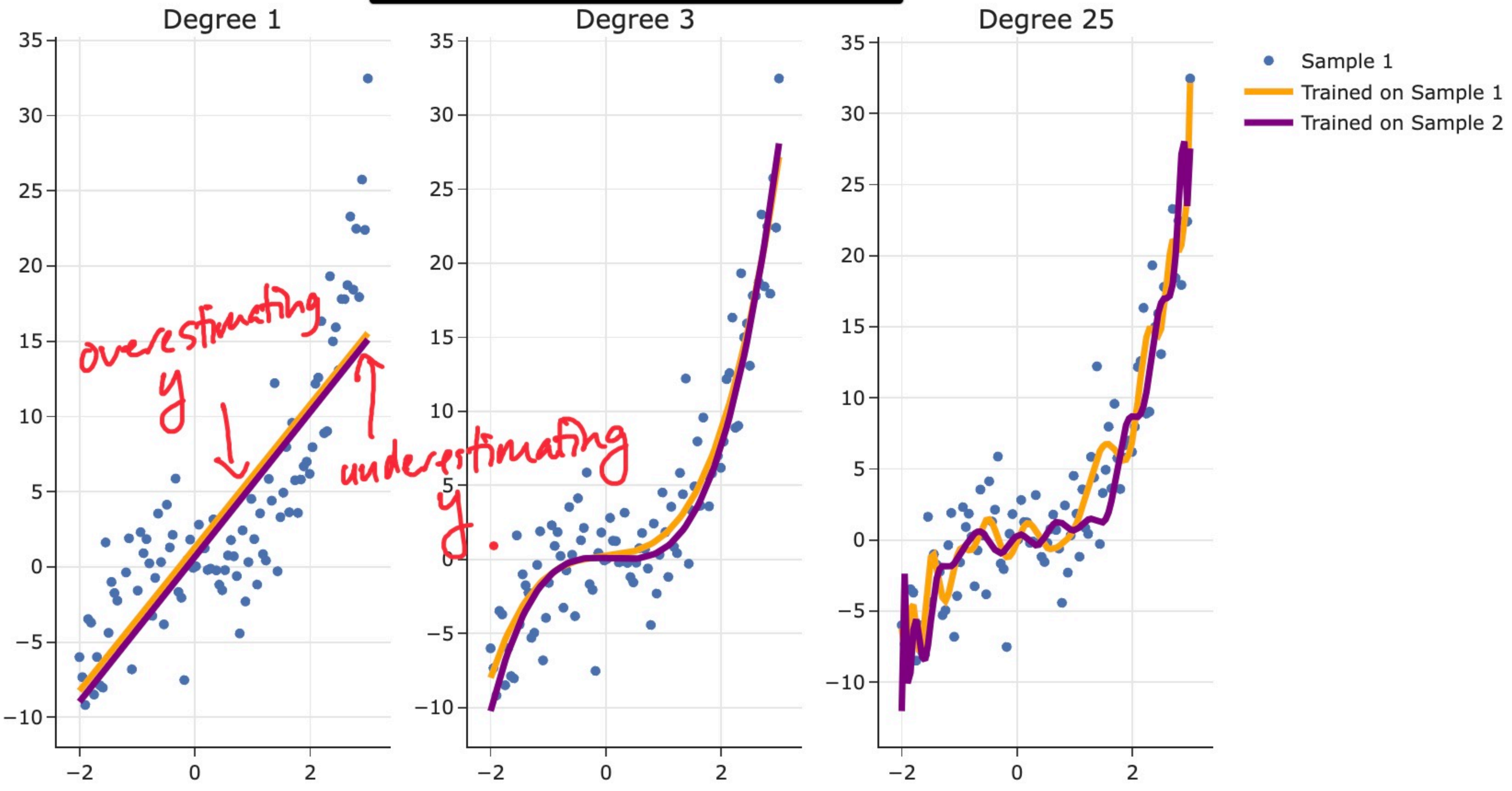
localhost

In [4]:
```python
from sklearn.preprocessing import PolynomialFeatures
# fit_transform fits and transforms the same input.
# We tell it not to add a column of 1s, because
# LinearRegression() does this automatically later on.
d2 = PolynomialFeatures(3, include_bias=False)
d2.fit_transform(np.array([1, 2, 3, 4, -2]).reshape(-1, 1))
```

