

# Creating train, test, and validation datasets

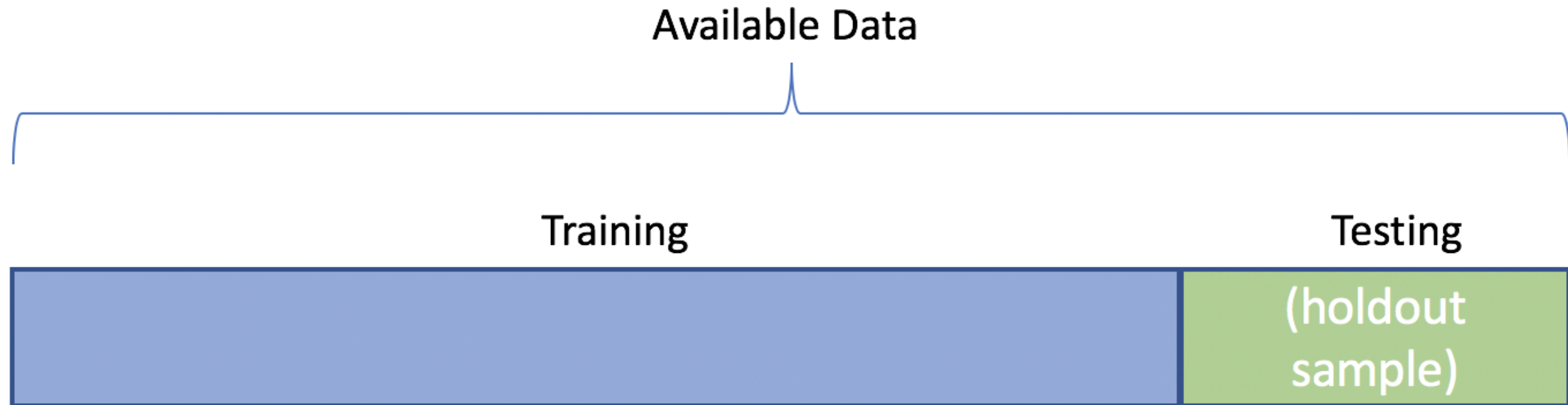
MODEL VALIDATION IN PYTHON



**Kasey Jones**  
Data Scientist

# Traditional train/test split

- Seen data (used for training)
- Unseen data (unavailable for training)



# Dataset definitions and ratios

Dataset	Definition
Train	The sample of data used when fitting models
Test (holdout sample)	The sample of data used to assess model performance

## Ratio Examples

- 80:20
- 90:10 (used when we have little data)
- 70:30 (used when model is computationally expensive)

# The X and y datasets

```
import pandas as pd

tic_tac_toe = pd.read_csv("tic-tac-toe.csv")
X = pd.get_dummies(tic_tac_toe.iloc[:,0:9])
y = tic_tac_toe.iloc[:, 9]
```

Python courses covering dummy variables:

- [Supervised Learning](#)
- [Preprocessing for Machine Learning](#)

# Creating holdout samples

```
X_train, X_test, y_train, y_test =\
    train_test_split(X, y, test_size=0.2, random_state=1111)
```

Parameters:

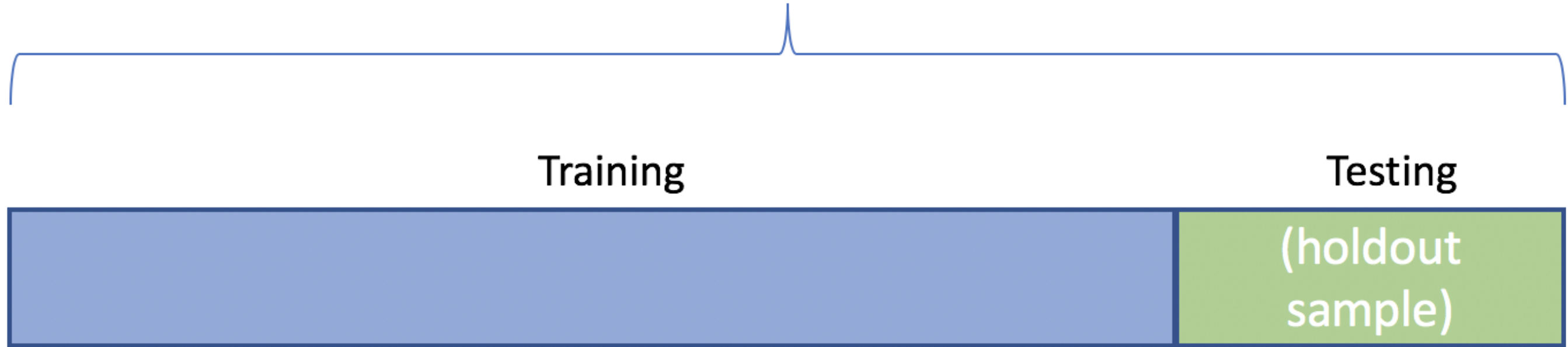
- `test_size`
- `train_size`
- `random_state`

# Dataset for preliminary testing?

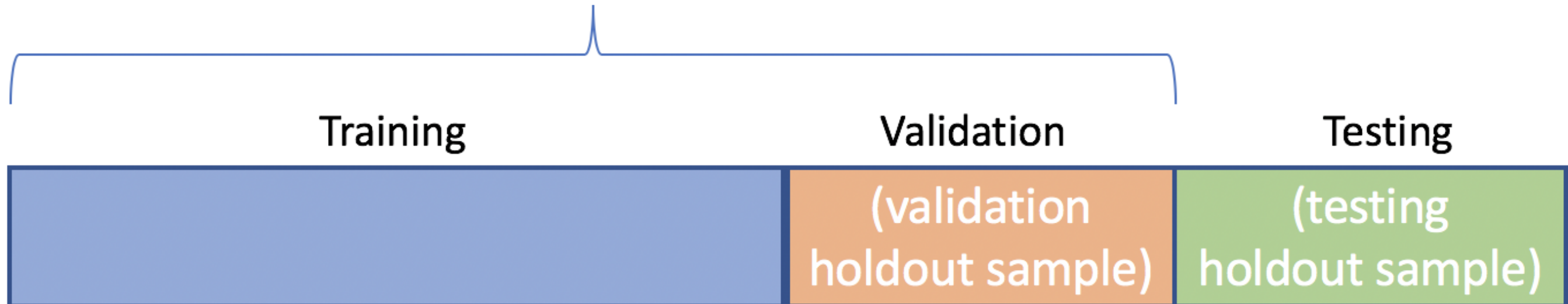
What do we do when testing different model parameters?

- 100 *versus* 1000 trees

Available Data



New Available Data



# Train, validation, test continued

```
X_temp, X_test, y_temp, y_test =\  
    train_test_split(X, y, test_size=0.2, random_state=1111)
```

```
X_train, X_val, y_train, y_val =\  
    train_test_split(X_temp, y_temp, test_size=0.25, random_state=11111)
```

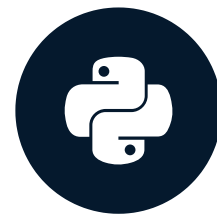


# It's holdout time

MODEL VALIDATION IN PYTHON

# Accuracy metrics: regression models

MODEL VALIDATION IN PYTHON



**Kasey Jones**  
Data Scientist

# Regression models

12.2 points

15 gallons of gas

\$1,323,492

6 new puppies

4,320 people

# Mean absolute error (MAE)

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$

- Simplest and most intuitive metric
- Treats all points equally
- Not sensitive to outliers

# Mean squared error (MSE)

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}$$

- Most widely used regression metric
- Allows outlier errors to contribute more to the overall error
- Random family road trips could lead to large errors in predictions

# MAE vs. MSE

- Accuracy metrics are always application specific
- MAE and MSE error terms are in different units and should not be compared

# Mean absolute error

```
rfr = RandomForestRegressor(n_estimators=500, random_state=1111)
rfr.fit(X_train, y_train)
test_predictions = rfr.predict(X_test)
sum(abs(y_test - test_predictions))/len(test_predictions)
```

