In [9]:
```
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))
```

because Lin. Reg. already adds intercept!

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

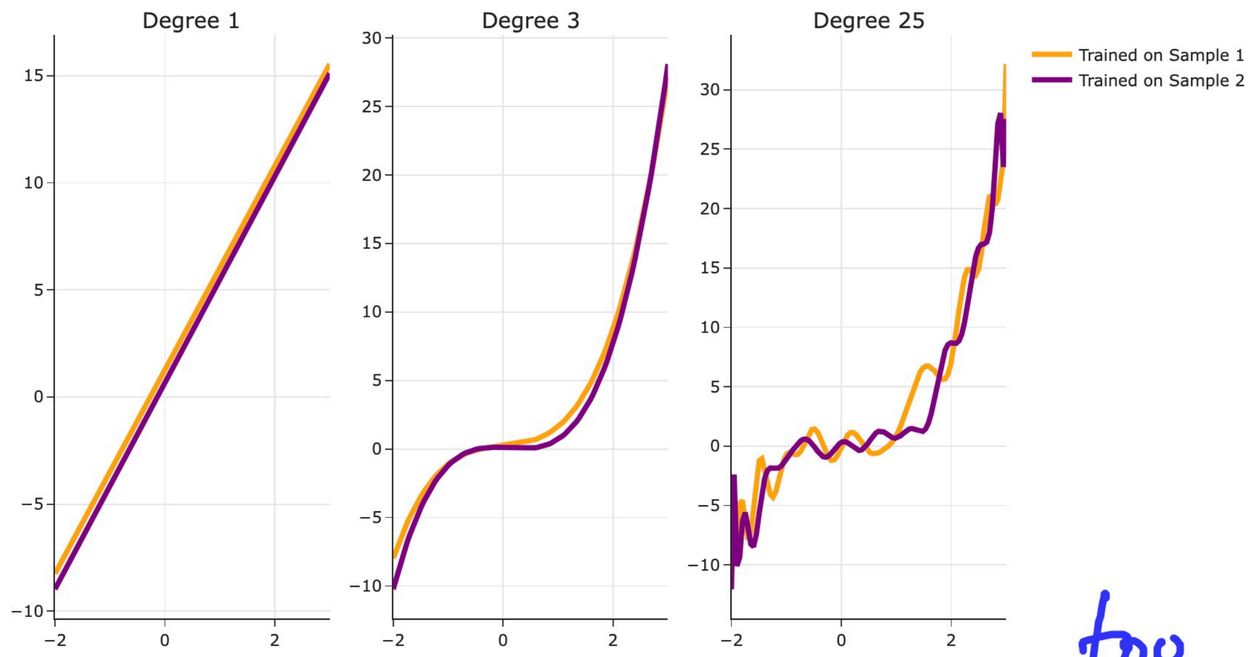
x^1 x^2 x^3

?

- **Model variance: The variance of a model's predictions, across all datasets.**

- In other words, for a given $\vec{x}_i$, how much does $H^*(\vec{x}_i)$ vary across all datasets?

In [13]:
```
1 fig
```



too flexible.

