65 rows × 2 columns

- First, we need to import the relevant class from `sklearn.preprocessing`.

In [5]:
```
1 from sklearn.preprocessing import OneHotEncoder
```

- Like with `StandardScaler`, we need to instantiate **and fit** our `OneHotEncoder` instsance before it can transform anything.

In [6]:
```
1 ohe = OneHotEncoder()
```

In [8]:
```
1 ohe.get_feature_names_out()
```

Out[8]:
```
array(['day_Fri', 'day_Mon', 'day_Thu', 'day_Tue', 'day_Wed',
       'month_August', 'month_December', 'month_February',
       'month_January', 'month_July', 'month_June', 'month_March',
       'month_May', 'month_November', 'month_October', 'month_September'],
      dtype=object)
```

names of the new OHE-d columns

In [7]:
```
1 ohe.fit(df[['day', 'month']])
```

Out[7]:
```
OneHotEncoder
OneHotEncoder()
```

- w_0^* and w_1^* are shown below, along with the

  We call this the model's **training MSE**.

In [17]:
```
1 people_one_feat.intercept_, people_one_feat.coef_
```

Out[17]: (-82.57574306454093, array([3.08]))

pay attention!

In [18]:
```
1 from sklearn.metrics import mean_squared_error
2 mean_squared_error(y, people_one_feat.predict(X))
```

Out[18]: 101.58853248632849

Out[21]:

LinearRegression

LinearRegression()

$-2.32 \cdot 10^{11}$ pounds / inch

- What are w_0^*, w_1^*, w_2^*, and the model's MSE?

In [22]:
```
1 people_two_feat.intercept_, people_two_feat.coef_
```

Out[22]: (-82.59155502376602, array([-2.32e+11, 2.78e+12]))

In [23]:
```
1 mean_squared_error(y, people_two_feat.predict(X2))
```

Out[23]: 101.58844271417476

same as last slide!

$2.78 \cdot 10^{12}$ pounds / feet

Same predictions, UNINTERPRETABLE coefficients, $\vec{w}^*$

?

Redundant features

- Suppose in the first model, $w_0^* = -80$ and $w_1^* = 3$.

$$\text{predicted weight}_i = -80 + 3 \cdot \text{height in inches}_i$$

- In the second model, we have:

$$\text{predicted weight}_i = w_0^* + w_1^* \cdot \text{height in inches}_i + w_2^* \cdot \text{height in feet}_i$$

in the span of the old features!

- Suppose in the first model, $w_0^* = -80$ and $w_1^* = 3$.

$$\text{predicted weight}_i = -80 + 3 \cdot \text{height in inches}_i$$

- In the second model, we have:

$$\text{predicted weight}_i = w_0^* + w_1^* \cdot \text{height in inches}_i + w_2^* \cdot \text{height in feet}_i$$

- But, since $\text{height in feet}_i = \frac{\text{height in inches}_i}{12}$:

$$\begin{aligned}\text{predicted weight}_i &= w_0^* + w_1^* \cdot \text{height in inches}_i + w_2^* \cdot \text{height in feet}_i \\ &= w_0^* + w_1^* \cdot \text{height in inches}_i + w_2^* \cdot \left(\frac{\text{height in inches}_i}{12}\right) \\ &= w_0^* + \left(w_1^* + \frac{w_2^*}{12}\right) \cdot \text{height in inches}_i\end{aligned}$$

Infinitely many parameter choices

- **Issue**: There are an infinite number of w_1^* and w_2^* that satisfy $w_1^* + \frac{w_2^*}{12} = 3!$

Any solution to this, with $w_0^* = -80$, minimizes MSE

$$\text{predicted weight}_i = -80 + 5 \cdot \text{height in inches}_i - 24 \cdot \text{height in feet}_i$$

- **One hot encoding will result in multicollinearity unless you drop one of the one hot encoded features.**

- Suppose we have the following fitted model:
  For illustration, assume `'weekend'` was originally a categorical feature with two possible values, `'Yes'` or `'No'`.

$$H(\vec{x}_i) = 1 - 3 \cdot \text{departure hour}_i + 2 \cdot (\text{weekend}_i == \text{Yes}) - 2 \cdot (\text{weekend}_i == \text{No})$$

- This is equivalent to:

$$H(\vec{x}_i) = 10 - 3 \cdot \text{departure hour}_i - 7 \cdot (\text{weekend}_i == \text{Yes}) - 11 \cdot (\text{weekend}_i == \text{No})$$

- **One hot encoding will result in multicollinearity unless you drop one of the one hot encoded features.**

multicollinearity → infinitely many $\vec{w}$ that satisfy $X^TX\vec{w} = X^T\vec{y}$.

