Lecture 9-2: Linear Regression II

Dr. James Chen, Animal Data Scientist, School of Animal Sciences

2023 APSC-5984 SS: Agriculture Data Science

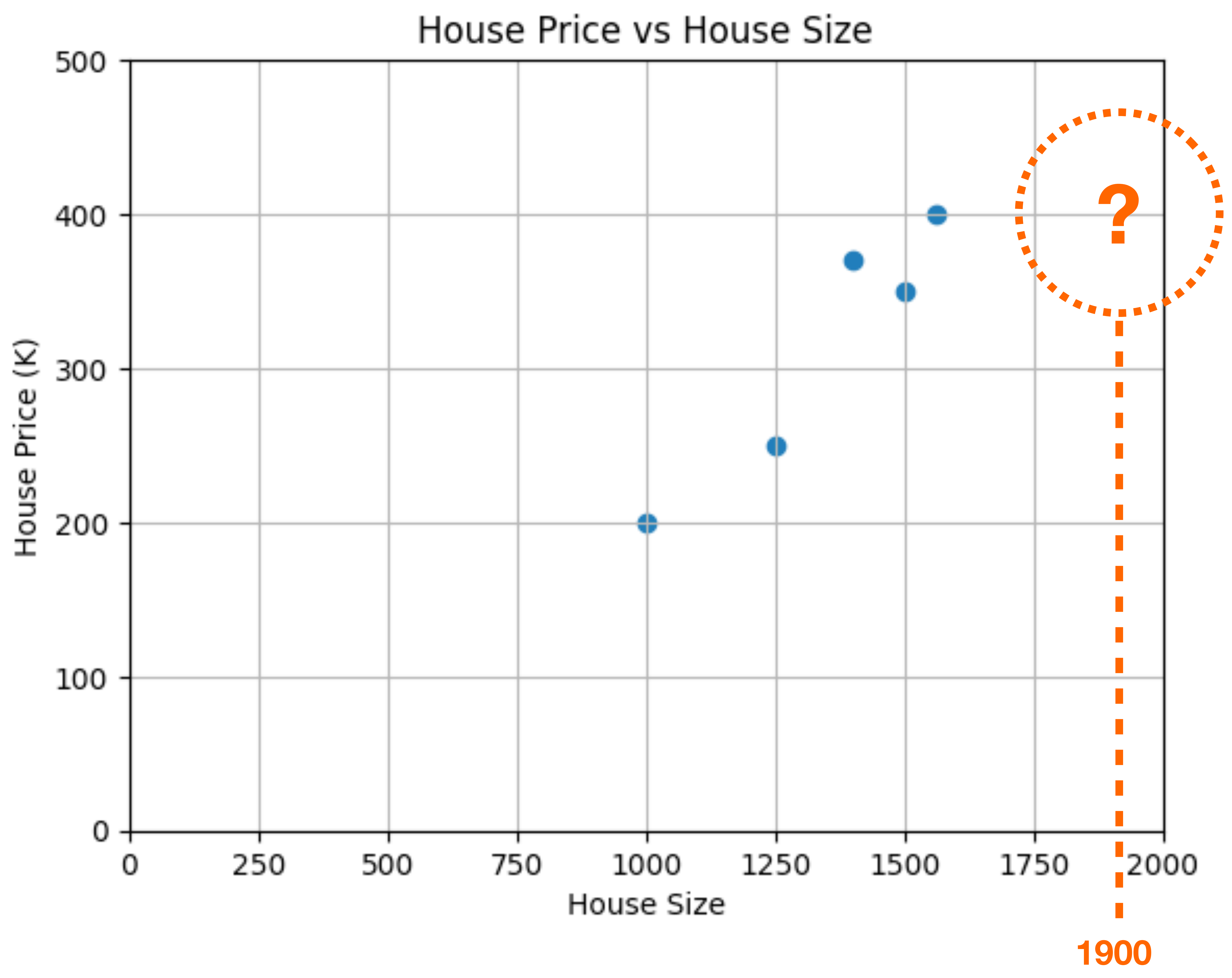
Recap - Supervised Learning