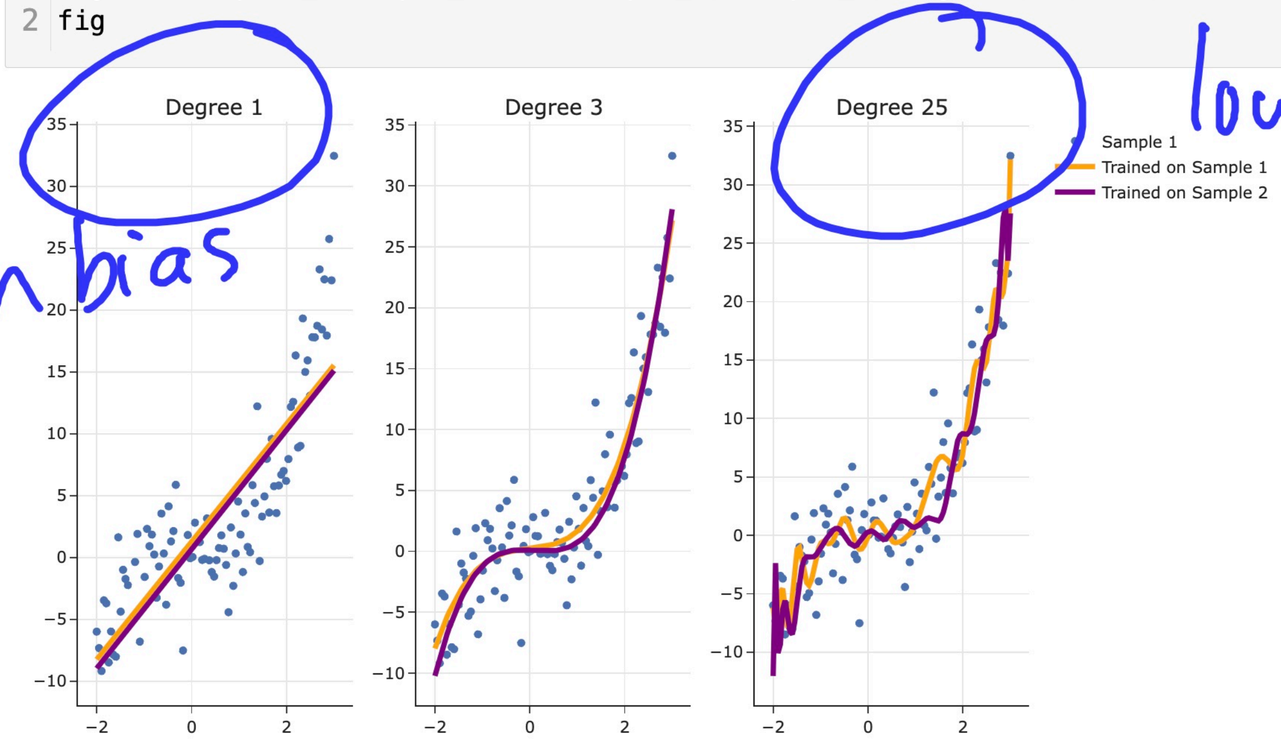
- Low model variance is good! ✅

- High model variance is a sign of **overfitting**, i.e. that our model is too **complicated** and is prone to fitting to the noise in our training data.

- **Model bias: The averaged deviation between a predicted value and an actual value, across all datasets.**

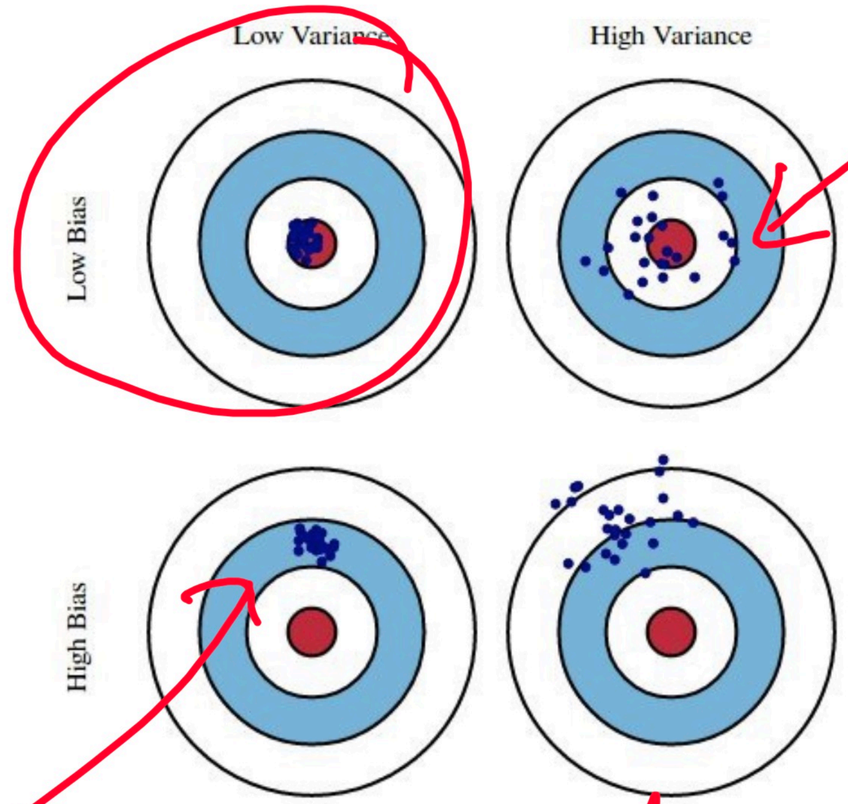
- In other words, for a given $\vec{x}_i$, how far is $H^*(\vec{x}_i)$ from the true y_i, on average?

