The general problem

commute time$_1$ commute time$_n$

- We have n data points, $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_n, y_n)$, where each $\vec{x}_i$ is a feature vector of d features:

$$\vec{x}_i = \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \\ \vdots \\ x_i^{(d)} \end{bmatrix}$$

new variables?

e.g. $\vec{x}_i = \begin{bmatrix} \text{departure hour}_i \\ \text{day of month}_i \end{bmatrix}_2$

$\text{Aug}(\vec{x}_i) = \begin{bmatrix} 1 \\ \text{departure hour}_i \\ \text{day of month}_i \end{bmatrix}_3$

- We have n data points, $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_n, y_n)$, where each $\vec{x}_i$ is a feature vector of d features:

$$\vec{x}_i = \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \\ \vdots \\ x_i^{(d)} \end{bmatrix}$$

- We want to find a good linear hypothesis function:

$$H(\vec{x}_i) = w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} + \dots + w_d x_i^{(d)} = \vec{w} \cdot \overbrace{\text{Aug}(\vec{x}_i)}^{d+1 \text{ elements}}$$

$$\vec{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} \qquad \text{Aug}(\vec{x}_i) = \begin{bmatrix} 1 \\ x_i^{(1)} \\ x_i^{(2)} \\ \vdots \\ x_i^{(d)} \end{bmatrix}$$

$$\begin{bmatrix} \vdots \\ x_i^{(d)} \end{bmatrix}$$

- We want to find a good linear hypothesis function:

$$H(\vec{x}_i) = w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} + \ldots + w_d x_i^{(d)} = \vec{w} \cdot \text{Aug}(\vec{x}_i)$$

- Specifically, we want to find the optimal parameters, $w_0^*, w_1^*, \ldots, w_d^*$ that minimize mean squared error:

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - H(\vec{x}_i))^2 \longrightarrow \text{avg}\left(\left(\text{actual} - \text{predicted}\right)^2\right)$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left(y_i - (w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} + \ldots + w_d x_i^{(d)})\right)^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \text{Aug}(\vec{x}_i) \cdot \vec{w}\right)^2$$

$$= \frac{1}{n} \|\underbrace{\vec{y} - X\vec{w}}\|^2$$

$\vec{y} - X\vec{w} = \vec{e}$

design matrix

minimized $R_{sq}(\vec{w})$ using linear algebra

5.1

The general solution

- Define the **design matrix** $X \in \mathbb{R}^{n\times(d+1)}$ and **observation vector** $\vec{y} \in \mathbb{R}^n$:

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(d)} \\ 1 & x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(d)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(d)} \end{bmatrix} = \begin{bmatrix} \text{Aug}(\vec{x_1})^T \\ \text{Aug}(\vec{x_2})^T \\ \vdots \\ \text{Aug}(\vec{x_n})^T \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$n \times (d+1)$

$$\text{Aug}(\vec{x}_1) = \begin{bmatrix} 1 \\ x_1^{(1)} \\ x_1^{(2)} \\ \vdots \\ x_1^{(d)} \end{bmatrix}$$

- Define the **design matrix** $X \in \mathbb{R}$

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(d)} \\ 1 & x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(d)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(d)} \end{bmatrix} = \begin{bmatrix} \text{Aug}(\vec{x_1})^T \\ \text{Aug}(\vec{x_2})^T \\ \vdots \\ \text{Aug}(\vec{x_n})^T \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

orthogonal

perpendicular.

- Then, solve the **normal equations** to find the optimal parameter vector, $\vec{w}^*$:

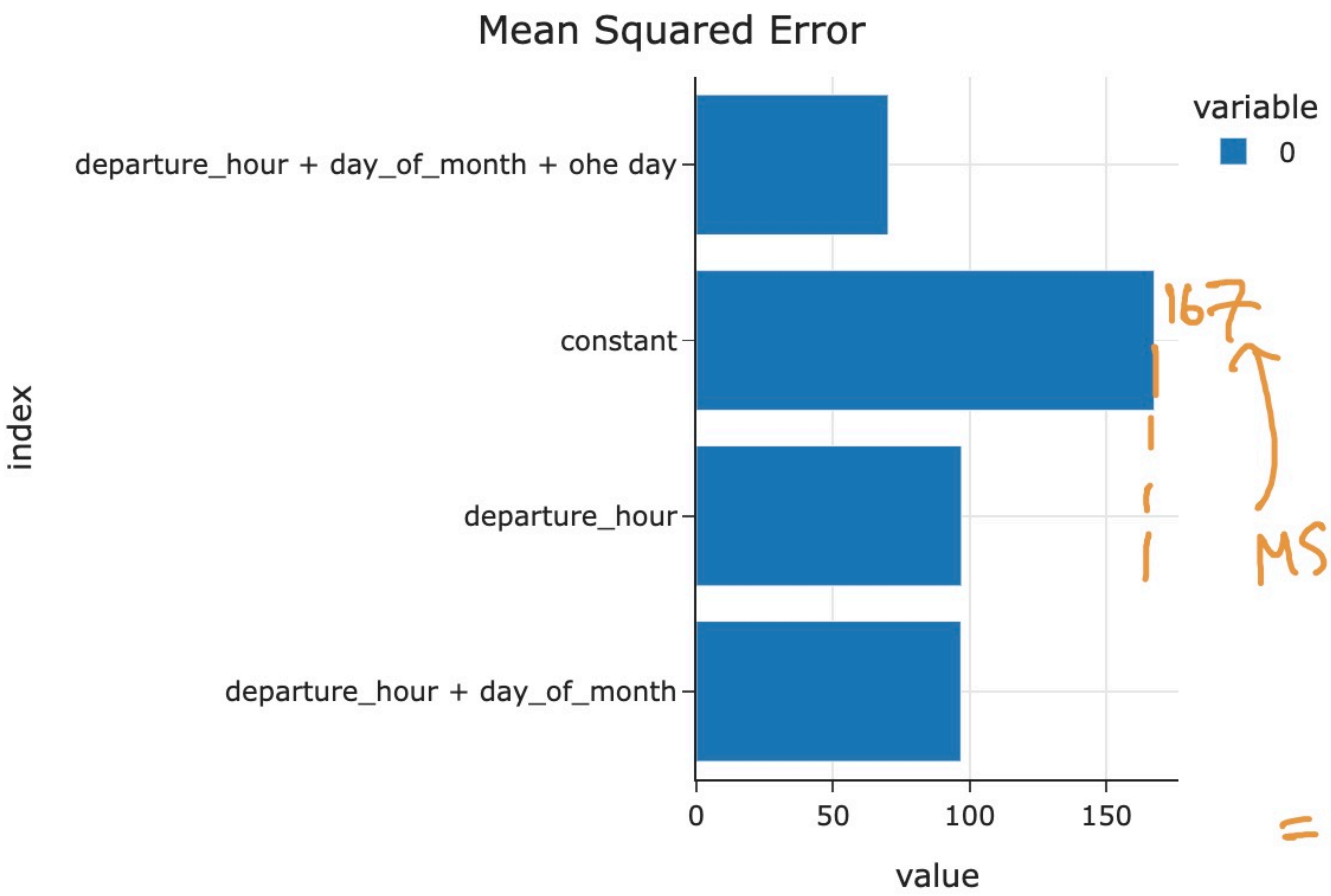
system of d+1 equations,
d+1 unknowns

$$X^T X \vec{w}^* = X^T \vec{y}$$

- The $\vec{w}^*$ that satisfies the equations above minimizes mean squared error, $R_{\text{sq}}(\vec{w})$. $= \frac{1}{n} \|\vec{y} - X\vec{w}\|^2$

6.1

```python
pd.Series(mse_dict).plot(kind='barh', title='Mean Squared Error')
```