X's: "input" variable / features / independent variables

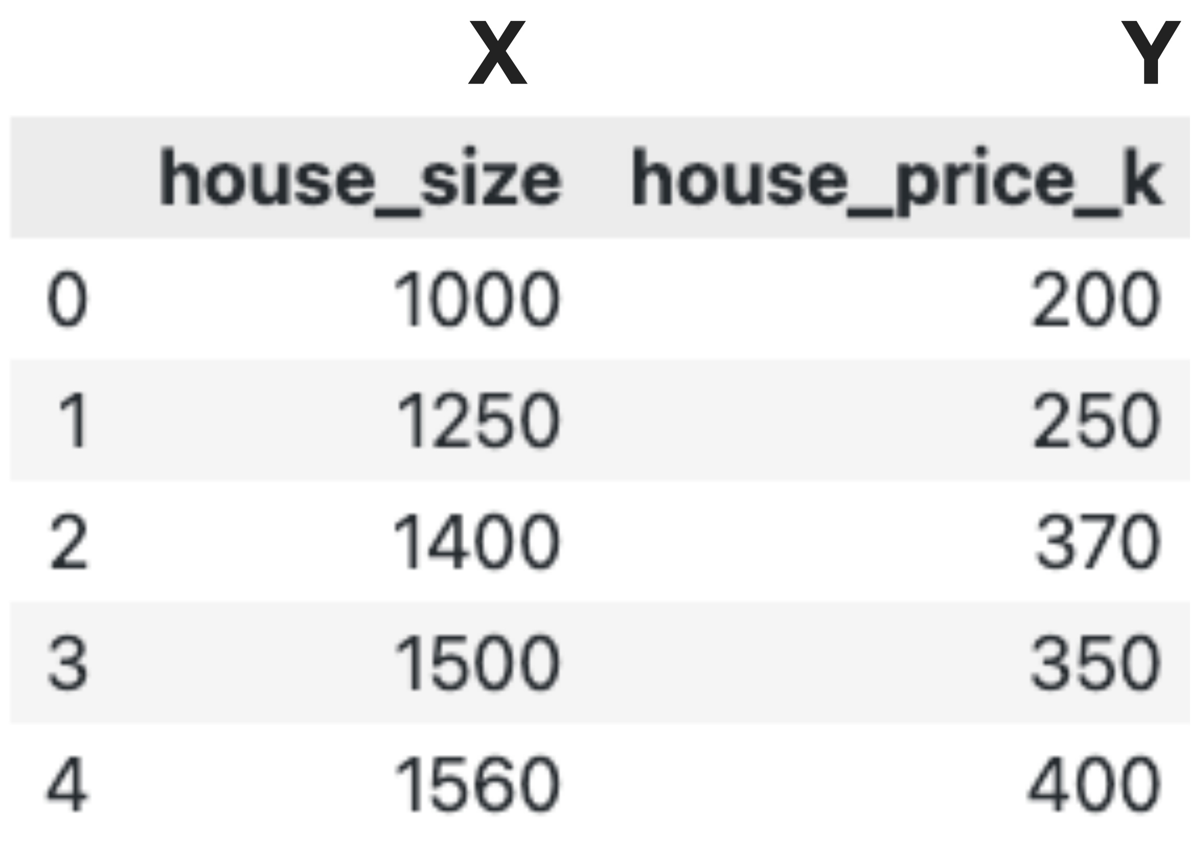
Y's: "output" variable / labels / response variables

	X	Y
	house_size	house_price_k
0	1000	200
1	1250	250
2	1400	370
3	1500	350
4	1560	400
5	1900	???