In [14]:
```
1 fig = util.plot_multiple_models(sample_1, sample_2, degs=[1, 3, 25], data=True)
2 fig
```



high bias

low bias
(on average, they make good predictions)

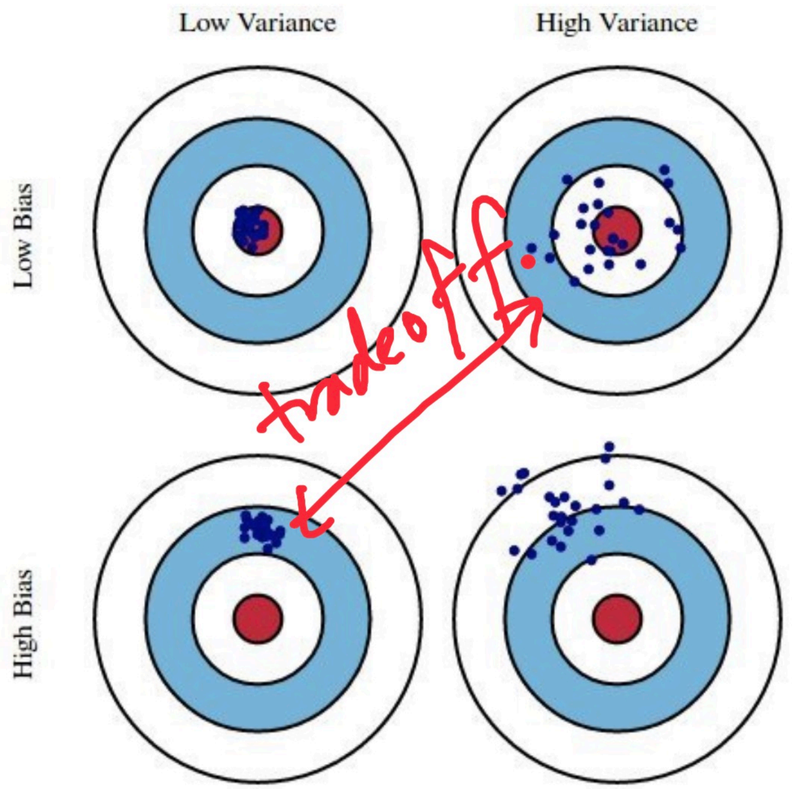


ideal

on average, predictions are good, but each one can be very wrong & different

wrong, in a consistent fashion

- 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.



- 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.

- We'd like our models to be in the top left, but in practice that's hard to achieve!

Risk vs. empirical risk

aka "average loss":

- Since Lecture 11, we've minimized **empirical risk** to find optimal model parameters $\vec{w}^*$:

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

$$\vec{w} = \underset{\vec{w}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - H(\vec{x}_i))$$

- **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**:

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

expected value of a random variable

- In general, we don't know the entire population distribution of xs and ys, so we can't compute risk exactly.

  That's why we compute **empirical** risk!

empirical risk ≈ risk.