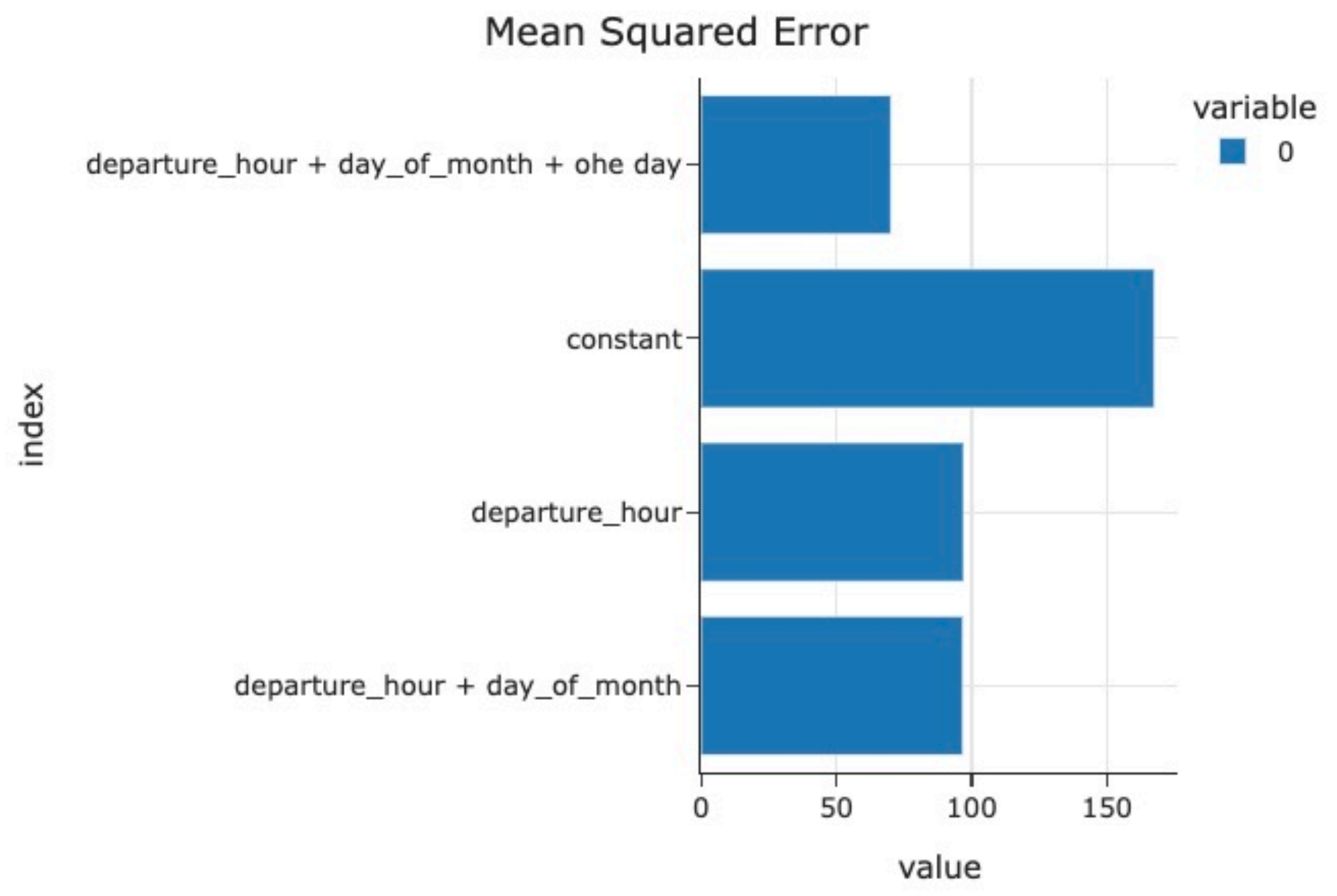
MSE (constant model that uses $h^* = \text{Mean}(y_1, \ldots, y_n)$)

= variance of ys!..

Comparing our latest model to earlier models

- Let's see how the inclusion of the day of the week impacts the quality of our predictions.

