The general problem

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

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

e.g. predicting commute time given
- departure hour
- day of month

$\rightarrow \vec{x}_5 = \begin{bmatrix} 8.5 \\ 13 \end{bmatrix}$

- 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:

mean squared error

$$\begin{aligned} R_{\text{sq}}(\vec{w}) &= \frac{1}{n} \sum_{i=1}^{n} (y_i - H(\vec{x}_i))^2 \\ &= \frac{1}{n} \sum_{i=1}^{n} \left(y_i - (w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} + \ldots + w_d x_i^{(d)}) \right)^2 \\ &= \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \text{Aug}(\vec{x}_i) \cdot \vec{w} \right)^2 \\ &= \frac{1}{n} \|\vec{y} - X\vec{w}\|^2 \end{aligned}$$

X: "design matrix"

The general solution

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

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

$n \times d+1$

$\vec{h}$, which contains predictions, is of the form $X\vec{w}$, which is a linear combination of X's columns.

$$\begin{bmatrix} 1 & x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(d)} \end{bmatrix} \quad \begin{bmatrix} \text{Aug}(\vec{x_n})^T \end{bmatrix}$$

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

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

solution: orthogonal projection of $\vec{y}$ onto span(X)

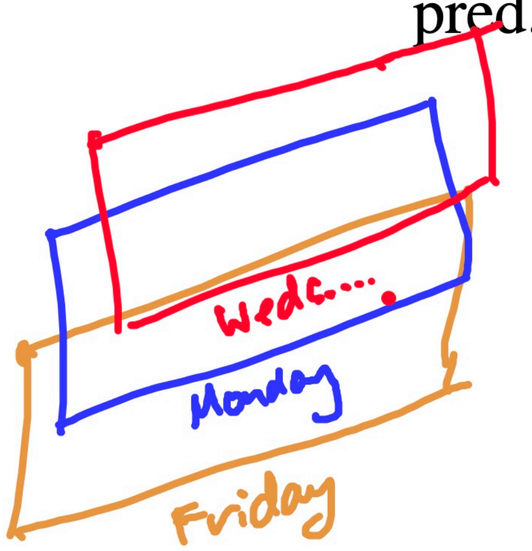
observe: sum of each row is always exactly 1! important

day	day == Mon	day == Tue	day == Wed	day == Thu	day == Fri
Mon	1	0	0	0	0
Tue	0	1	0	0	0
Mon	1	0	0	0	0
...	...	...	...	...	...
Mon	1	0	0	0	0
Tue	0	1	0	0	0
Thu	0	0	0	1	0

Visualizing our latest model

- Our trained linear model to predict commute time given `'departure_hour'`, `'day_of_month'`, and `'day'` (Mon, Tue, Wed, or Thu) is:

$$\begin{aligned}\text{pred. commute time}_i = 134 &- 8.42 \cdot \text{departure hour}_i \\ &- 0.03 \cdot \text{day of month}_i \\ &+ 5.09 \cdot \text{day}_i == \text{Mon} \\ &+ 16.38 \cdot \text{day}_i == \text{Tue} \\ &+ 5.12 \cdot \text{day}_i == \text{Wed} \\ &+ 11.5 \cdot \text{day}_i == \text{Thu}\end{aligned}$$



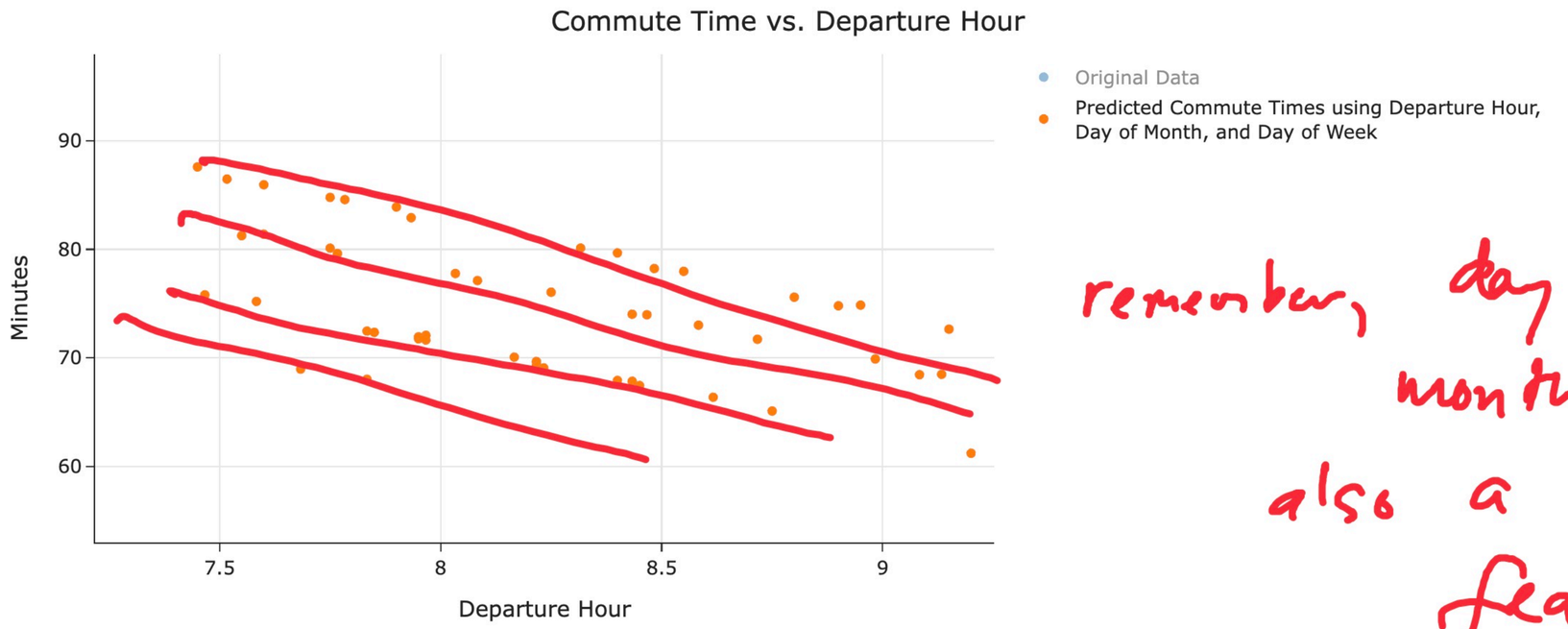
- Since we have 6 features here, we'd need 7 dimensions to graph our model.