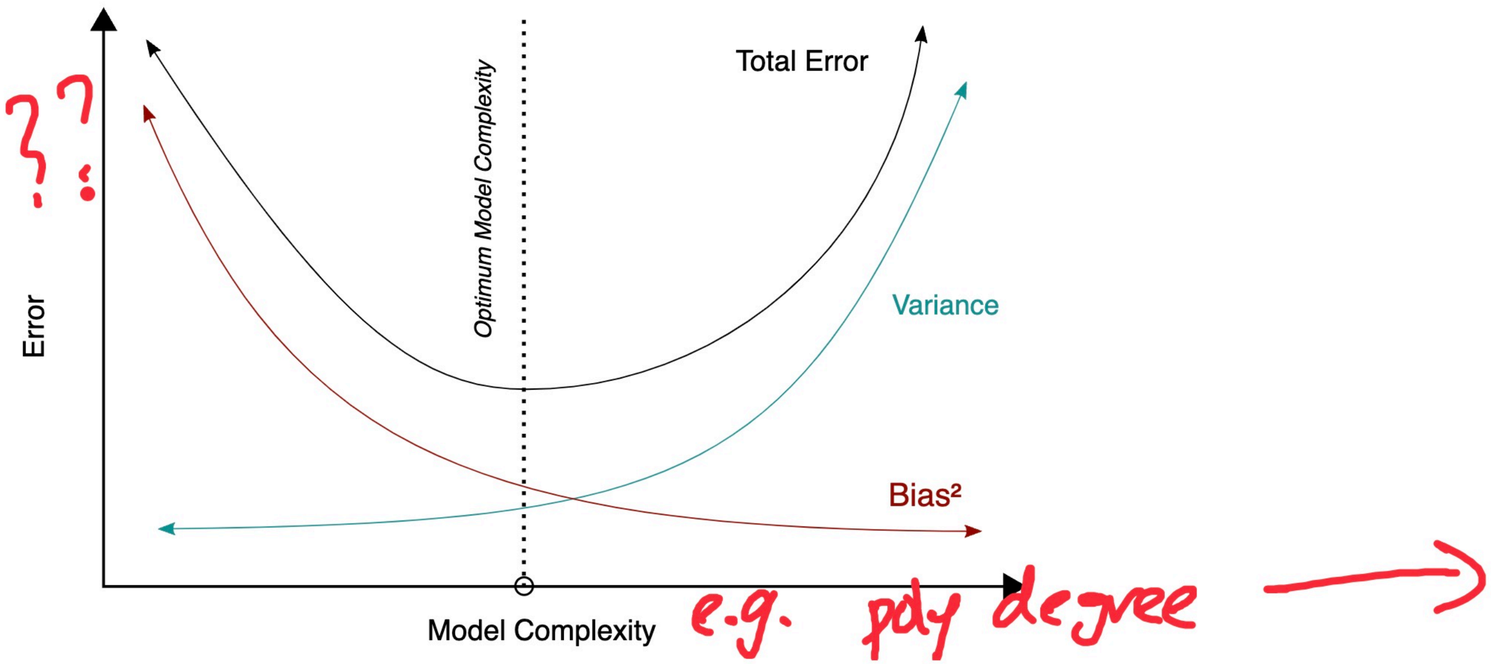
$$\mathbb{E}[y_{\text{new}} - H(\vec{x}_{\text{new}})]^2 \approx \frac{1}{n} \sum_{i=1}^{n} \left(y_i - H(\vec{x}_i)\right)^2 = R(H)$$

$$R_{\text{sq}}(h) = \frac{1}{n}\sum_{i=1}^{n}(y_i - h)^2 = \underbrace{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2}_{\text{variance of } y} + (\bar{y} - h)^2$$

- **Key takeaway**: If we care about minimizing (empirical) risk, we can equivalently try to minimize both model bias and model variance.

- As model variance increases, model bias tends to decrease, and vice versa.
  That is, there is a **tradeoff** between bias and variance:

how do we choose model complexity???



Out[18]:

	x	y
0	-2.00	-6.00
1	-1.95	-7.33
2	-1.90	-9.18
...	...	...
97	2.90	25.75
98	2.95	22.40
99	3.00	32.47

100 rows × 2 columns

In [19]:
```
X = sample_1[['x']]
y = sample_1['y']
# We don't have to choose 0.25.
# We also don't have to set a random_state;
# we've done this so that we get the same results in lecture every time.
train_test_split(X, y, random_state=23)
```

Out[19]: Ellipsis

good for reproducibility.

```
# We also don't have to set a random_state;
# we've done this so that we get the same results in lecture every time.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=23)
```

- Before proceeding, let's check the sizes of `X_train` and `X_test`.