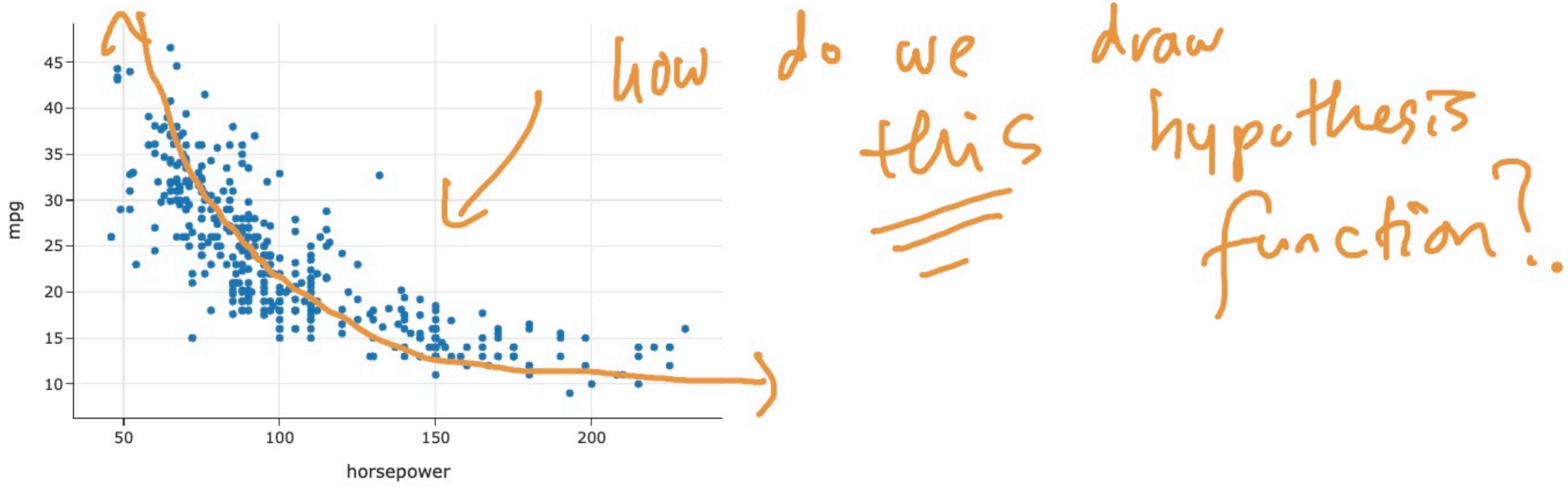
In [26]:
```python
mse_dict['departure_hour + day_of_month + ohe day'] = mean_squared_error(
    df['minutes'],
    model_with_ohe.predict(X_for_ohe)
)
pd.Series(mse_dict).plot(kind='barh', title='Mean Squared Error')
```



$$\text{var}(y) = \frac{\sum (y_i - \bar{y})^2}{n}$$

The relationship between `'horsepower'` and `'mpg'`

```
In [30]: px.scatter(mpg, x='horsepower', y='mpg')
```



- It appears that there is a negative association between `'horsepower'` and `'mpg'`, though it's not quite linear.

Linear in the parameters

$$\sum w \cdot \boxed{x}$$

- **Using linear regression**, we can fit hypothesis functions like:

$$H(x_i) = w_0 + w_1 x_i + w_2 x_i^2 \qquad H(\vec{x}_i) = w_1 e^{-x_i^{(1)^2}} + w_2 \cos(x_i^{(2)} + \pi) + w_3 \frac{\log 2x_i^{(3)}}{x_i^{(2)}}$$

This includes all polynomials, for example. These are all **linear combinations of (just) features**.

$$\text{Aug}(\vec{x}_i) = \begin{bmatrix} 1 \\ x_i \\ x_i^2 \end{bmatrix} \qquad \vec{x}_i = \begin{bmatrix} e^{-x_i^{(1)^2}} \\ \cos(x_i^{(2)} + \pi) \\ \dfrac{\log 2x_i^{(3)}}{x_i^{(2)}} \end{bmatrix}$$

Linear in the parameters

① linear in the parameters

② linear combinations of features

- **Using linear regression**, we can fit hypothesis functions like:

$$H(x_i) = w_0 + w_1 x_i + w_2 x_i^2 \qquad H(\vec{x}_i) = w_1 e^{-{x_i^{(1)}}^2} + w_2 \cos(x_i^{(2)} + \pi) + w_3 \frac{\log 2x_i^{(3)}}{x_i^{(2)}}$$

$$= \sum w_j \cdot \boxed{x}$$

use linear regression

This includes all polynomials, for example. These are all **linear combinations of (just) features**.

- For any of the above examples, we **could** express our model as $\vec{w} \cdot \text{Aug}(\vec{x}_i)$, for some carefully chosen feature vector $\vec{x}_i$,

and that's all that `LinearRegression` in `sklearn` needs.

What we put in the `X` argument to `model.fit` is up to us!

- Using linear regression, we **can't** fit hypothesis functions like:

$$H(x_i) = w_0 + e^{w_1 x_i} \qquad H(\vec{x}_i) = w_0 + \sin(w_1 x_i^{(1)} + w_2 x_i^{(2)})$$

These are **not** linear combinations of just features.

$$H(x_i) = a + b \sin(wx - c)$$

How do we fit hypothesis functions that aren't linear in the parameters?

- Suppose we want to fit the hypothesis function:

$$H(x_i) = w_0 e^{w_1 x_i}$$

- This is **not** linear in terms of w_0 and w_1, so our results for linear regression don't apply.