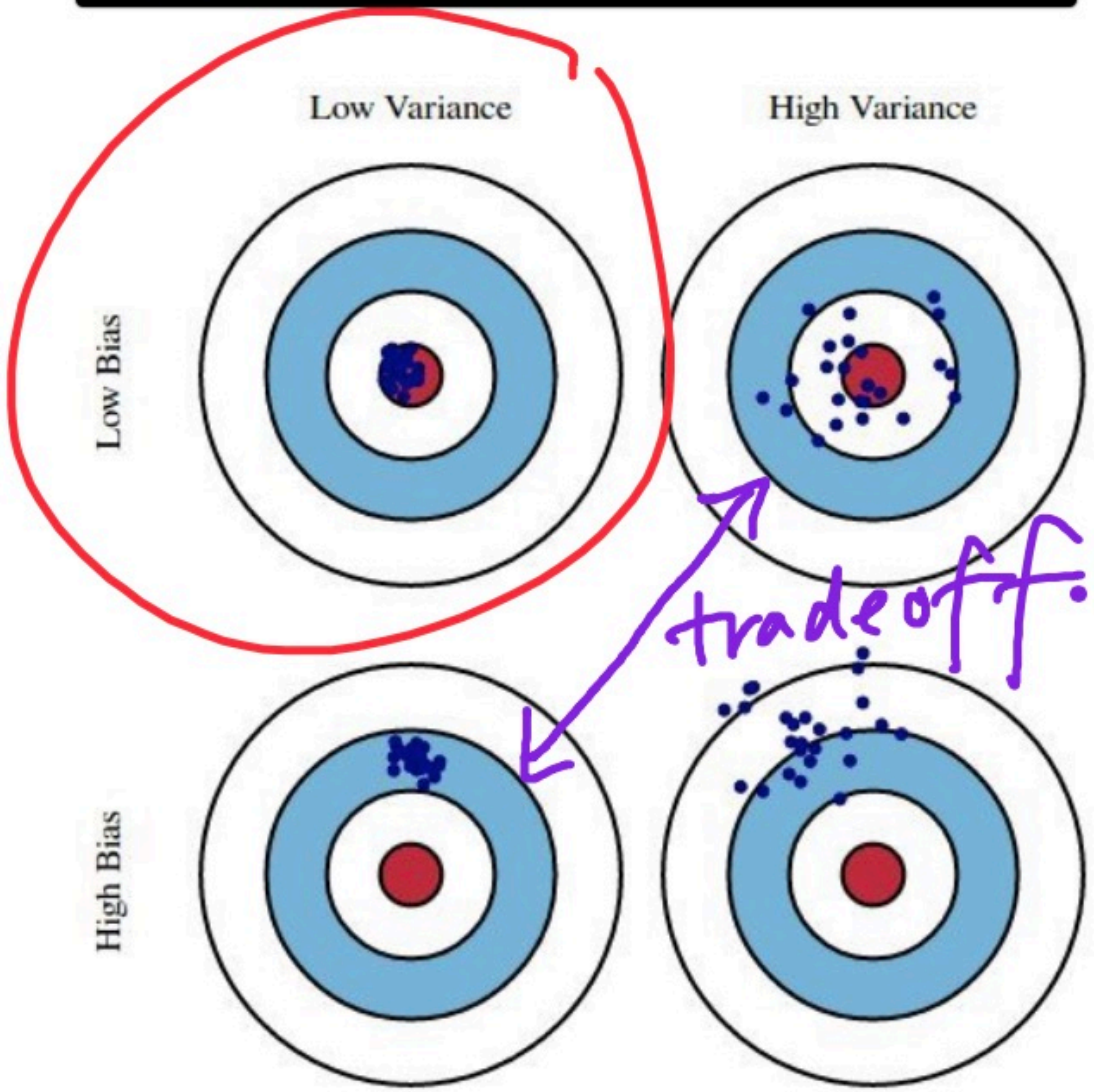
Out[4]:
```
array([[ 1.,  1.,  1.],
       [ 2.,  4.,  8.],
       [ 3.,  9., 27.],
       [ 4., 16., 64.],
       [-2.,  4., -8.]])
```