9.99

```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, test_predictions)
```

9.99

# Mean squared error

```
sum(abs(y_test - test_predictions)**2)/len(test_predictions)
```

```
141.4
```

```
from sklearn.metrics import mean_squared_error  
mean_squared_error(y_test, test_predictions)
```

```
141.4
```



# Accuracy for a subset of data

```
chocolate_preds = rfr.predict(X_test[X_test[:, 1] == 1])  
mean_absolute_error(y_test[X_test[:, 1] == 1], chocolate_preds)
```

8.79

```
nonchocolate_preds = rfr.predict(X_test[X_test[:, 1] == 0])  
mean_absolute_error(y_test[X_test[:, 1] == 0], nonchocolate_preds)
```

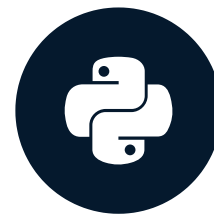
10.99

# Let's practice

MODEL VALIDATION IN PYTHON

# Classification metrics

MODEL VALIDATION IN PYTHON



**Kasey Jones**  
Data Scientist

# Classification metrics

- Precision
- Recall (also called sensitivity)
- Accuracy
- Specificity
- F1-Score, and its variations
- ...

# Classification metrics

- **Precision**
- **Recall** (also called sensitivity)
- **Accuracy**
- Specificity
- F1-Score, and its variations
- ...

# Confusion matrix

		Predicted Values	
		0	1
Actual Values	0	23 (TN)	7 (FP)
	1	8 (FN)	62 (TP)

True Positive: Predict/Actual are both 1

True Negative: Predict/Actual are both 0

False Positive: Predicted 1, actual 0

False Negative: Predicted 0, actual 1

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, test_predictions)
print(cm)
```

```
array([[ 23,   7],
       [   8,  62]])
```

```
cm[<true_category_index>, <predicted_category_index>]
cm[1, 0]
```

```
8
```

# Accuracy

Predicted Values			
		0	1
Actual Values	0	23	7
	1	8	62

$$\frac{23(\text{TN}) + 62(\text{TP})}{23 + 7 + 8 + 62} = .85$$



# Precision

Predicted Values			
		0	1
Actual Values	0	23	7
	1	8	62

$$\frac{62(\text{TP})}{62(\text{TP}) + 7(\text{FP})} = .90$$

# Recall

Predicted Values			
		0	1
Actual Values	0	23	7
	1	8	62

$$\frac{62(\text{TP})}{62(\text{TP})+8(\text{FN})} = .885$$

# Accuracy, precision, recall

```
from sklearn.metrics import accuracy_score, precision_score, recall_score  
accuracy_score(y_test, test_predictions)
```

.85