b.c. X^TX not invertible

- Suppose we have the following fitted model:

  For illustration, assume `'weekend'` was originally a categorical feature with two possible values, `'Yes'` or `'No'`.

important

$$H(\vec{x}_i) = 1 - 3 \cdot \text{departure hour}_i + 2 \cdot (\text{weekend}_i == \text{Yes}) - 2 \cdot (\text{weekend}_i == \text{No})$$

1 OR 0 / 0 OR 1

- This is equivalent to:

$$H(\vec{x}_i) = 10 - 3 \cdot \text{departure hour}_i - 7 \cdot (\text{weekend}_i == \text{Yes}) - 11 \cdot (\text{weekend}_i == \text{No})$$

$$X = \begin{bmatrix} 1 & dh_1 & 1 & 0 \\ 1 & dh_2 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & dh_n & 0 & 1 \end{bmatrix}$$

$\vec{1}$ $\vec{Yes}$ $\vec{No}$

if you keep both $\vec{Yes}$ and $\vec{No}$, X's columns are not linearly independent!

$\vec{No} = \vec{1} - \vec{Yes}$

X^TX not invertible

1	Tue	May
2	Mon	May
...	...	...
62	Mon	March
63	Tue	March
64	Thu	March

65 rows × 2 columns

- Let's try using `drop='first'` when instantiating a `OneHotEncoder`.

always do this (at least for Linear Regression)!

In [25]:
```
1 ohe_drop_one = OneHotEncoder(drop='first')
```

In [ ]:
```
1 ohe_drop_one.fit(df[['day', 'month']])
```

- Let's try using `drop='first'` when instantiating a `OneHotEncoder`.

In [25]:
```
1 ohe_drop_one = OneHotEncoder(drop='first')
```

In [26]:
```
1 ohe_drop_one.fit(df[['day', 'month']])
```

Out[26]:

OneHotEncoder

OneHotEncoder(drop='first')

- How many features did the resulting transformer create?

In [27]:
```
1 len(ohe_drop_one.get_feature_names_out())
```

Out[27]: 14

$(5-1)+(11-1)=14$

- Where did this number come from?

In [28]:
```
1 df['day'].nunique()
```

Out[28]: 5

In [29]:
```
1 df['month'].nunique()
```

Out[29]: 11

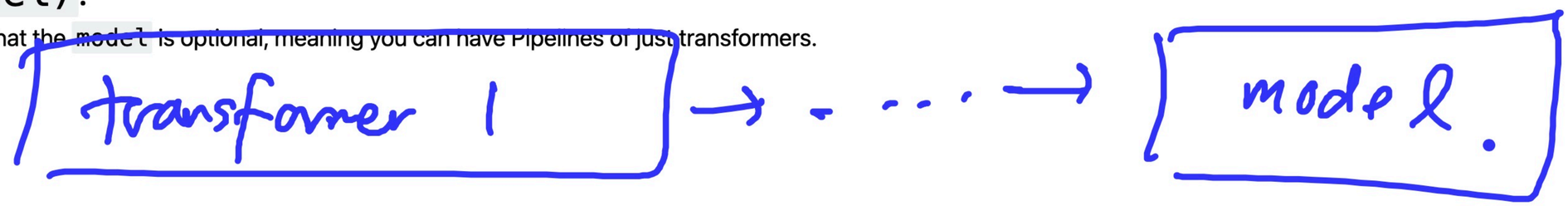
Pipelines in `sklearn`

- From `sklearn`'s documentation:

> *`Pipeline` allows you to sequentially apply a list of transformers to preprocess the data and, **if desired**, conclude the sequence with a final predictor for predictive modeling.*
>
> *Intermediate steps of the pipeline must be "transforms", that is, they must implement `fit` and `transform` methods. The final estimator only needs to implement `fit`.*

- General template: `pl = make_pipeline(transformer_1, transformer_2, ..., model)`.

Note that the `model` is optional, meaning you can have Pipelines of just transformers.



```
# Here, we're having that function return a new Series/DataFrame,
# depending on what's passed in to .tranform (experiment on your own).
from sklearn.pipeline import FunctionTransformer
WeekConverterTransformer = FunctionTransformer(
    lambda x: 'Week ' + ((x - 1) // 7 + 1).astype(str)
)
```

In [67]:
```
WeekConverterTransformer.transform(df[['day_of_month']])
```

Out[67]:

	day_of_month
0	Week 3
1	Week 3
2	Week 4
...	...
62	Week 1
63	Week 1
64	Week 1

65 rows × 1 columns

both are transformers

- We need to apply two consecutive transformations to `'day_of_month'`, which calls for a Pipeline.