x x^2 x^3



- Low bias is good! ✅

ideal



- Here, suppose:
  - The **red bulls-eye** represents your **true weight and height** 🧍.
  - The **dark blue darts** represent **predictions of your weight and height** using different models that were fit using different samples drawn from the same population.

$$\vec{w}^* = \underset{\vec{w}}{\text{argmin}} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - H(\vec{x}_i) \right)^2$$

- **Key idea**: A model that works well on past data should work well on future data, if future data looks like past data.

- What we really want is for the:
  - **expected** loss for a new data point $(\vec{x}_{\text{new}}, y_{\text{new}})$,
  - drawn from the same population as the training set, to be small.

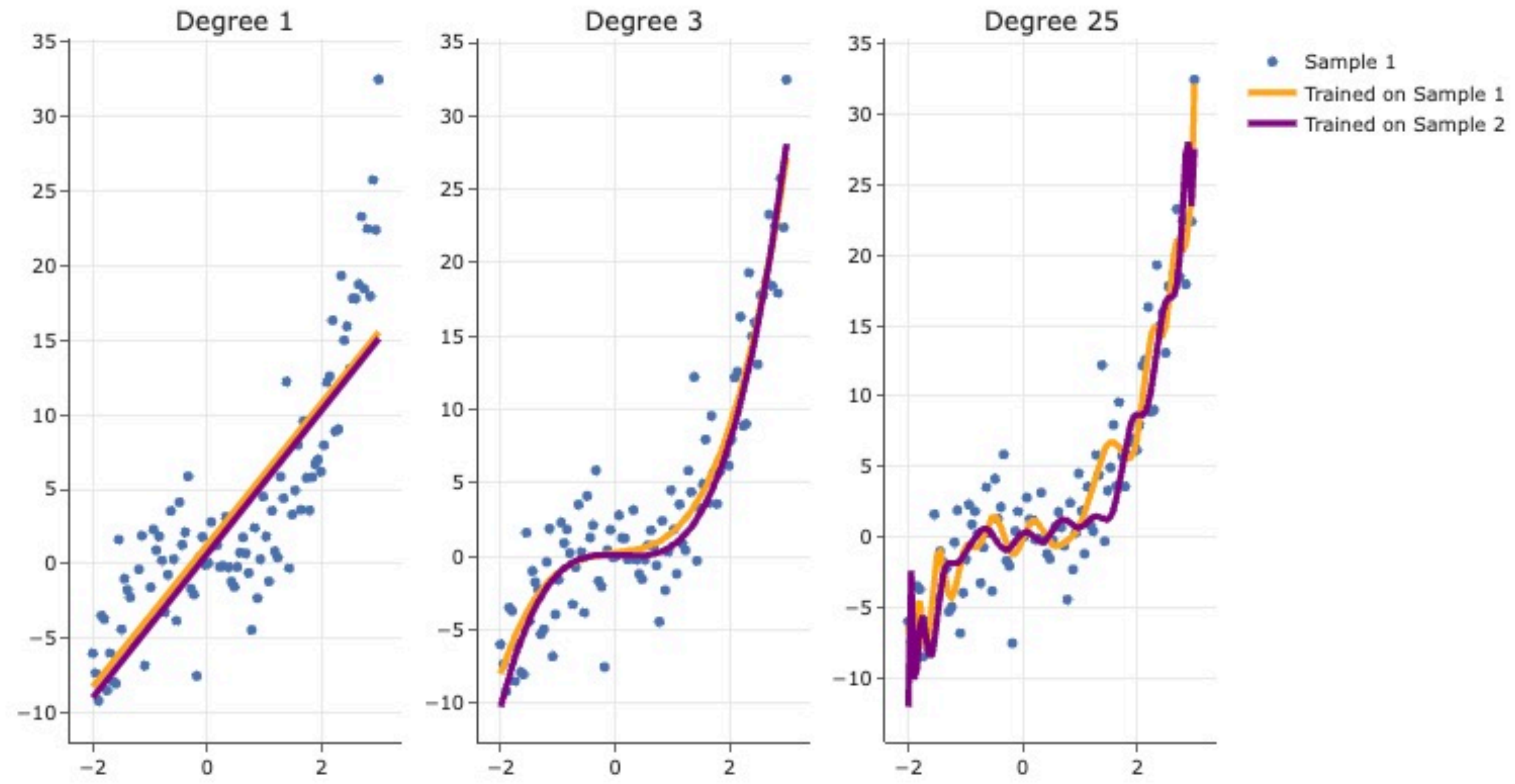
That is, we want to minimize **risk**:

$$\text{risk} = \mathbb{E}[y_{\text{new}} - H(\vec{x}_{\text{new}})]^2$$

X_{new}, y_{new} drawn from same population distribution.

expected value of a random variable

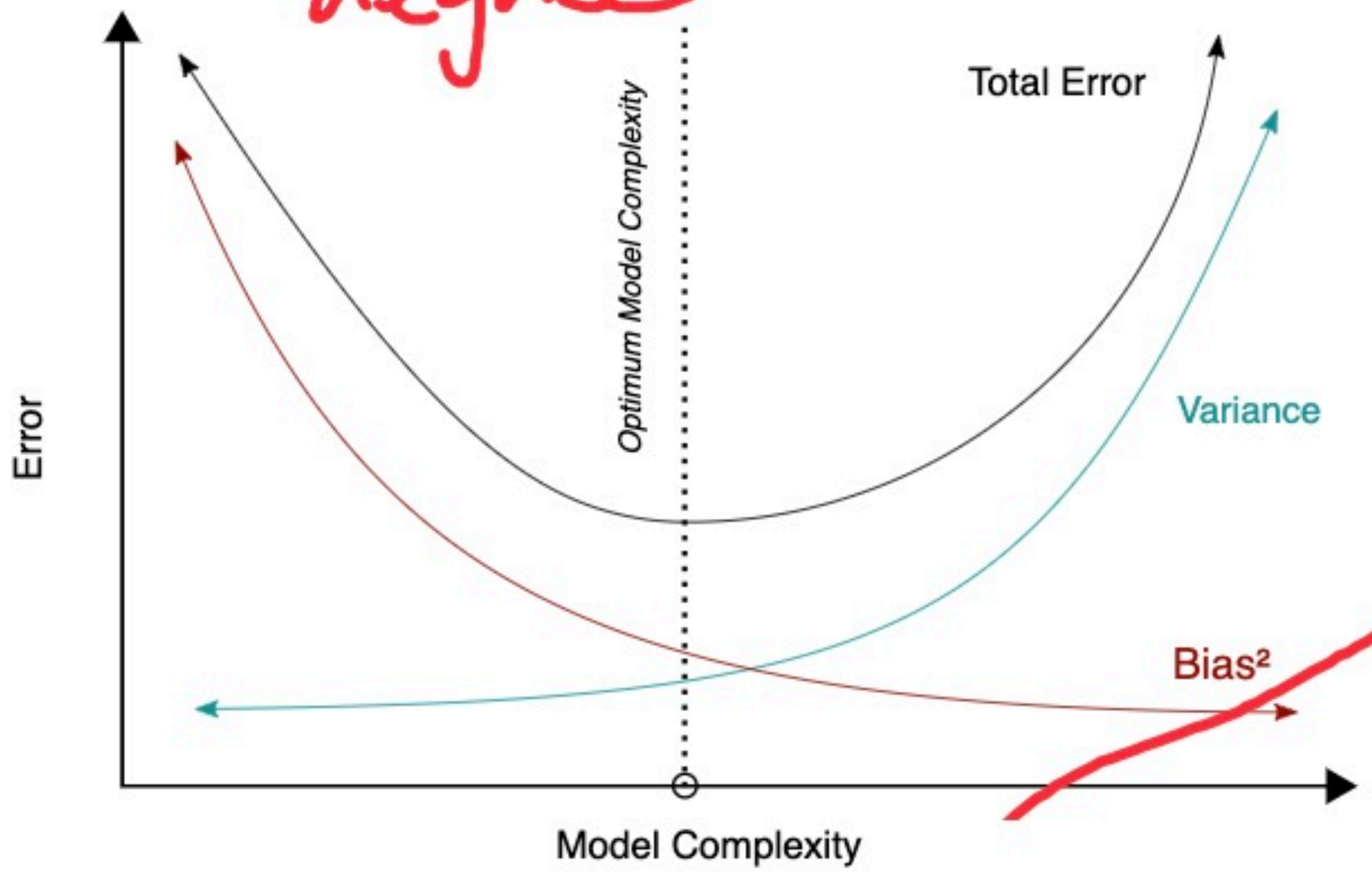
```
In [10]: fig
```



- As model variance increases, model bias tends to decrease, and vice versa.

  That is, there is a **tradeoff** between bias and variance:

degree →

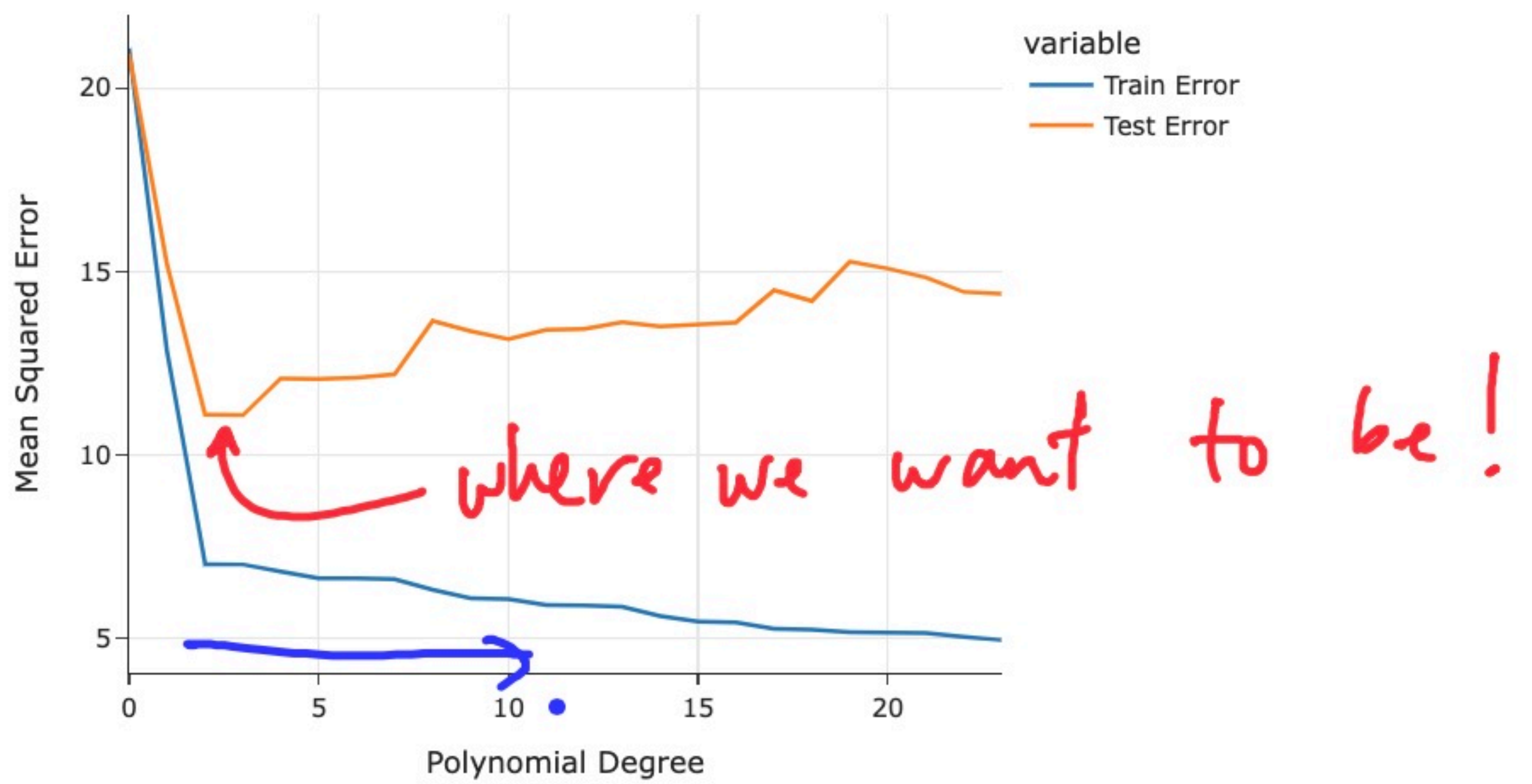


polynomial degree = complexity

- Now that we've performed a train/test split of Sample 1, we'll create models through 25 polynomial features and compute their train and test errors.

In [ ]:
```
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
train_errs = []
test_errs = []
for d in range(1, 26):
    pl = make_pipeline(PolynomialFeatures(d, include_bias=False), LinearRegression())
    pl.fit(X_train, y_train)
    train_errs.append(mean_squared_error(y_train, pl.predict(X_train)))
    test_errs.append(mean_squared_error(y_test, pl.predict(X_test)))
errs = pd.DataFrame({'Train Error': train_errs, 'Test Error': test_errs})
errs
```

```
fig.update_layout(showlegend=True, xaxis_title="Polynomial Degree", yaxis_title="Mean Squared Error
```



- Training error appears to decrease as polynomial degree increases.
- Test error appears to decrease until a "valley", and then increases again.
- Here, we'd choose a degree of 3, since that degree has the **lowest test error**.

GridSearchCV

- Let's use k-fold cross-validation to choose a polynomial degree that best generalizes to unseen data.

As before, we'll choose our polynomial degree from the list [1, 2, ..., 25].

- GridSearchCV takes in:
    - an **un-fit** instance of an estimator, and
    - a **dictionary** of hyperparameter values to try,

and performs k-fold cross-validation to find the **combination of hyperparameters** with the best average validation performance.

In [33]:
```
from sklearn.model_selection import GridSearchCV
GridSearchCV?
```

- Why do you think it's called "grid search"?

Interpreting the results of k-fold cross-validation

- Let's peek under the hood.

```
In [46]: errs_df = util.format_results(searcher)
         errs_df
```

Out[46]:

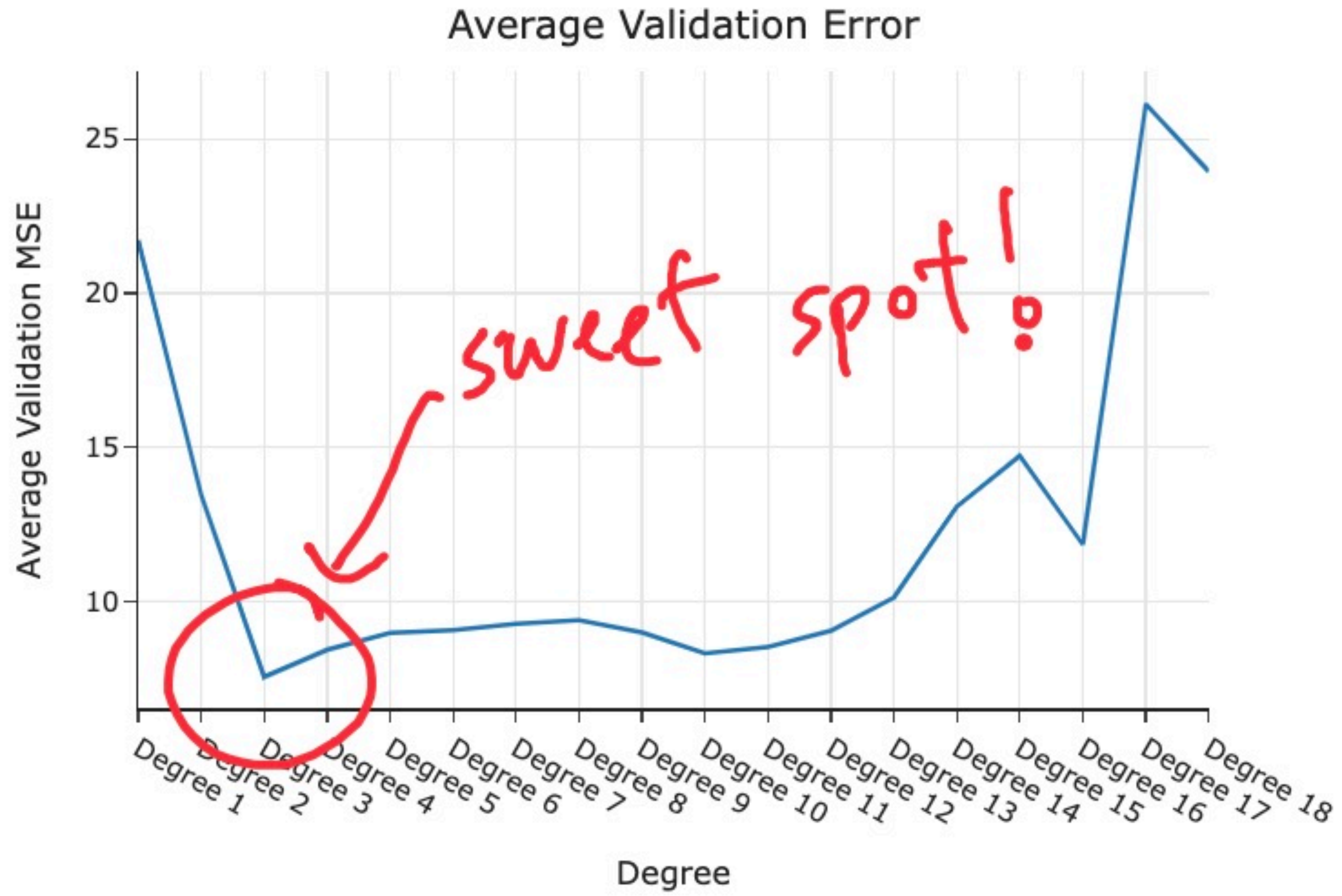
	Degree 1	Degree 2	Degree 3	Degree 4	...	Degree 22	Degree 23	Degree 24	Degree 25
Fold 1	24.36	14.06	7.45	7.62	...	40.48	69.03	43.38	220.49
Fold 2	31.50	19.17	7.05	10.85	...	796.97	1446.11	2208.78	5666.03
Fold 3	15.64	11.78	9.39	9.48	...	61.79	80.64	474.69	4620.71
Fold 4	25.95	17.57	9.47	9.46	...	12.24	12.02	11.86	15.02
Fold 5	11.12	4.74	4.44	4.81	...	8.85	8.54	8.65	8.27

5 rows × 25 columns

validation MSE when using a degree 3 polynomial trained on slices 1, 2, 4, 5 (slice 3 = validation)!