- But, as we see in Homework 7, Question 5, our model is really a collection of **five parallel planes** in 3D, all with slightly different z-intercepts!

- If we want to visualize in 2D, we need to pick a single feature to place on the x-axis.

In [16]:

```
fig = go.Figure()
fig.add_trace(go.Scatter(x=df['departure_hour'], y=df['minutes'],
                         mode='markers', name='Original Data'))
fig.add_trace(go.Scatter(x=df['departure_hour'], y=model_with_ohe.predict(X_for_ohe),
                         mode='markers', name='Predicted Commute Times using Departure Hour, <br>Da
fig.update_layout(showlegend=True, title='Commute Time vs. Departure Hour',
                  xaxis_title='Departure Hour', yaxis_title='Minutes', width=1000)
```



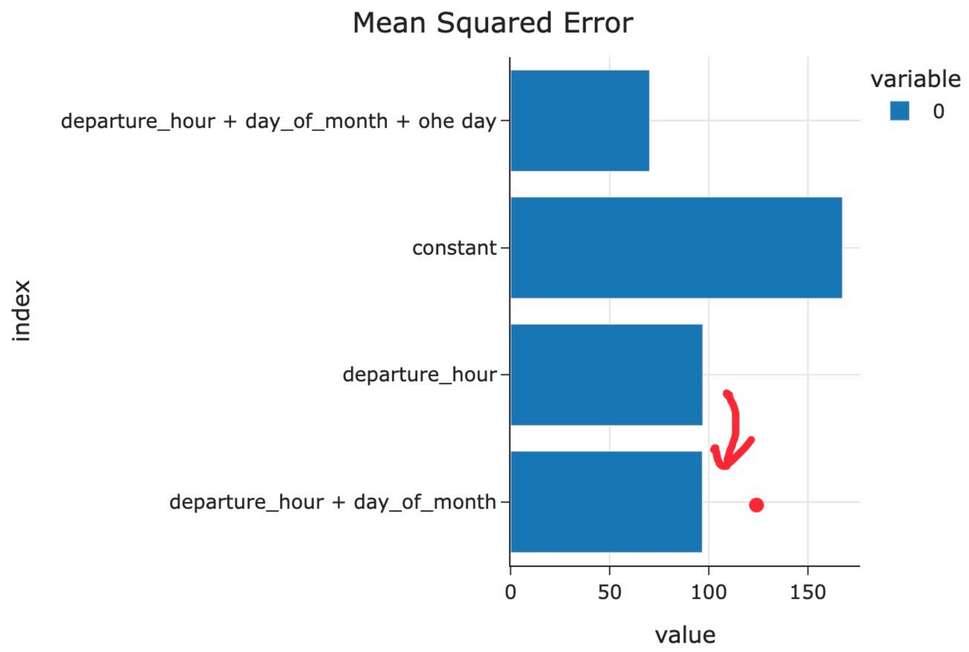
- Despite being a linear model, why **doesn't** this model **look** like a straight line?

?

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.

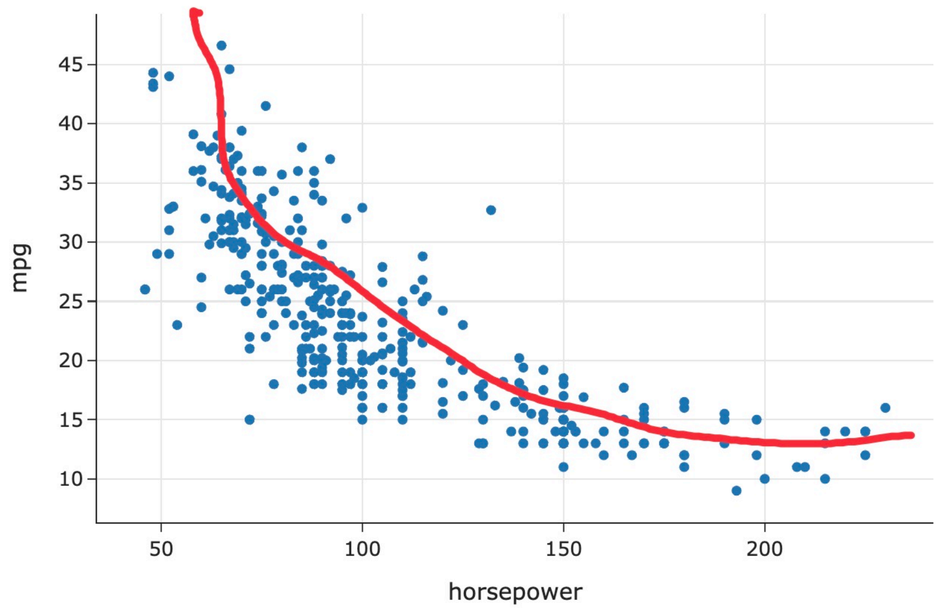
In [21]:
```
mse_dict['departure_hour + day_of_month + ohe day'] = mean_squared_error(
    df['minutes'],
    model_with_ohe.predict(X_for_ohe)
)
pd.Series(mse_dict).plot(kind='barh', title='Mean Squared Error')
```



day of month
kind of useless,
but
day of week is
very useful!