In [ ]:
```
day_of_month_transformer = make_pipeline(
    WeekConverterTransformer, OneHotEncoder(drop='first')
)
day_of_month_transformer
```

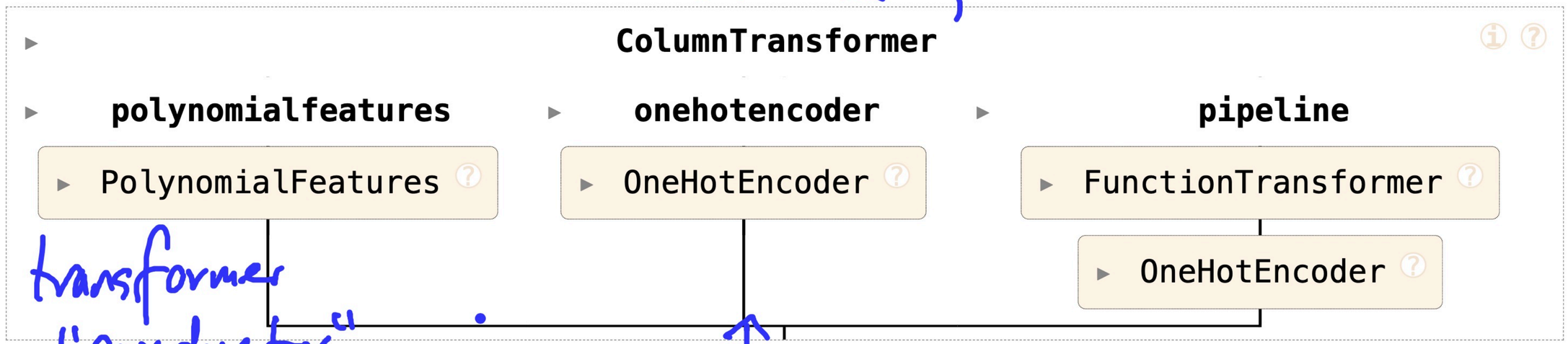
To specify which transformations to apply to which columns, create a `Colu`

In [71]:
```
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.preprocessing import PolynomialFeatures
```

In [72]:
```
preprocessing = make_column_transformer(
    (PolynomialFeatures(3, include_bias=False), ['departure_hour']),
    (OneHotEncoder(drop='first'), ['day', 'month']),
    (day_of_month_transformer, ['day_of_month']),
    remainder='drop'
)
preprocessing
```

(transformer instance, [columns to use])

Out[72]:



column transformer is a "conductor"

raw data

users can call .predict without having to create features manually!!!

In [78]:
```
1 model.predict(pd.DataFrame([{
2     'departure_hour': 8.5,
3     'day': 'Tue',
4     'month': 'June',
5     'day_of_month': 3
6 }]))
```

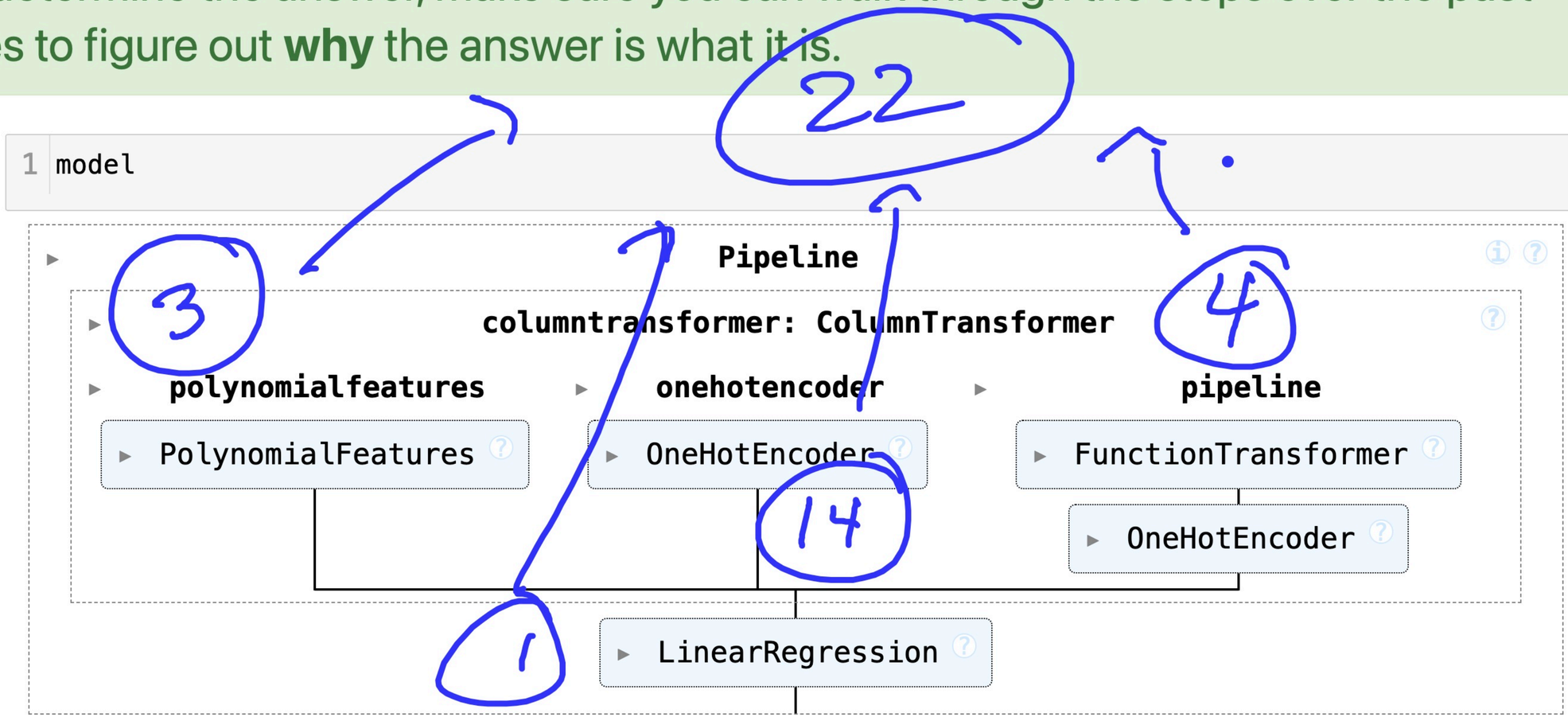
Out[78]: array([76.11])