Out[49]:
```
Degree 3        7.56
Degree 10       8.33
Degree 4        8.44
                 ...
Degree 23     323.27
Degree 24     549.47
Degree 25    2106.10
Length: 25, dtype: float64
```

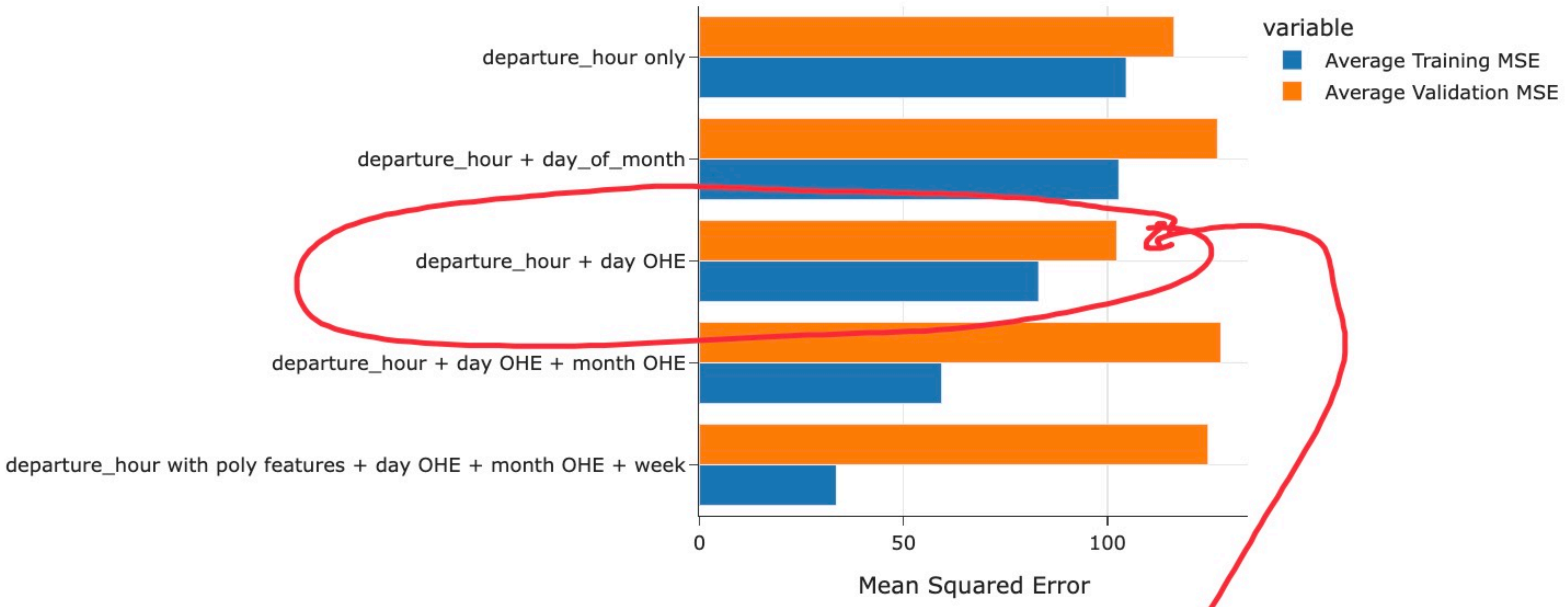
In [50]:
```
fig = errs_df.mean(axis=0).iloc[:18].plot(kind='line', title='Average Validation Error')
fig.update_layout(xaxis_title='Degree', yaxis_title='Average Validation MSE', showlegend=False)
```



In [ ]:
```
# Chosen automatically by sklearn.
```

35.1

```
    .update_layout(xaxis_title='Mean Squared Error', yaxis_title='Model')
)
commute_models_summarized
```



smallest average validation MSE.