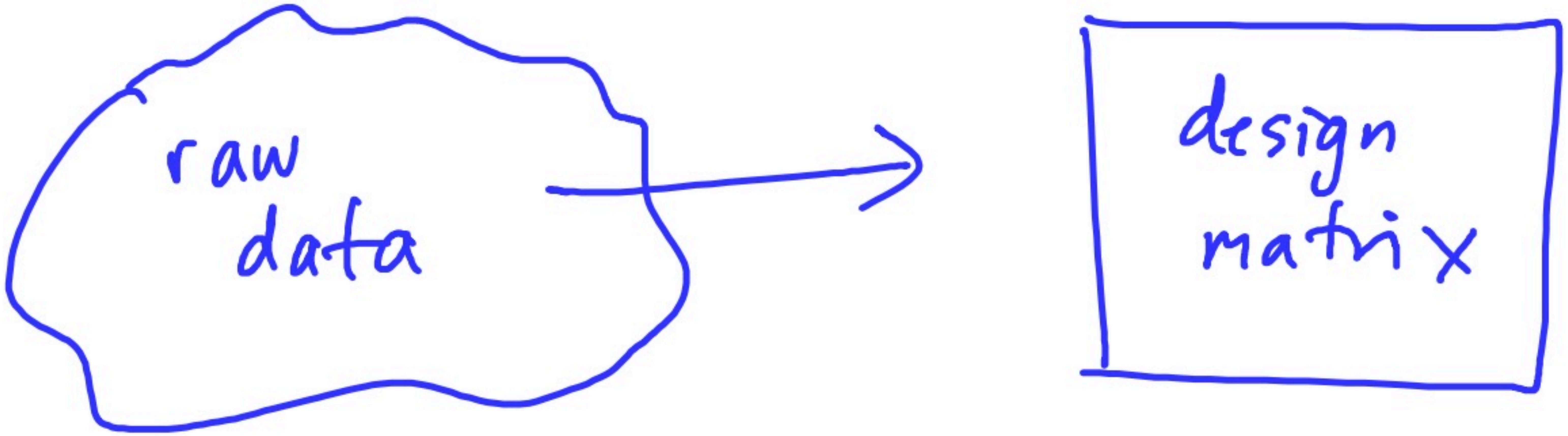
- **Possible solution**: Try to transform the above equation so that it **is** linear in some other parameters, by applying an operation to both sides.

- See the attached Reference Slide for more details.