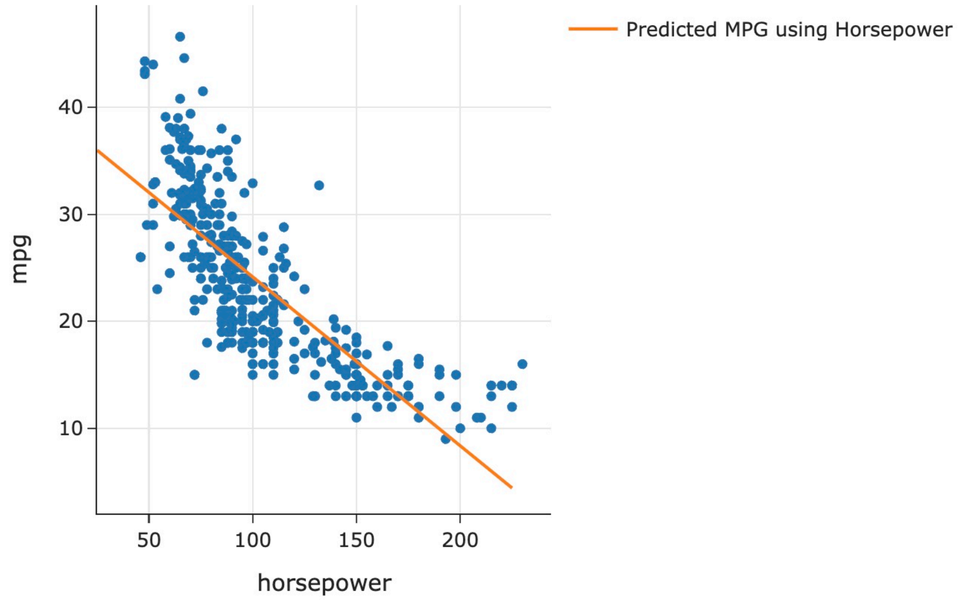
The relationship between `'horsepower'` and `'mpg'`

In [25]:
```
1 px.scatter(mpg, x='horsepower', y='mpg')
```



decay.

```
    x=hp_points['horsepower'],
    y=car_model.predict(hp_points),
    mode='lines',
    name='Predicted MPG using Horsepower'
))
```

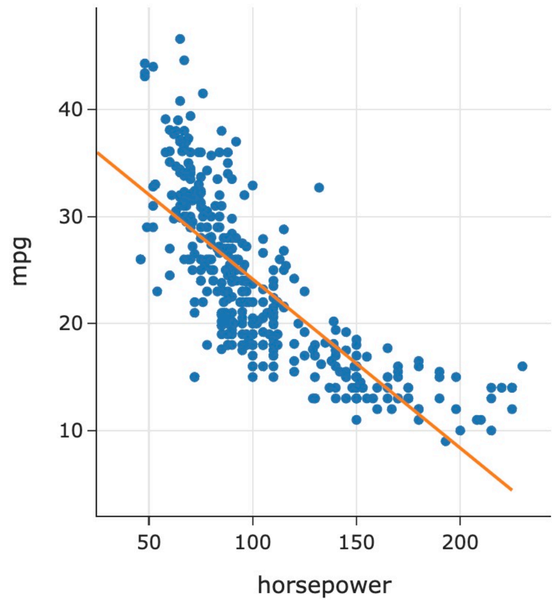


units: mpg^2

- Our regression line doesn't capture the curvature in the relationship between `'horsepower'` and `'mpg'`.

In [28]:
```
# As a baseline:
mean_squared_error(mpg['mpg'], car_model.predict(mpg[['horsepower']]))
```

Out[28]: 23.943662938603108



RMSE ≠ MAE

$\sqrt{\frac{1}{n}\Sigma(y_i - H(x_i))^2} \neq \frac{1}{n}\Sigma|y_i - H(x_i)|$

RMSE : mpg

- Our regression line doesn't capture the curvature in the relationship between `'horsepower'` and `'mpg'`.

In [29]:
```
# root mean squared error
mean_squared_error(mpg['mpg'], car_model.predict(mpg[['horsepower']])) ** 0.5
```

Out[29]: 4.893226230065713

mpg²

In [28]:
```
# As a baseline:
mean_squared_error(mpg['mpg'], car_model.predict(mpg[['horsepower']]))
```

Out[28]: 23.943662938603108

Linear in the parameters

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

$$H(x_i) = w_0 + w_1 x_i + w_2 x_i^2$$

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

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

- 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!