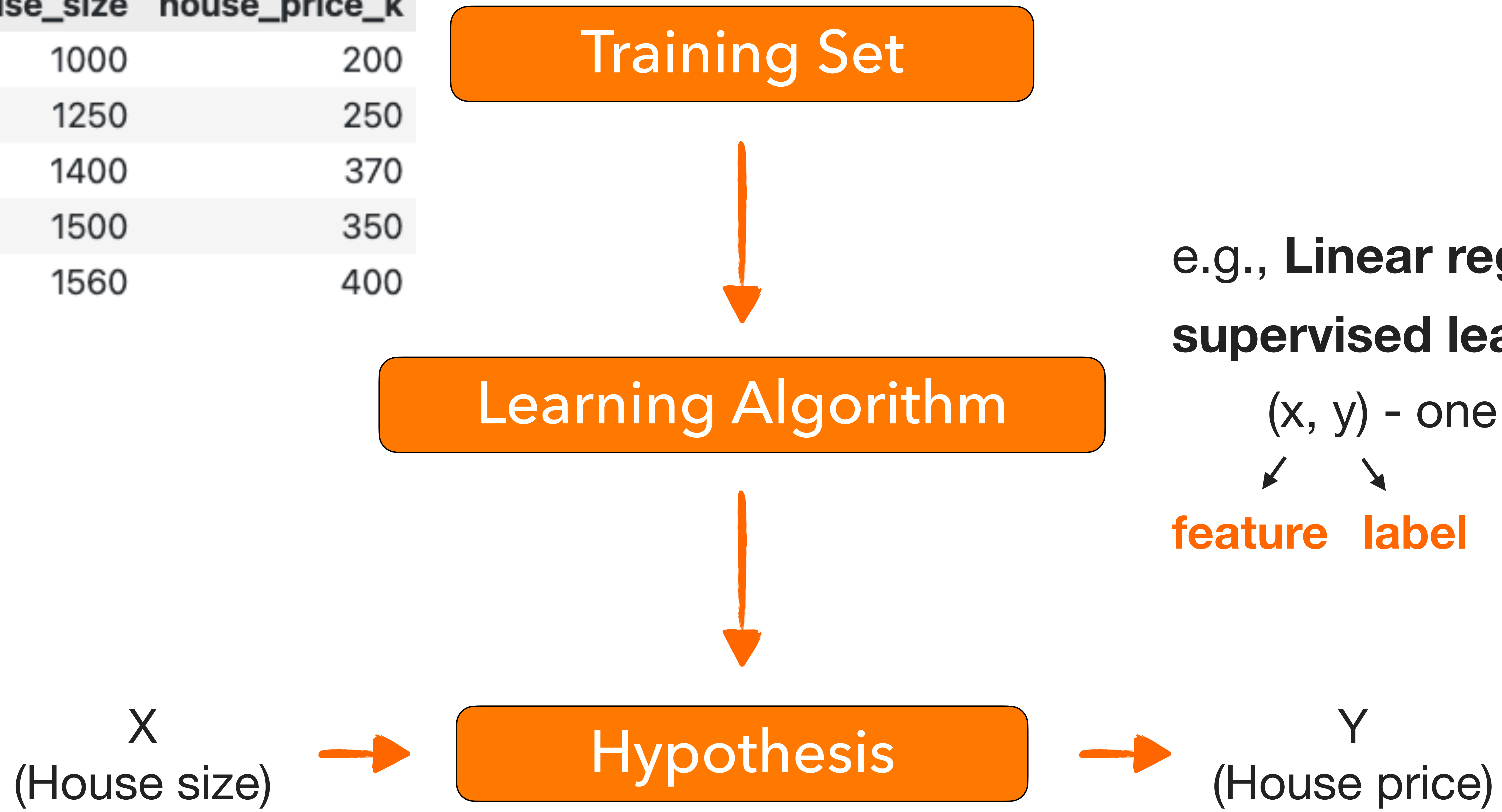
How do we predict the price?