localhost

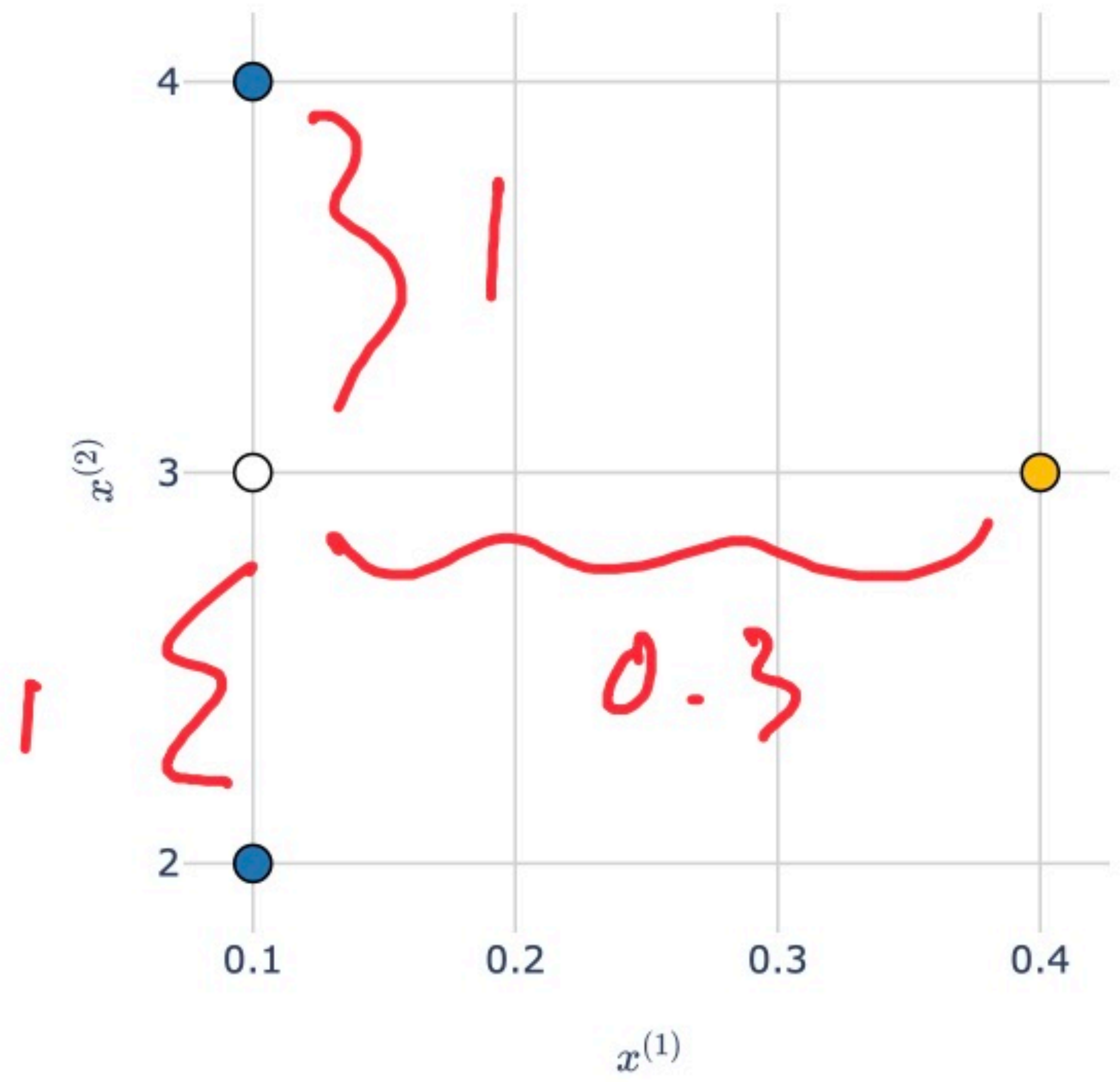
`preprocessing` and `linear_models`

- For the **feature engineering** step of the modeling pipeline, we will use `sklearn`'s `preprocessing` module.



- For the **model creation** step of the modeling pipeline, we will use `sklearn`'s `linear_model` module, as we've already seen. `linear_model.LinearRegression` is an example of an **estimator** class.

- Consider the white point in the scatter plot below.



- Which class is it more "similar" to – **blue** or **orange**?

- Intuitively, the answer may be **blue**, but take a close look at the scale of the axes! The **orange** point is much closer to the white point than the **blue** points are.

Standardization

- When we standardize two or more features, we bring them to the **same scale**.

- Recall: to standardize a feature $x_1, x_2, \ldots, x_n$, we use the formula:

$$z(x_i) = \frac{x_i - \bar{x}}{\sigma_x}$$

same as z-score!

- Example: 1, 7, 7, 9.
  - Mean: $\frac{1+7+7+9}{4} = \frac{24}{4} = 6$.
  - Standard deviation:

$$\text{SD} = \sqrt{\frac{1}{4}\left((1-6)^2 + (7-6)^2 + (7-6)^2 + (9-6)^2\right)} = \sqrt{\frac{1}{4} \cdot 36} = 3$$

  - Standardized data:

$1 - 6$ $\boxed{5}$ $7 - 6$ $\boxed{1}$ $\boxed{1}$ $9 - 6$

Out[62]: array([[0.85, -0.47, 0.51, 1.05, -0.36]])

```
In [63]: stdscaler.transform(sales.iloc[:, 1:].tail(5))
```

```
Out[63]: array([[-1.13, -1.31, -1.35, -1.6 ,  0.89],
                [ 0.14,  0.39,  0.4 ,  0.32, -0.36],
                [ 0.09, -0.03,  0.46,  0.36, -0.57],
                [ 0.9 ,  1.08,  1.05,  1.19, -1.61],
                [ 2.67,  0.69, -0.3 ,  0.46,  0.05]])
```

used the entire sales dataset to compute the mean and SD of each column

- If needed, the `fit_transform` method will fit the transformer and then transform the data in one go.