we can be creative with $\vec{x}_i$

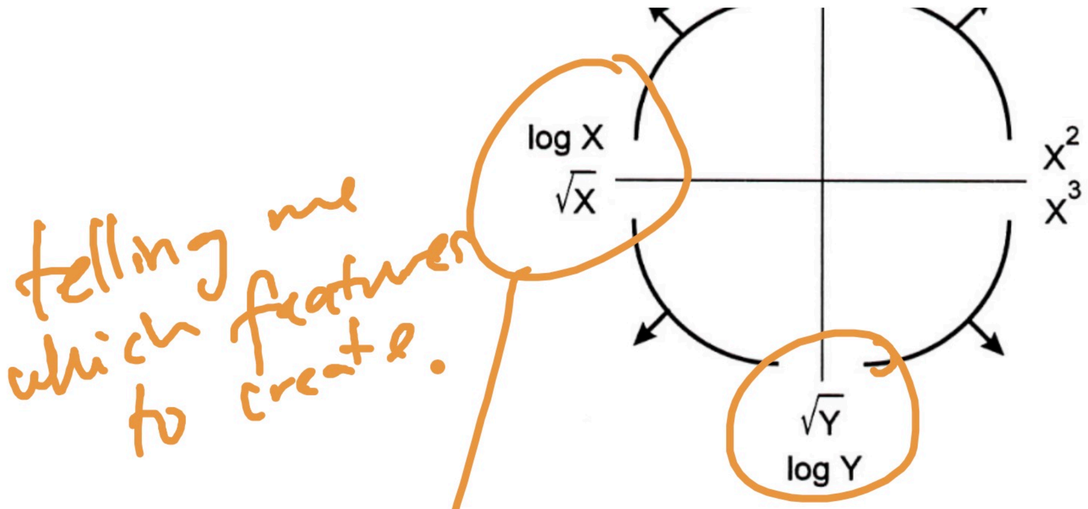
- Using linear regression, we **can't** fit hypothesis functions like:

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

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

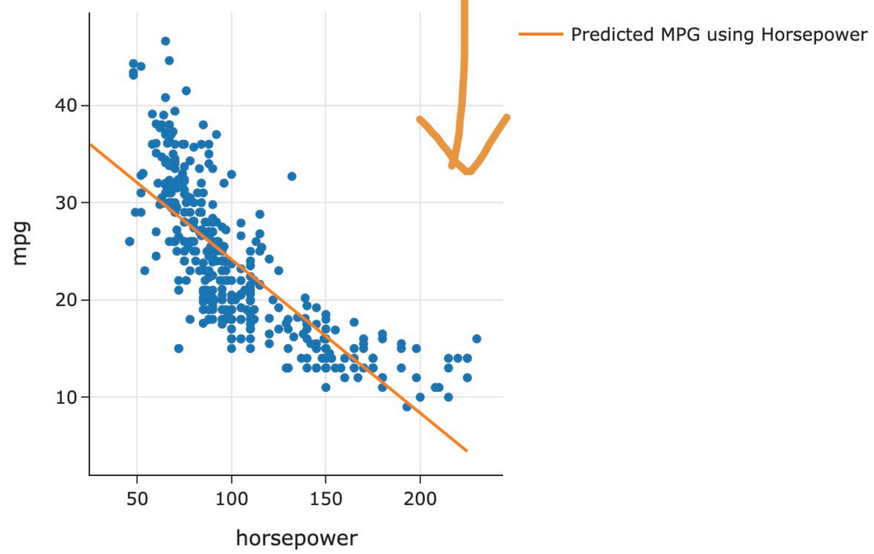
doesn't work!

