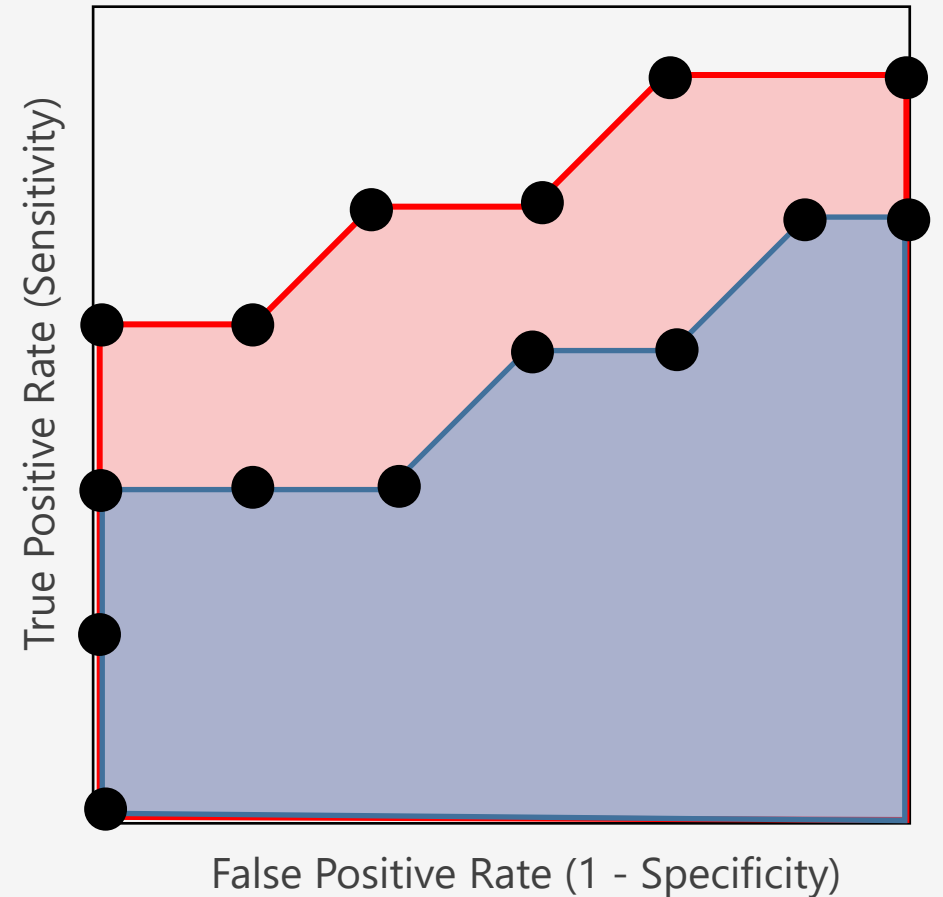


ROC and AUC

We can also compare different machine learning models by creating separate ROC curves for each.

For example, if our red curve represents a logistic regression and the blue curve represents a decision tree, we can see the performance of them side-by-side.

The **area under the curve (AUC)** is a good metric for choosing which model to use. Since the red curve (logistic regression) has a greater area, this suggests it is a superior model.



Violence and Video Games

Let's say a machine learning dataset claims 85% of homicidal criminals in the United States have played violent video games.

What does this mean? Should we be alarmed and blame the video game industry?

Keep in mind that a conditional probability $P(A|B)$ is not the same as its reverse conditional probability $P(B|A)$.

$$P(\text{gamer}|\text{homicidal}) = .85$$

$$P(\text{homicidal}|\text{gamer}) = ?$$

Just because most homicidal people are gamers (according to this study), does that mean most gamers are homicidal?

The answer we will discover is "NO!"



[This Photo](#) is licensed under [CC BY](#)

Violence and Video Games

We need to use Bayes Theorem to flip the condition:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

$$P(homicidal|gamer) = \frac{P(gamer|homicidal) * P(homicidal)}{P(gamer)}$$

But we only have $P(gamer|homicidal)$ which is .85.