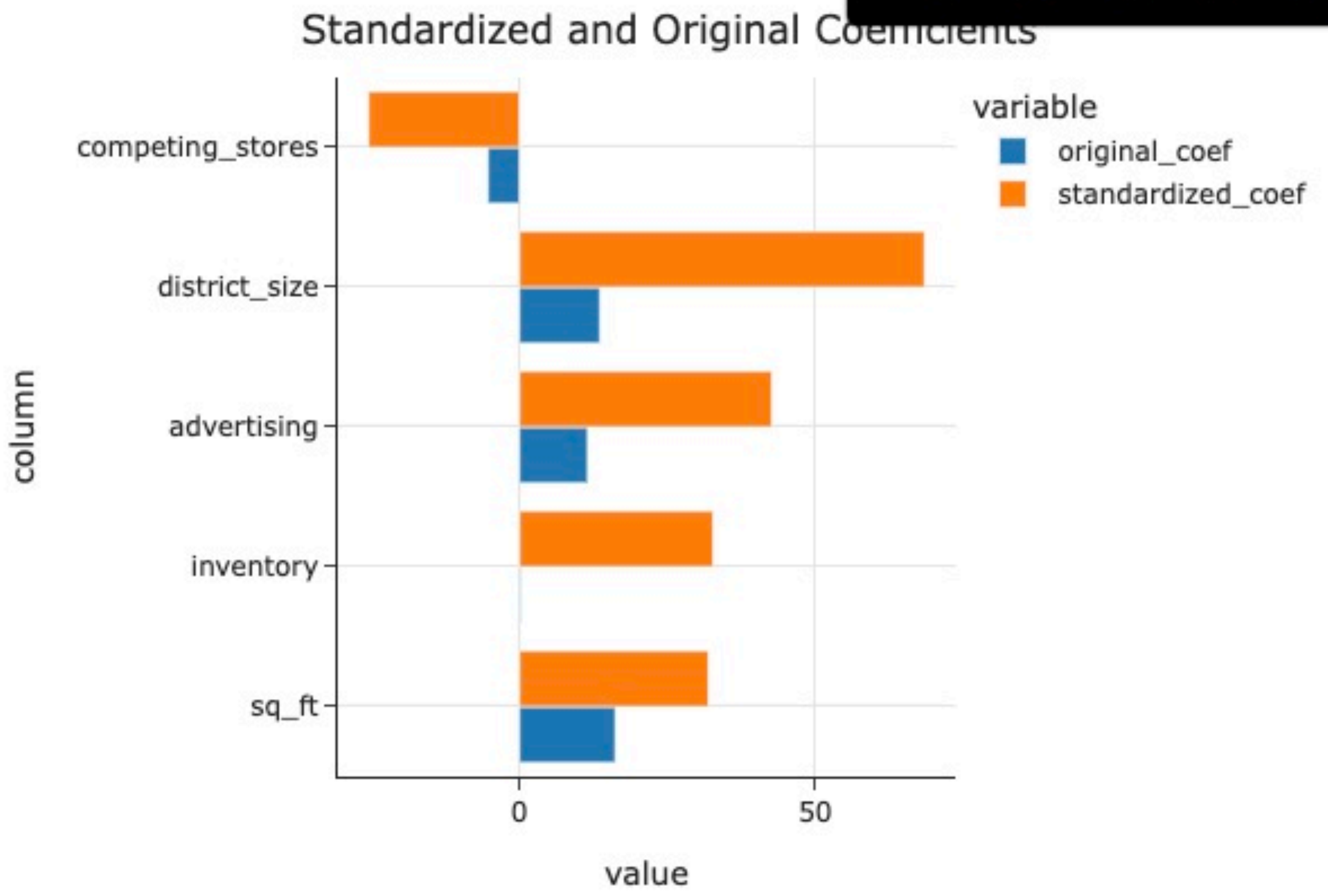
```
In [64]: new_scaler = StandardScaler()
```