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

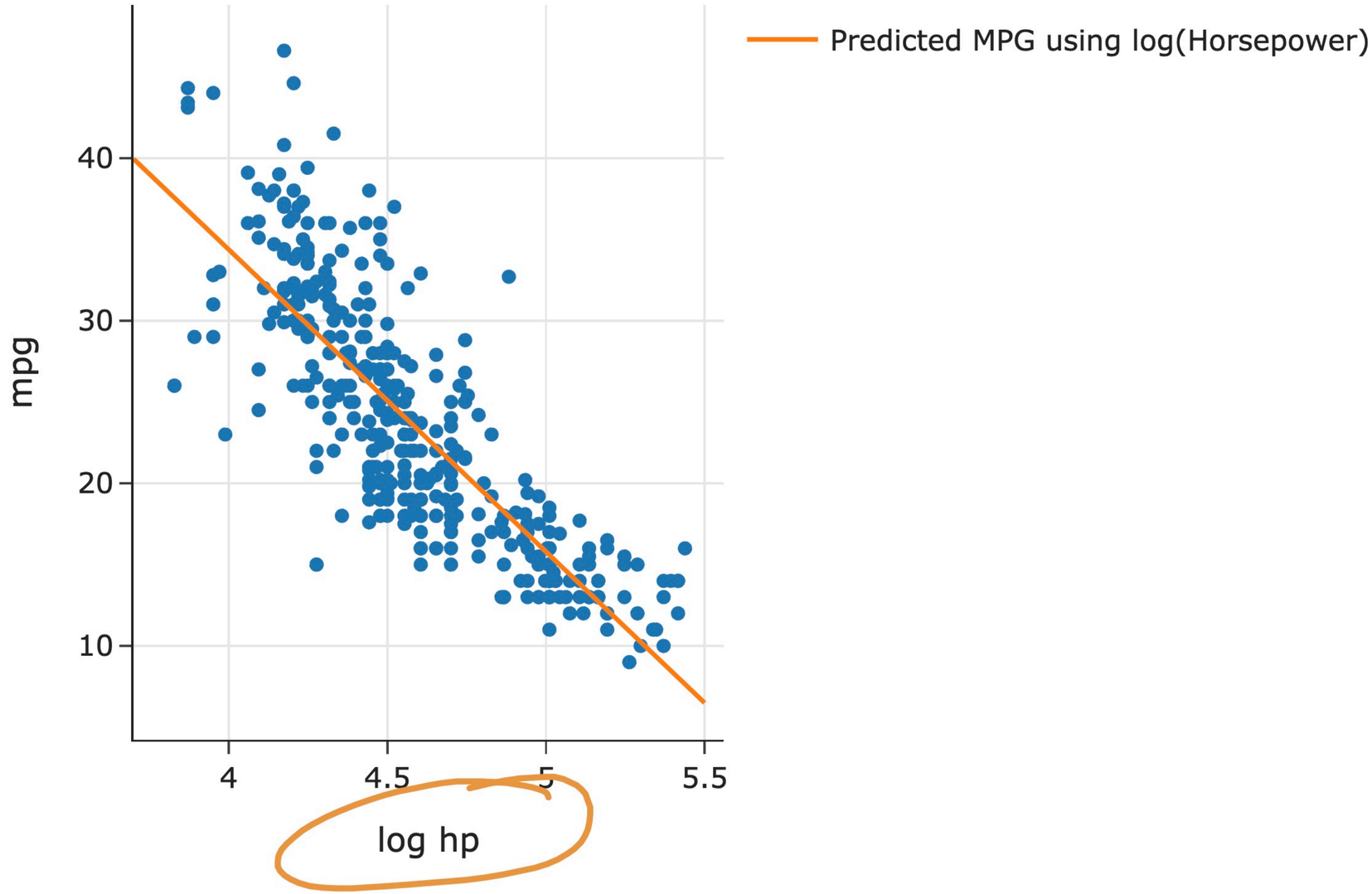


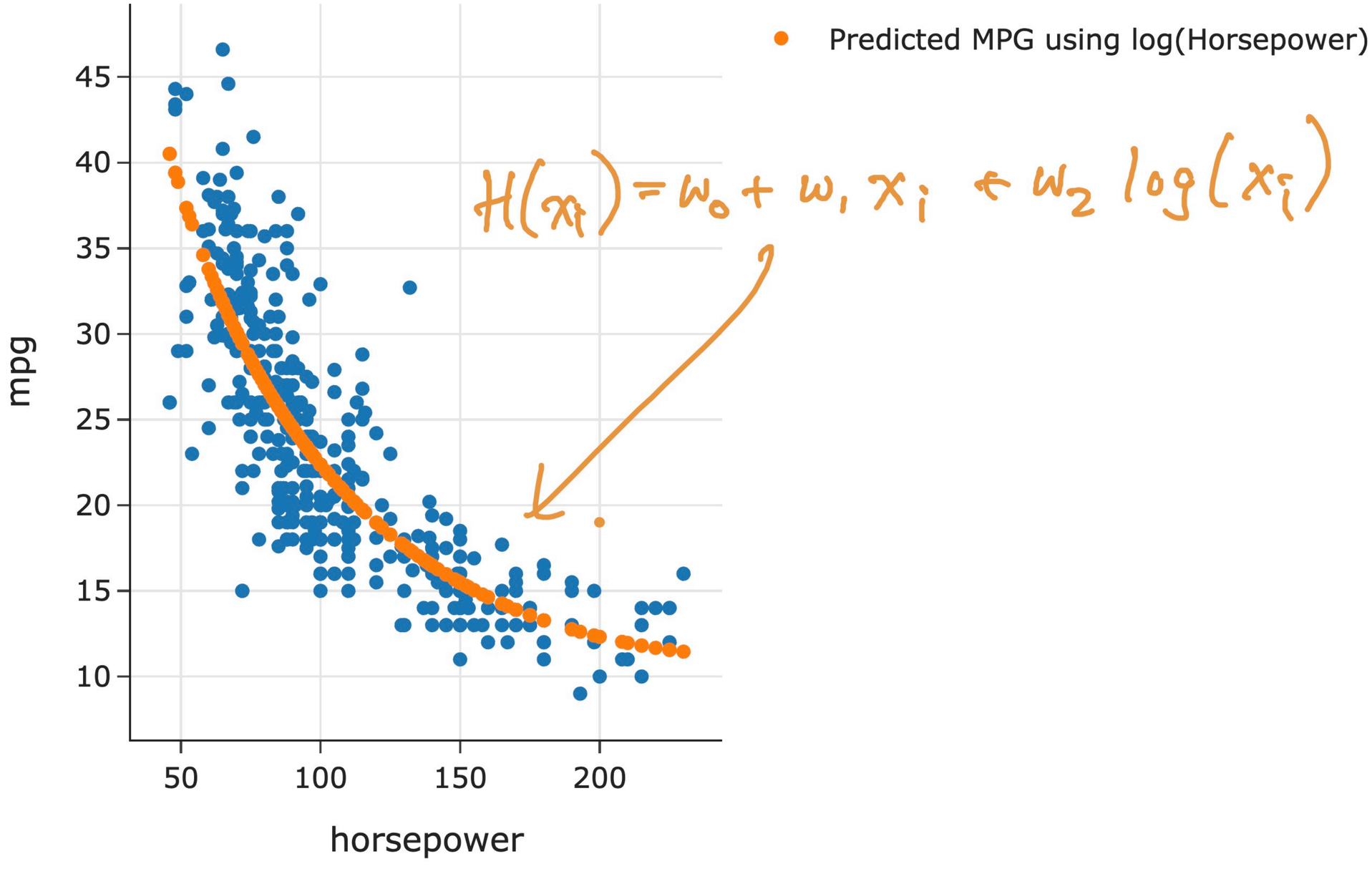
- **Why**? We're working with linear models. The more linear our data looks in terms of its features, the better we'll able to model the data.