We also need $P(homicidal)$ and $P(gamer)$ if we are going to flip the direction of the probability.



[This Photo](#) is licensed under [CC BY](#)

Violence and Video Games

Sometimes it can be hard to get these other probabilities, but with population studies a little bit of research might give us some numbers.

According to the FBI, there are 17,251 known homicidal offenders in 2017 and according to Wolfram Alpha there are 324 million people in the United States.

$$P(homicial) = \frac{17251}{324000000} = .00005$$

Gathering data from video game industry market research, you estimate that 19% of the population plays violent video games.

$$P(gamer) = .19$$



[This Photo](#) is licensed under [CC BY](#)

Discovering Bayes Theorem

Unlike the media and politicians, we continued doing the math and research and found 19% of the population plays violent video games, and .005% of the population is homicidal.

We can use Bayes Theorem to merge these probabilities together.

$$P(\text{Homicidal if Gamer}) = P(\text{Gamer if Homicidal}) * P(\text{Homicidal}) / P(\text{Gamer})$$

$$P(\text{Homicidal if Gamer}) = (.85 * .00005) / .19$$

$$P(\text{Homicidal if Gamer}) = 0.0002$$

Wow! So if you take any gamer there is only a .02% chance they are homicidal.

This number is much different than the 85% probability a homicidal criminal is a gamer.



[This Photo](#) is licensed under [CC BY](#)

Bayes and the Confusion Matrix

A medical technology vendor approaches your organization and says they have an “AI system” that takes a small blood sample, assays several variables, ensembles machine learning algorithms, and predicts a health risk (simply categorized as “AT RISK” or “NOT AT RISK”).

The vendor says that 99% of patients who are at risk will test positive.

Is this significant and credible? Why or why not? What should our next questions be?



Bayes and the Confusion Matrix

There is still some missing information you will have to solicit from the vendor.

We are told that for patients that have health risk, 99% will be identified successfully (*sensitivity*).

But what if we flip the question? What percentage of those who tested positive have the health risk (*precision*)?

The vendor goes back to their research team and returns with a confusion matrix of 1000 patients they tested.

They give you an answer: $198 / (198 + 50) = 79.8\%$



	TESTS POSITIVE	TESTS NEGATIVE
AT RISK	198	2
NOT AT RISK	50	750

Bayes and the Confusion Matrix

With this confusion matrix, we indeed see 99% of those who are at risk test positive, and that 79.8% of those tested positive are at risk. We found the conditional probability in both directions.

Despite these numbers, you start to feel uncomfortable that you are not using outside data beyond the vendor's test.

What else can we do to verify the vendor's claims and testing results?



	TESTS POSITIVE	TESTS NEGATIVE
AT RISK	198	2
NOT AT RISK	50	750

Bayes and the Confusion Matrix

After some Wikipedia research, you discover something pivotal...

Only 1% of the population is at risk.

This means if you planned on deploying this test to 100K patients, only 1000 are likely at risk.

The vendor advertised a 99% true positive rate. You now wonder: will the vendor's product single out these 1000 patients? Out of 100K? Without error?



	TESTS POSITIVE	TESTS NEGATIVE
AT RISK	198	2
NOT AT RISK	50	750

Bayes and the Confusion Matrix

If you feel like something is off because only 1% of the population has the health risk, but 20% of the test patients do, you would be right.

Utilizing Bayes Theorem, we can incorporate this 1% statistic to find the true probability of being at risk if tested positive.

$$P(\text{At Risk if Positive}) = P(\text{Positive if at risk}) * P(\text{At risk}) / P(\text{Positive})$$

$$P(\text{At Risk if Positive}) = (.99 * .01) / .248$$

$$P(\text{At Risk if Positive}) = .0339$$

3.39% is now the probability of risk given a positive test. How did it drop from 99% to 3.39%?



	TESTS POSITIVE	TESTS NEGATIVE
AT RISK	198	2
NOT AT RISK	50	750

Bayes and the Confusion Matrix

3.39% is now the probability of risk given a positive test. How did it drop from 99% to 3.39%?