Activity

How many columns does the final design matrix that `model` creates have? If you write code to determine the answer, make sure you can walk through the steps over the past few slides to figure out **why** the answer is what it is.

```
In [79]: 1 model
```

Out[79]:

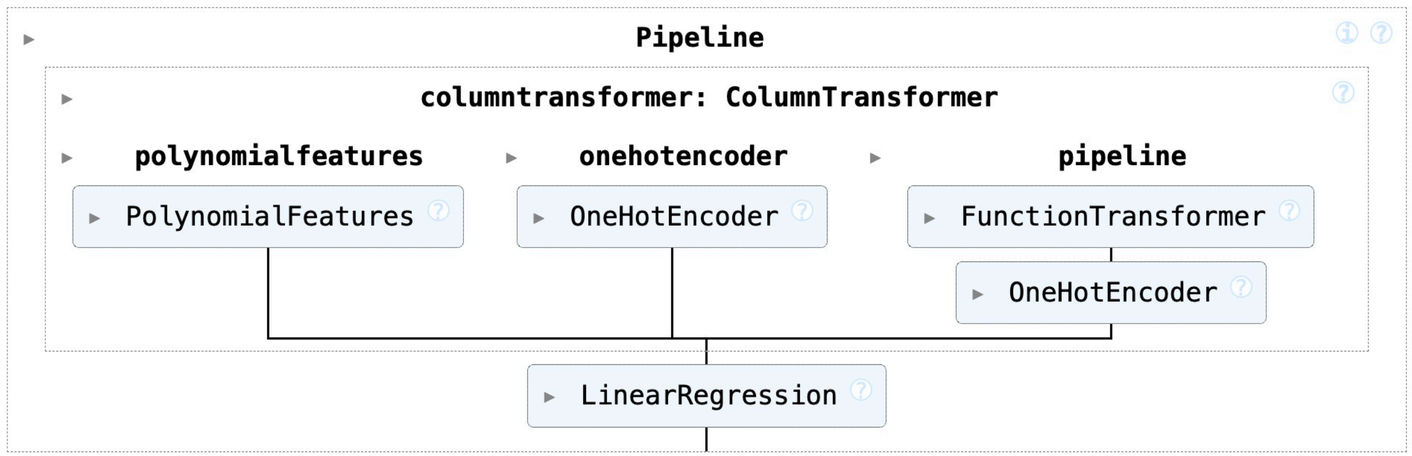


Activity

How many columns does the final design matrix that `model` creates have? If you write code to determine the answer, make sure you can walk through the steps over the past few slides to figure out **why** the answer is what it is.

In [85]:
```
1 model
```

Out[85]:



In [82]:
```
1 model.named_steps['linearregression']
```

Out[82]:

LinearRegression
LinearRegression()

In [84]:
```
1 model[-1]
```

Out[84]:

LinearRegression
LinearRegression()

In [88]:
```
1 len(model[-1].coef_) + 1
```

Out[88]: 22