In [30]:
```
1 fig
```



```
))
```





Question 🤔 (Answer at practicaldsc.org/q)

Which hypothesis function is **not** linear in the parameters?

- A. $H(\vec{x}_i) = w_1(x_i^{(1)} x_i^{(2)}) + \frac{w_2}{x_i^{(1)}} \sin\left(x_i^{(2)}\right)$
- B. $H(\vec{x}_i) = 2^{w_1} x_i^{(1)}$
- C. $H(\vec{x}_i) = \vec{w} \cdot \text{Aug}(\vec{x}_i)$
- D. $H(\vec{x}_i) = w_1 \cos(x_i^{(1)}) + w_2 2^{x_i^{(2)}} \log x_i^{(3)}$
- ~~E. More than one of the above.~~

$\beta_1 = 2^{w_1}$, $H(x_i) = \beta_1 x_i^{(1)}$

transformation.

$\sum w_j \square$

$$\vec{\beta} = (X^T X)^{-1} X^T \vec{z}$$

β_0, β_1

$$\vec{z} = \begin{bmatrix} \log y_1 \\ \log y_2 \\ \vdots \\ \log y_n \end{bmatrix}$$

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

log of BS

$$\log H(x_i) = \log(w_0) + w_1 x_i$$

define: $z_i = \log y_i$ $\quad T(x_i) = \log H(x_i)$

$\beta_0 = \log w_0$

$\beta_1 = w_1$

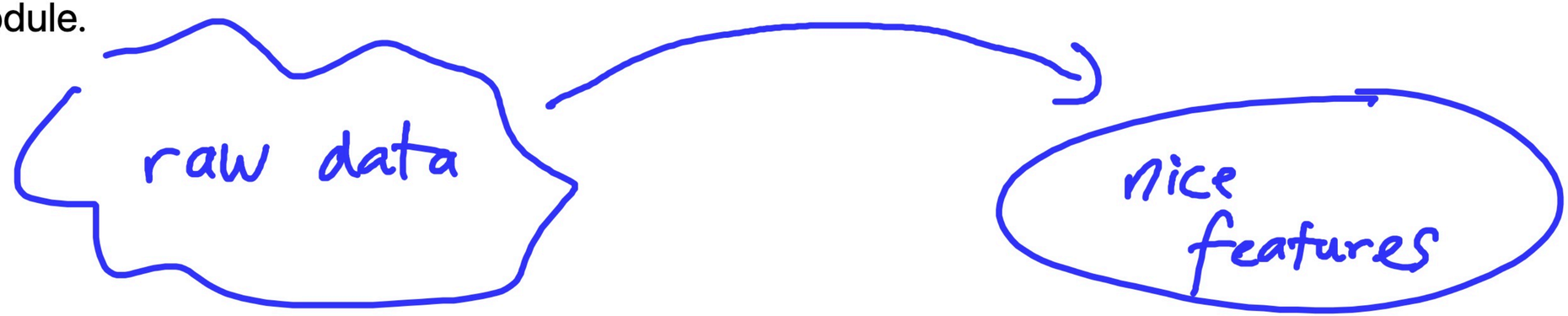
$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$$

Then:

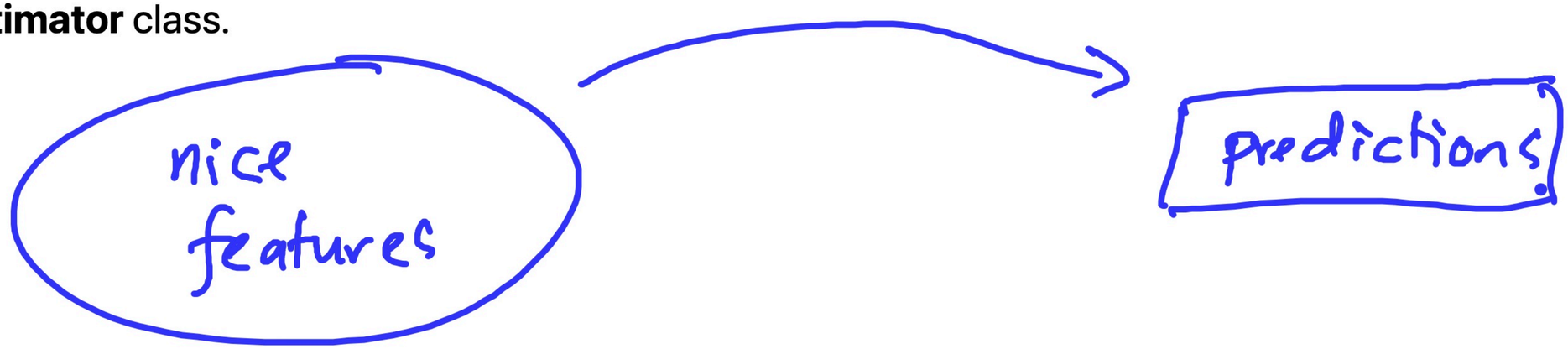
$$T(x_i) = \beta_0 + \beta_1 x_i$$

`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.