This just shows how easily we can get duped by probabilities that are only high in a specific population like the vendor's 1000 test patients (remember the homicides and video games example)?

So if this test only has a 3.39% probability of successfully identifying a true positive, we probably should not use it.



	TESTS POSITIVE	TESTS NEGATIVE
AT RISK	198	2
NOT AT RISK	50	750

Imbalanced Data

An open secret with machine learning is that imbalanced datasets are a problem, and practitioners will artificially inflate data to make it balanced.

This means that for predicting a specific event, like criminality, we need to create a balanced dataset where 50% of the data have criminal labels and 50% do not.

As we have seen predicting rare events (criminality, diseases, security breaches, etc) are imbalanced because the event of interest is rare, and techniques like duplicating that data is done to inflate it.

AGE	VIDEO GAMER	LIKES ICE CREAM	MINORITY	CRIMINAL
55	1	0	0	1
38	0	0	0	0
47	0	1	0	1
35	1	0	1	1
23	0	1	0	1
31	1	1	1	1
33	1	0	0	1
50	1	1	0	0
32	0	0	1	1
31	0	0	1	1
35	0	0	0	1
24	1	1	1	0
22	0	1	1	0
27	1	1	1	0
34	0	1	1	1
32	1	0	0	1
34	0	1	1	1
31	0	0	0	1
23	1	1	0	0
27	0	1	0	0
52	1	1	0	0
21	1	1	0	0
41	1	0	1	1
31	0	0	1	1
35	1	1	0	0

Imbalanced Data

But what is talked less about with imbalanced data is the input variables being imbalanced can cause problems too.

Take this criminal prediction dataset where the input variables have the following proportions:

50% ARE CRIMINALS

70% likely to be 30-39

30% percent likely to like ice cream

70% percent likely to be a video gamer

60% percent likely to be minority

50% ARE NOT CRIMINALS

20% percent likely to be 30-39

80% percent to likely like ice cream

30% percent likely to be a video gamer

20% percent likely to be minority

What do you think is going to happen if I train a machine learning system with this dataset?

AGE	VIDEO GAMER	LIKES ICE CREAM	MINORITY	CRIMINAL
55	1	0	0	1
38	0	0	0	0
47	0	1	0	1
35	1	0	1	1
23	0	1	0	1
31	1	1	1	1
33	1	0	0	1
50	1	1	0	0
32	0	0	1	1
31	0	0	1	1
35	0	0	0	1
24	1	1	1	0
22	0	1	1	0
27	1	1	1	0
34	0	1	1	1
32	1	0	0	1
34	0	1	1	1
31	0	0	0	1
23	1	1	0	0
27	0	1	0	0
52	1	1	0	0
21	1	1	0	0
41	1	0	1	1
31	0	0	1	1
35	1	1	0	0

Imbalanced Data

Even if I do a 3-fold validation and follow best practices, any machine learning algorithm I throw at it (decision tree, random forest, logistic regression, neural network) is going to discriminate on these imbalances.

Because minorities, video gamers, and certain age groups are overrepresented in a sample as criminals, any machine learning algorithm may single them out as criminals even if this is not representative of the population.

The remedy would be to balance every input variable artificially or through careful and deliberate data sampling that's representative of the population.

ARGH!!!!!!!

This applies also to computer vision and autonomous vehicles, where rare traffic situations (e.g. accident conditions, outrageous events) are going to be highly imbalanced.

AGE	VIDEO GAMER	LIKES ICE CREAM	MINORITY	CRIMINAL
55	1	0	0	1
38	0	0	0	0
47	0	1	0	1
35	1	0	1	1
23	0	1	0	1
31	1	1	1	1
33	1	0	0	1
50	1	1	0	0
32	0	0	1	1
31	0	0	1	1
35	0	0	0	1
24	1	1	1	0
22	0	1	1	0
27	1	1	1	0
34	0	1	1	1
32	1	0	0	1
34	0	1	1	1
31	0	0	0	1
23	1	1	0	0
27	0	1	0	0
52	1	1	0	0
21	1	1	0	0
41	1	0	1	1
31	0	0	1	1
35	1	1	0	0

Criminal Justice and Machine Learning

Using machine learning in criminal justice has had challenges.