```
precision_score(y_test, test_predictions)
```

.8986

```
recall_score(y_test, test_predictions)
```

.8857

# Practice time

MODEL VALIDATION IN PYTHON

# The bias-variance tradeoff

MODEL VALIDATION IN PYTHON

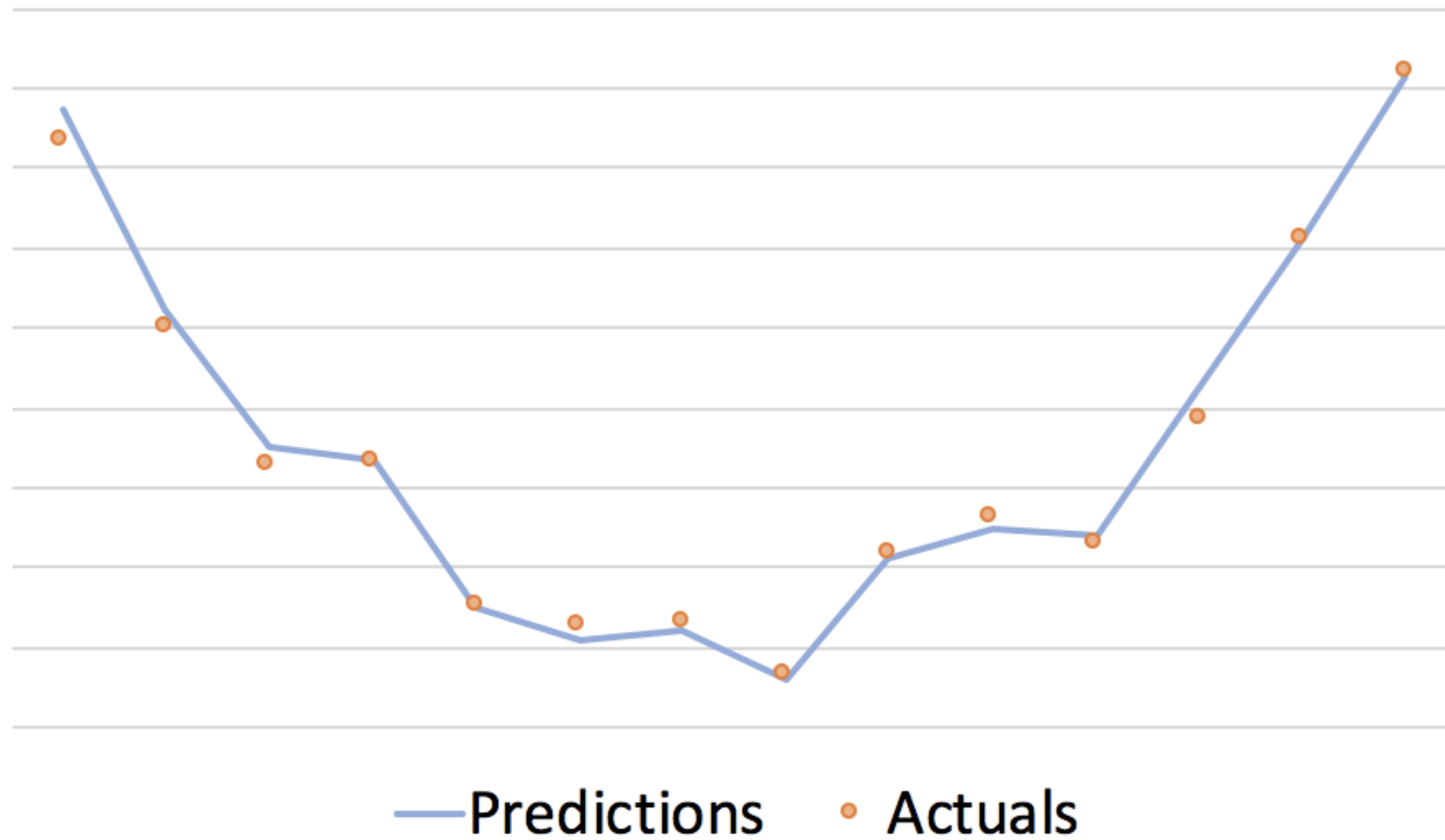


**Kasey Jones**  
Data Scientist

# Variance

- Variance: following the training data too closely
  - Fails to generalize to the test data
  - Low training error but high testing error
  - Occurs when models are overfit and have high complexity

# Overfitting models (high variance)

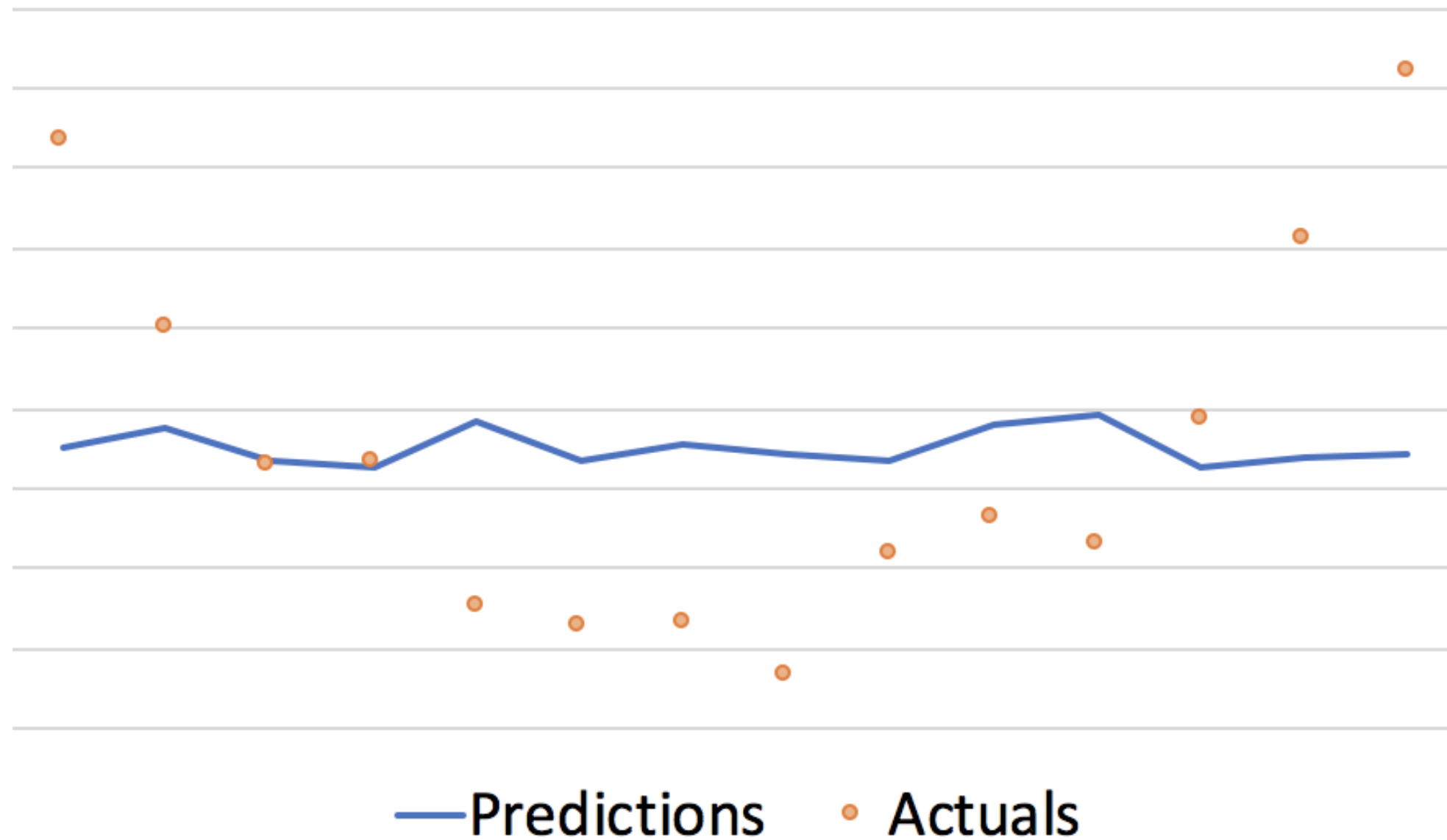


# Bias

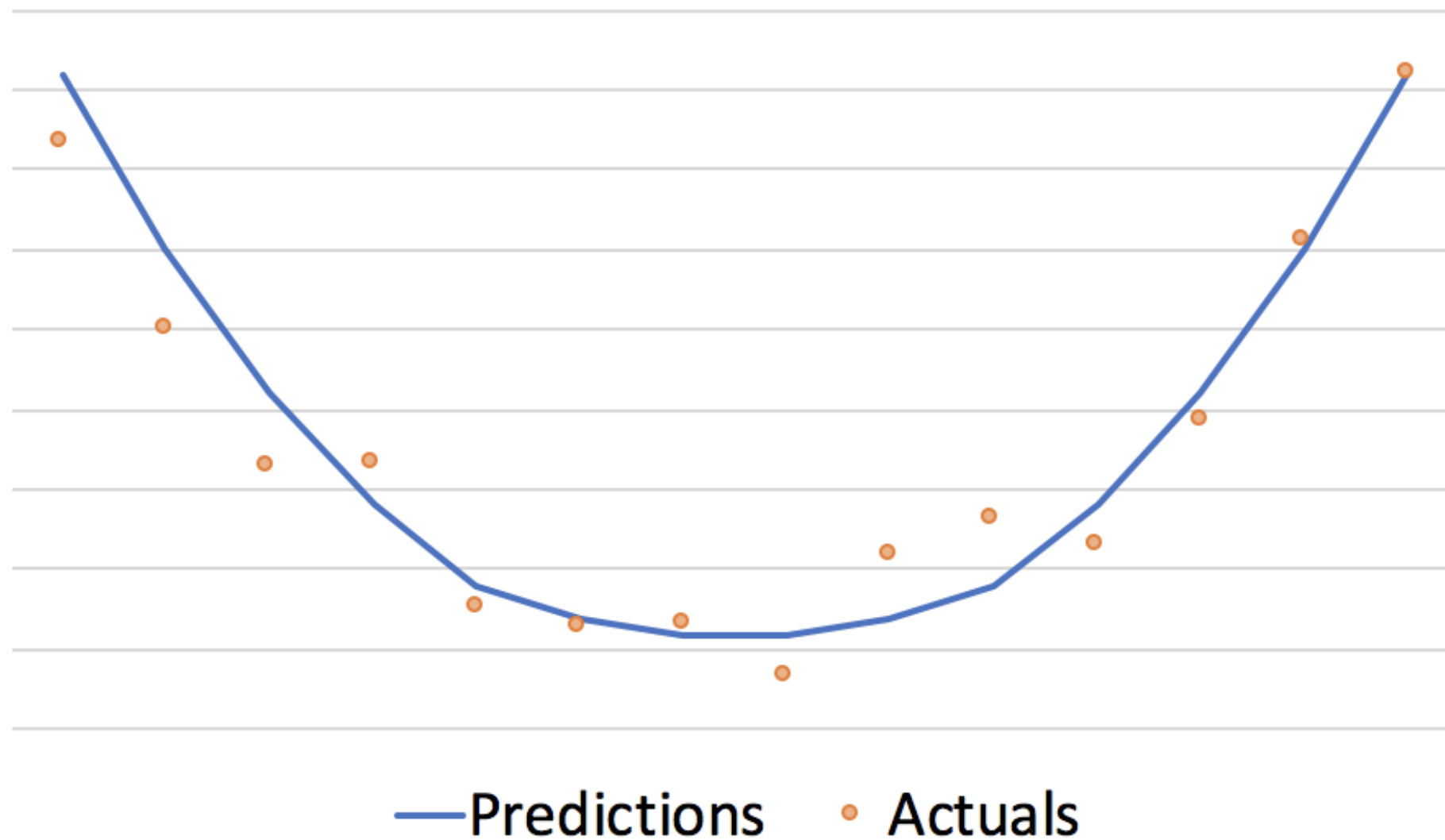