In [44]:
```
1 sales = pd.read_csv('data/sales.csv')
2 sales.head()
```

Out[44]:

	net_sales	sq_ft	inventory	advertising	district_size	competing_stores
0	231.0	3.0	294	8.2	8.2	11
1	156.0	2.2	232	6.9	4.1	12
2	10.0	0.5	149	3.0	4.3	15
3	519.0	5.5	600	12.0	16.1	1
4	437.0	4.4	567	10.6	14.1	5

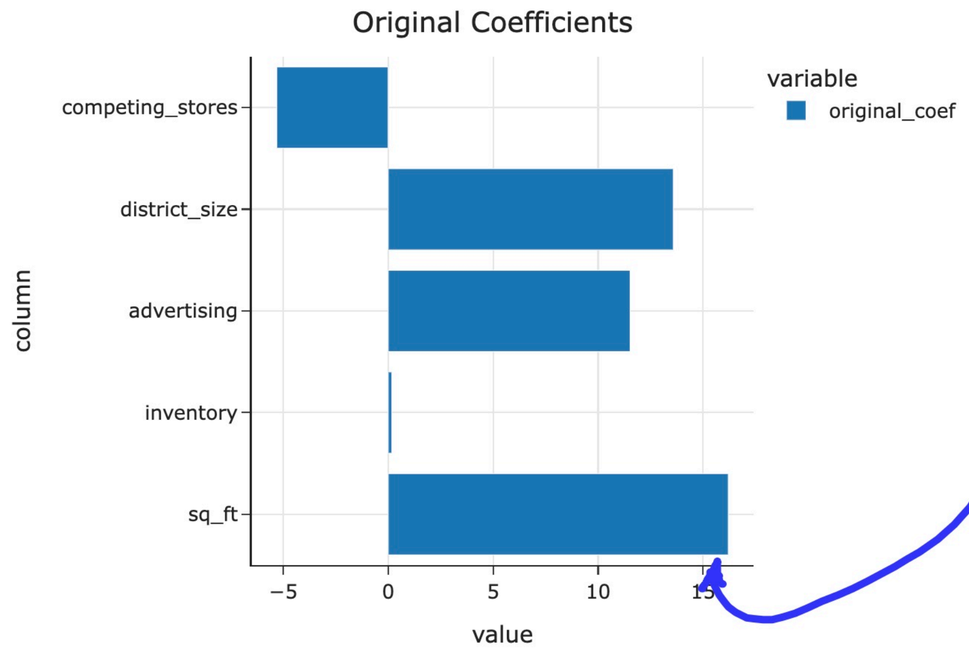
- For each of 26 stores, we have:
  - net sales,
  - square feet,
  - inventory,
  - advertising expenditure,
  - district size, and
  - number of competing stores.

- Our goal is to predict `'net_sales'` as a function of other features.

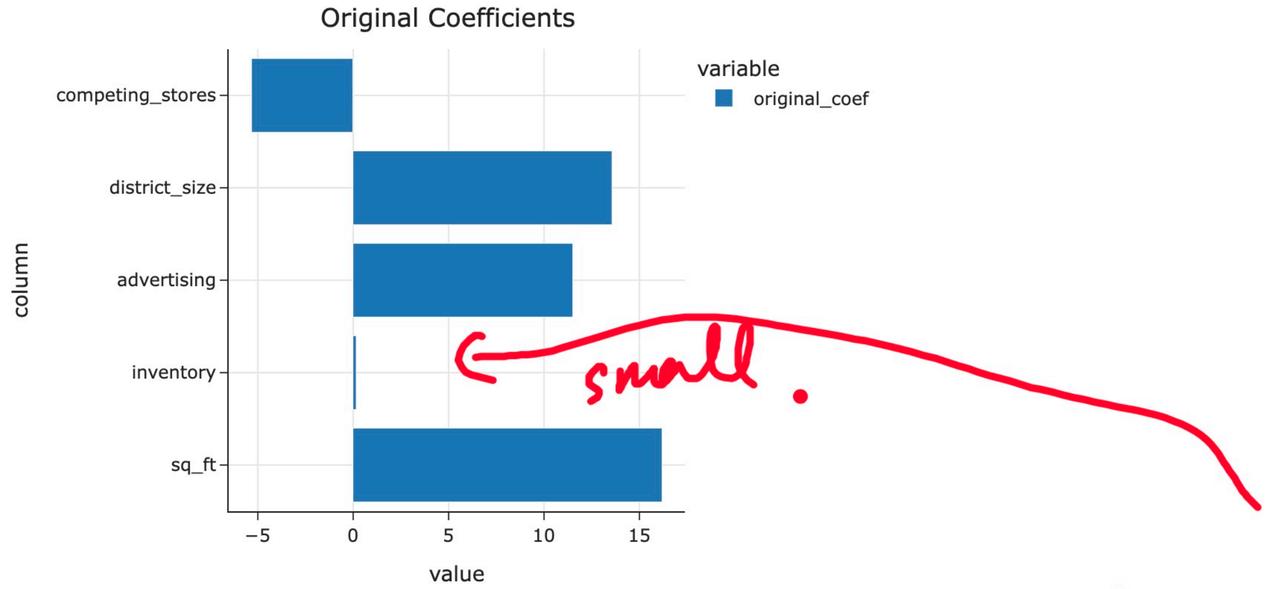
In [47]:
```
1 sales_model.coef_
```

Out[47]:
```
array([16.2 ,  0.17, 11.53, 13.58, -5.31])
```

In [48]:
```
1 coefs = pd.DataFrame().assign(
2     column=sales.columns[1:],
3     original_coef=sales_model.coef_,
4 ).set_index('column')
5 coefs.plot(kind='barh', title='Original Coefficients')
```



plotting the W value for each feature.



small.

numbers are very big!!!