Minorities are often victim to false positives in facial recognition systems, by a factor of 10 to 100 compared to Caucasian individuals.¹

Paroles and sentencing machine learning systems have had similar problems.²

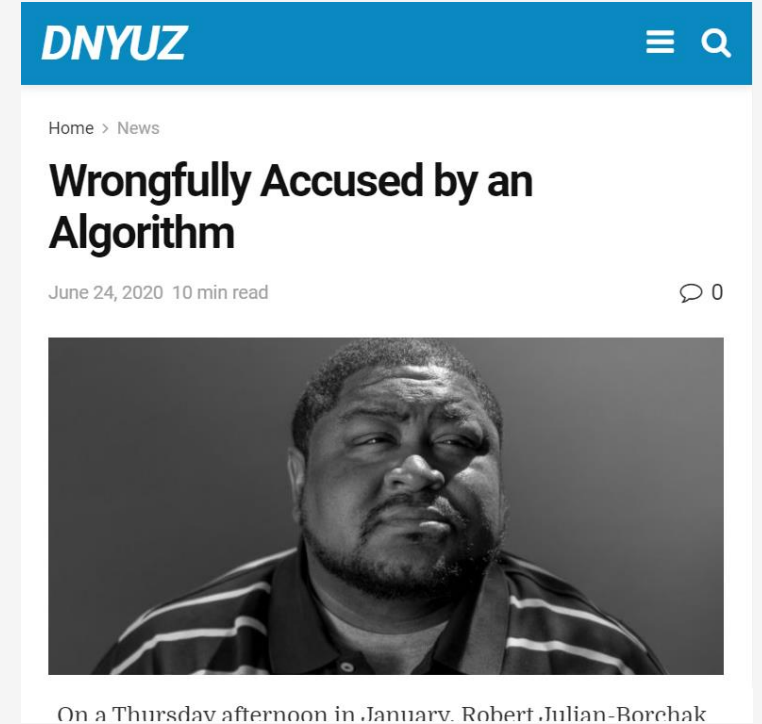
What makes machine learning precarious for criminal justice is it relies on patterns and thus leverage variables that correlate with race, sex, age, and other discriminatory variables.

Criminal datasets can also be imbalanced, not representing citizens fairly across all demographics.