- Bias: failing to find the relationship between the data and the response
  - High training/testing error
  - Occurs when models are underfit



# Underfitting models (high bias)



# Optimal performance



- Bias-Variance Tradeoff

# Parameters causing over/under fitting

```
rfc = RandomForestClassifier(n_estimators=100, max_depth=4)
rfc.fit(X_train, y_train)

print("Training: {0:.2f}".format(accuracy_score(y_train, train_predictions)))
```

```
Training: .84
```

```
print("Testing: {0:.2f}".format(accuracy_score(y_test, test_predictions)))
```

```
Testing: .77
```

```
rfc = RandomForestClassifier(n_estimators=100, max_depth=14)
rfc.fit(X_train, y_train)

print("Training: {0:.2f}".format(accuracy_score(y_train, train_predictions)))
```

```
Training: 1.0
```

```
print("Testing: {0:.2f}".format(accuracy_score(y_test, test_predictions)))
```

```
Testing: .83
```

```
rfc = RandomForestClassifier(n_estimators=100, max_depth=10)
rfc.fit(X_train, y_train)

print("Training: {0:.2f}".format(accuracy_score(y_train, train_predictions)))
```

```
Training: .89
```

```
print("Testing: {0:.2f}".format(accuracy_score(y_test, test_predictions)))
```

```
Testing: .86
```

**Remember, only you  
can prevent  
overfitting!**

MODEL VALIDATION IN PYTHON