- What do you notice?

In [49]:
```
1 sales.iloc[:, 1:]
```

Out[49]:

	sq_ft	inventory	advertising	district_size	competing_stores
0	3.0	294	8.2	8.2	11
1	2.2	232	6.9	4.1	12
2	0.5	149	3.0	4.3	15
...	...	...	...	...	...
24	8.5	382	9.8	11.5	5
25	5.1	590	12.0	15.7	0
26	8.6	517	7.0	12.0	8

27 rows × 5 columns

much smaller scale

features are on very different scales ∴.

small

numbers are very big!!!

- What do you notice?

In [49]:
```
1 sales.iloc[:, 1:]
```

Out[49]:

	sq_ft	inventory	advertising	district_size	competing_stores
0	3.0	294	8.2	8.2	11
1	2.2	232	6.9	4.1	12
2	0.5	149	3.0	4.3	15
...	...	...	...	...	...
24	8.5	382	9.8	11.5	5
25	5.1	590	12.0	15.7	0
26	8.6	517	7.0	12.0	8

2.94
2.32
1.49
⋮

much smaller scale

27 rows × 5 columns

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:

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

do this separately for each column..

- 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:

$$1 \mapsto \frac{1-6}{3} = \boxed{-\frac{5}{3}} \qquad 7 \mapsto \frac{7-6}{3} = \boxed{\frac{1}{3}} \qquad 7 \mapsto \boxed{\frac{1}{3}} \qquad 9 \mapsto \frac{9-6}{3} = \boxed{1}$$

Here, "fitting" the transformer involves computing and saving the mean and SD of each column.

In [53]:
```
1 stdscaler = StandardScaler()
```

In [55]:
```
1 # This is like saying "determine the mean and SD of each column in sales,
2 # other than the 'net_sales' column".
3 stdscaler.fit(sales.iloc[:, 1:])
```

Out[55]:
```
StandardScaler
StandardScaler()
```

uses mean and SD of columns it was fit on!

In [57]:
```
1 stdscaler.transform([[5, 300, 10, 15, 6]])
```

inventory

Out[57]:
```
array([[ 0.85, -0.47,  0.51,  1.05, -0.36]])
```

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

300 is 0.47 SDs BELOW the average inventory in sales.

In [53]:
```
stdscaler = StandardScaler()
```

In [55]:
```
# This is like saying "determine the mean and SD of each column in sales,
# other than the 'net_sales' column".
stdscaler.fit(sales.iloc[:, 1:])
```

Out[55]:

StandardScaler

StandardScaler()

In [57]:
```
stdscaler.transform([[5, 300, 10, 15, 6]])
```

Out[57]:
```
array([[ 0.85, -0.47,  0.51,  1.05, -0.36]])
```

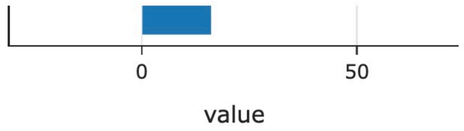
In [59]:
```
stdscaler.transform(sales.iloc[:, 1:].tail(5))
```

Out[59]:
```
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]])
```

In [60]:
```
new_scaler = StandardScaler()
new_scaler.fit(sales.iloc[:, 1:].tail(5))
new_scaler.transform(sales.iloc[:, 1:].tail(5))
```

Out[60]:
```
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]])
```

Why different? because different datasets were used in .fit!



- Did the performance of the resulting model change?

In [67]:
```
mean_squared_error(sales.iloc[:, 0],
                   sales_model.predict(sales.iloc[:, 1:]))
```

Out[67]: 242.27445717154964

In [69]:
```
sales_model.predict(sales.iloc[:, 1:])
```

Out[69]: array([228.54, 128.78, 28.57, ..., 347.13, 518.33, 411.92])

same preds!

In [68]:
```
mean_squared_error(sales.iloc[:, 0],
                   sales_model_std.predict(stdscaler.transform(sales.iloc[:, 1:])))
```

Out[68]: 242.27445717154956

In [70]:
```
sales_model_std.predict(stdscaler.transform(sales.iloc[:, 1:]))
```

Out[70]: array([228.54, 128.78, 28.57, ..., 347.13, 518.33, 411.92])

- **No!**

  The span of the design matrix did not change, so the predictions did not change. It's just the coefficients that changed.

$$X = \begin{bmatrix} | & | & | \\ | & \vec{x}^{(1)} & \vec{x}^{(2)} \\ | & | & | \end{bmatrix}$$

$$\vec{z}^{(1)} = \frac{\vec{x}^{(1)} - \text{mean}(\vec{x}^{(1)})}{SD(\vec{x}^{(1)})}$$

$$= \frac{\vec{x}^{(1)}}{SD(\vec{x}^{(1)})} - \frac{\text{mean}(\vec{x}^{(1)})}{SD(\vec{x}^{(1)})}$$

$$= \underbrace{\frac{1}{SD(\vec{x}^{(1)})}}_{a}\vec{x}^{(1)} - \underbrace{\frac{\text{mean}(\vec{x}^{(1)})}{SD(\vec{x}^{(1)})}}_{b}\vec{1}$$

$$= a\vec{x}^{(1)} + b\vec{1}$$

$\vec{z}^{(1)}$ is a lin. comb. of $\vec{1}$ & $\vec{x}^{(1)}$

$$\text{span}(\vec{1}, \vec{x}^{(1)}, \vec{x}^{(2)}) = \text{span}(\vec{1}, \vec{z}^{(1)}, \vec{z}^{(2)})$$

big picture