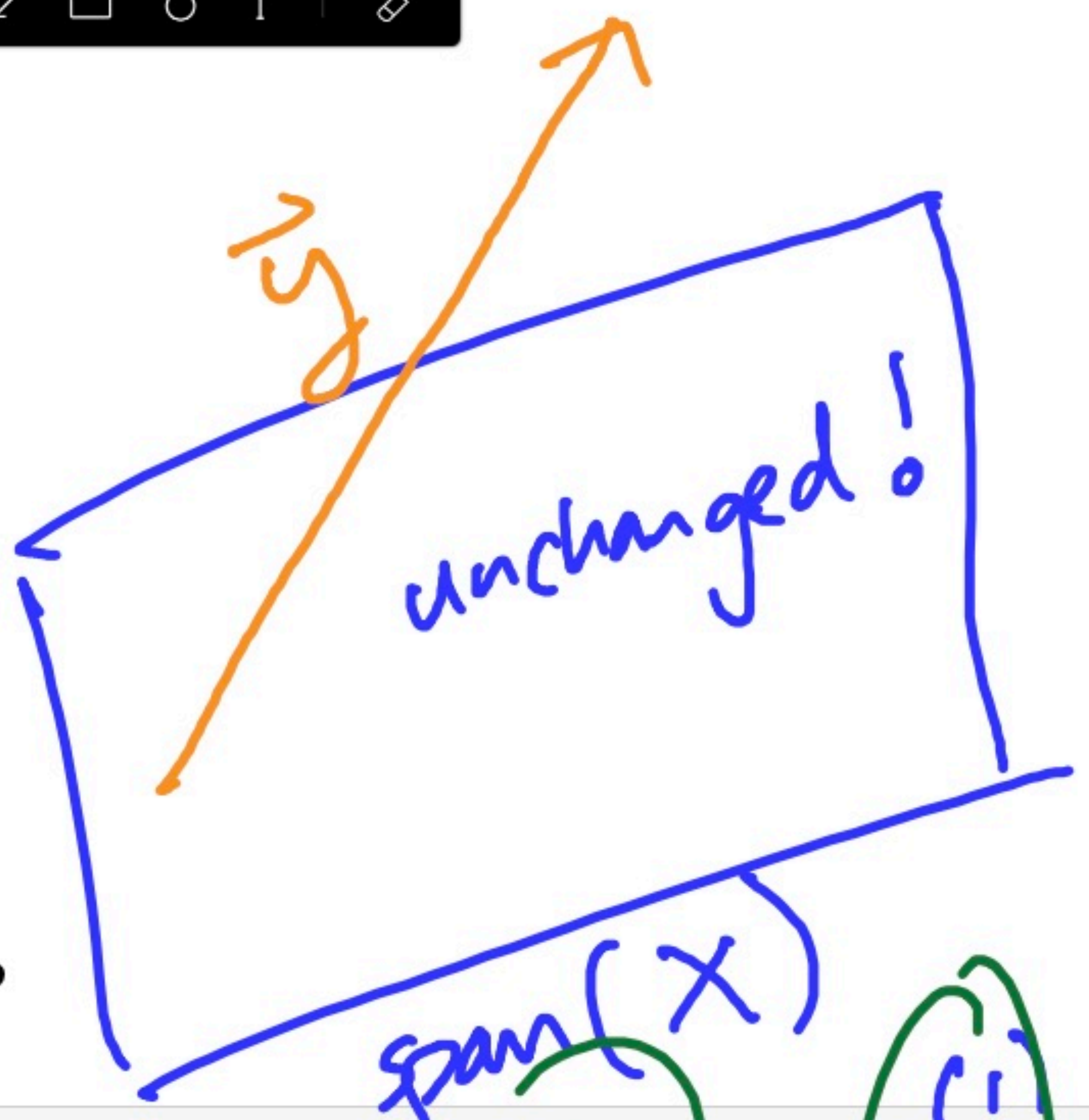
```
In [65]: new_scaler.fit_transform(sales.iloc[:, 1:].tail(5))
```

```
Out[65]: array([[-1.33, -1.79, -1.71, -1.88,  1.48],
                [-0.32,  0.28,  0.43,  0.19, -0.05],
                [-0.36, -0.24,  0.49,  0.23, -0.31],
                [ 0.29,  1.11,  1.22,  1.13, -1.58],
                [ 1.71,  0.64, -0.43,  0.34,  0.46]])
```

only ever saw the last 5 rows.

- Why are the values above different from the values in `stdscaler.transform(sales.iloc[:, 1:].tail(5))`?





- Did the performance of the resulting model change?

```
In [77]: mean_squared_error(sales.iloc[:, 0],
                            sales_model.predict(sales.iloc[:, 1:]))
```

```
Out[77]: 242.27445717154964
```

```
In [78]: mean_squared_error(sales.iloc[:, 0],
                            sales_model_std.predict(stdscaler.transform(sales.iloc[:, 1:])))
```

```
Out[78]: 242.27445717154956
```

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_1^{(2)} \\ 1 & x_2^{(1)} & x_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & & x_n^{(2)} \end{bmatrix}$$

span of cols of X