In [24]:
```
print('Rows in X_train:', X_train.shape[0])
display(X_train.head())
print('Rows in X_test:', X_test.shape[0])
display(X_test.head())
```

```
Rows in X_train: 75
```

	x
4	-1.80
53	0.68
5	-1.75
81	1.09
98	2.95

randomly chosen!..

```
Rows in X_test: 25
```

	x
26	-0.69
80	2.04
82	2.14
68	1.43
77	1.89

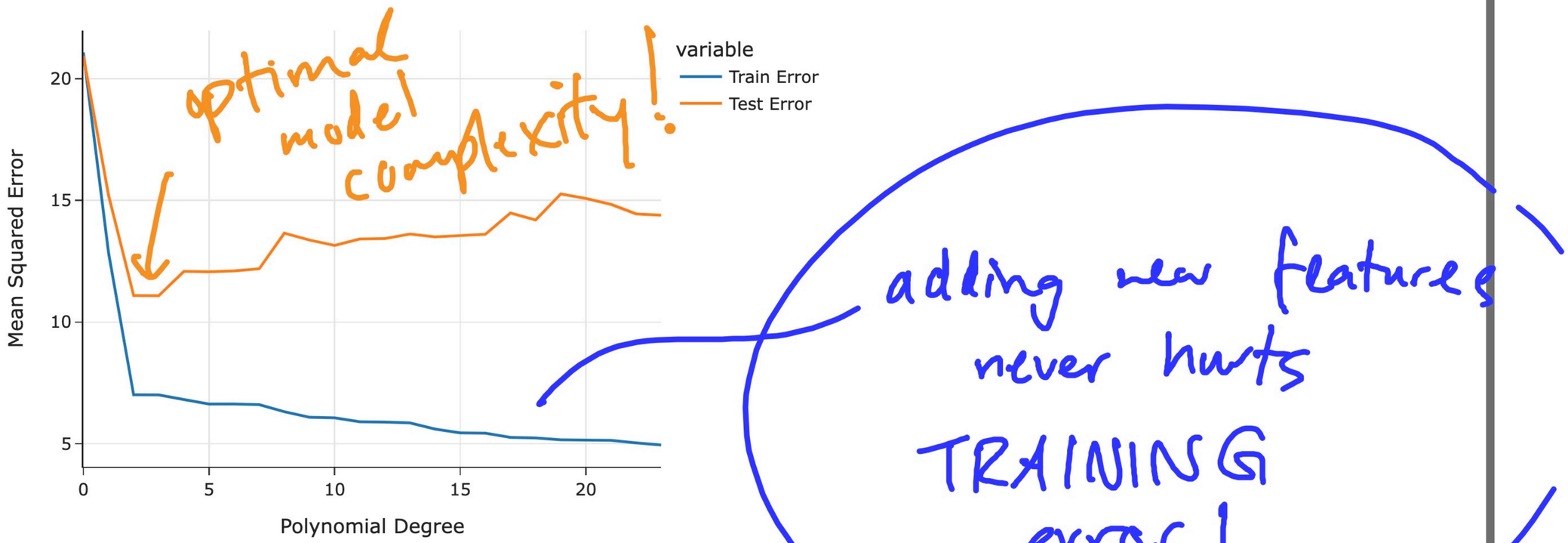
ow that we've performed a train/test split of Sample 1, we'll create models w

rough 25 polynomial features and compute their train and test errors.

```
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
```

- Let's look at the plots of training error vs. degree and test error vs. degree.

In [26]:
```
1 fig = px.line(errs.iloc[:-1])
2 fig.update_layout(showlegend=True, xaxis_title='Polynomial Degree', yaxis_title='Mea
```



`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 [29]:
```
1 from sklearn.model_selection import GridSearchCV
2 GridSearchCV?
```

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

for each of 20 hyperparams:
train 10x (9x including X.iloc[0],
1x with n valid)

Question 🤔 (Answer at practicaldsc.org/q)

- Suppose you have a training dataset with ~~1000~~ rows.
- You want to decide between 20 hyperparameters for a particular model.
- To do so, you perform 10-fold cross-validation.
- **How many times is the first row in the training dataset (`X.iloc[0]`) used for training a model?**

sl: $20 \times 9 = \boxed{180}$

```
    results
    .sort_values('Average Training MSE')
    .plot(kind='barh', barmode='group', width=1000)
    .update_layout(xaxis_title='Mean Squared Error', yaxis_title='Model')
)
commute_models_summarized
```