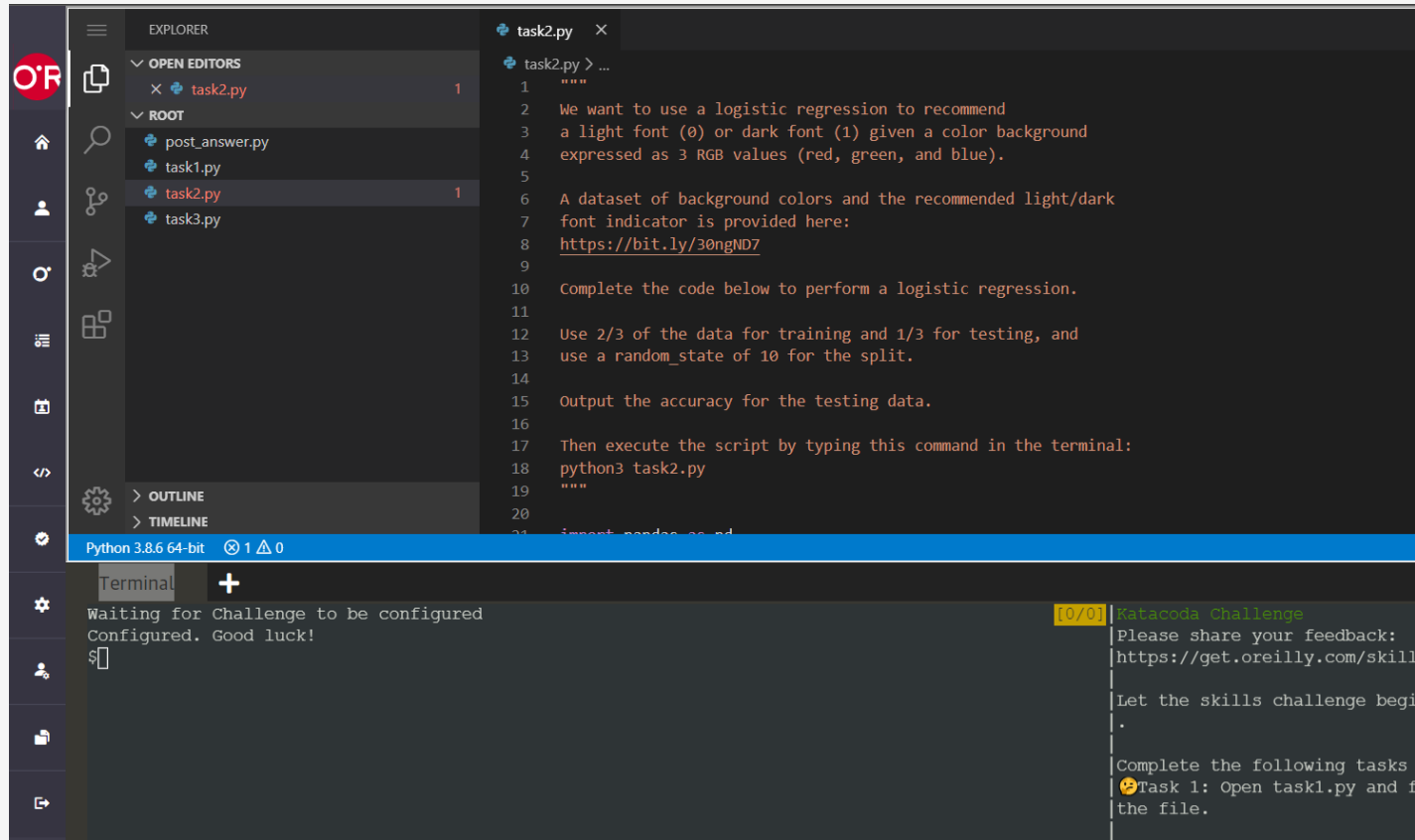
Confounding variables like income, residence, and commute time can also correlate with prejudicial variables as well.

[1] <https://dnyuz.com/2020/06/24/wrongfully-accused-by-an-algorithm/>

[2] <https://www.technologyreview.com/2019/01/21/137783/algorithms-criminal-justice-ai/>



Katacoda – Task 3



The screenshot displays the Katacoda IDE interface. On the left, the Explorer panel shows the file structure with 'task2.py' selected. The main editor area displays the content of 'task2.py', which includes instructions for a logistic regression task. The terminal at the bottom shows the challenge configuration and the start of the task instructions.

```
task2.py
1 """
2 We want to use a logistic regression to recommend
3 a light font (0) or dark font (1) given a color background
4 expressed as 3 RGB values (red, green, and blue).
5
6 A dataset of background colors and the recommended light/dark
7 font indicator is provided here:
8 https://bit.ly/30ngND7
9
10 Complete the code below to perform a logistic regression.
11
12 Use 2/3 of the data for training and 1/3 for testing, and
13 use a random_state of 10 for the split.
14
15 Output the accuracy for the testing data.
16
17 Then execute the script by typing this command in the terminal:
18 python3 task2.py
19 """
20
21 import pandas as pd
```

```
Terminal
Waiting for Challenge to be configured
Configured. Good luck!
$ [0/0] Katacoda Challenge
Please share your feedback:
https://get.oreilly.com/skill

Let the skills challenge begin.

Complete the following tasks
🤖 Task 1: Open task1.py and f
the file.
```

<https://learning.oreilly.com/scenarios/data-science-math/9781098108601/>

Great Books to Learn More About Machine Learning