Recap - Supervised Learning



	X house_size	Y house_price_k
0	1000	200
1	1250	250
2	1400	370
3	1500	350
4	1560	400



$$y = h(x) = \beta_0 + \beta_1 x$$

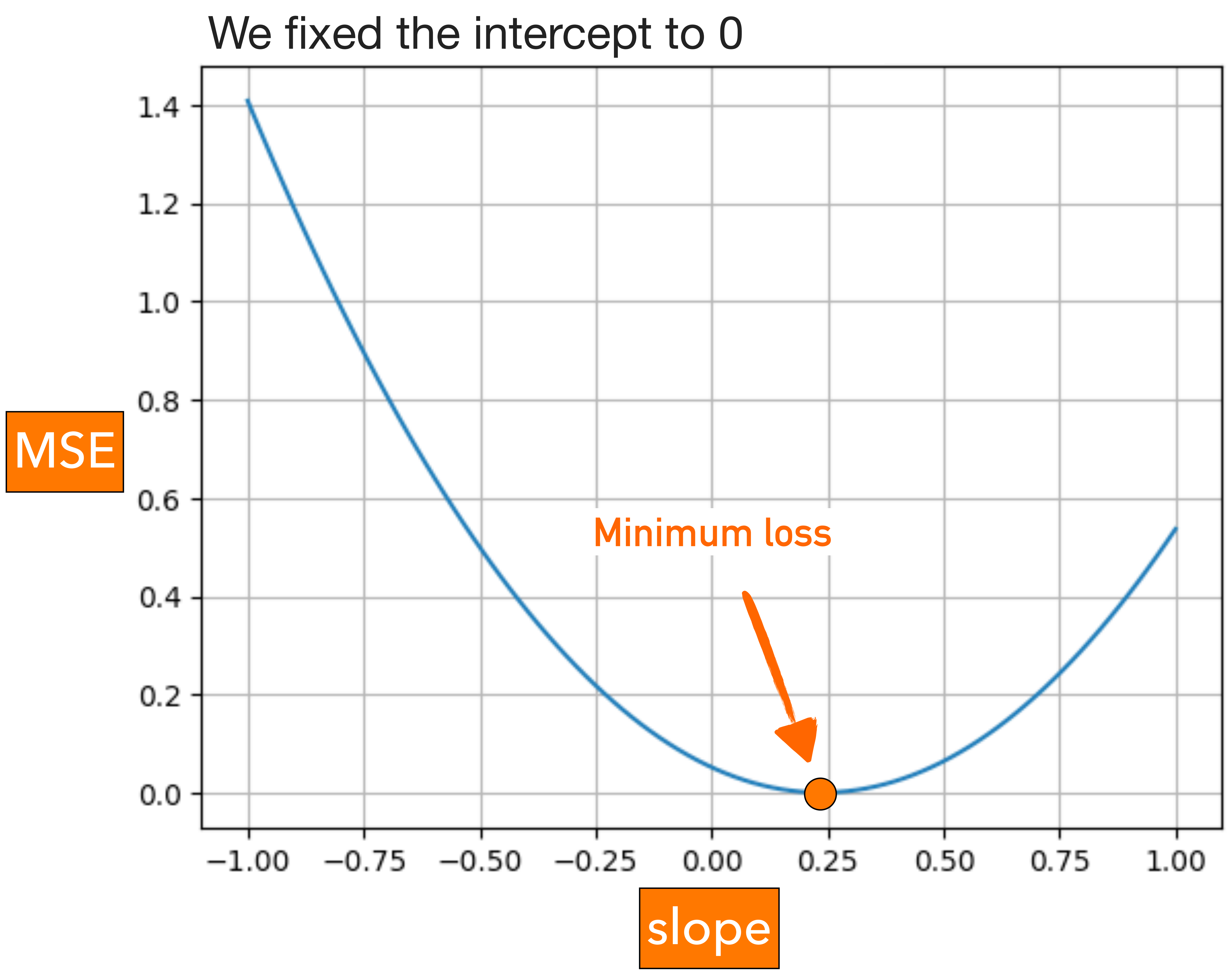
The hypothesis that maps x to y

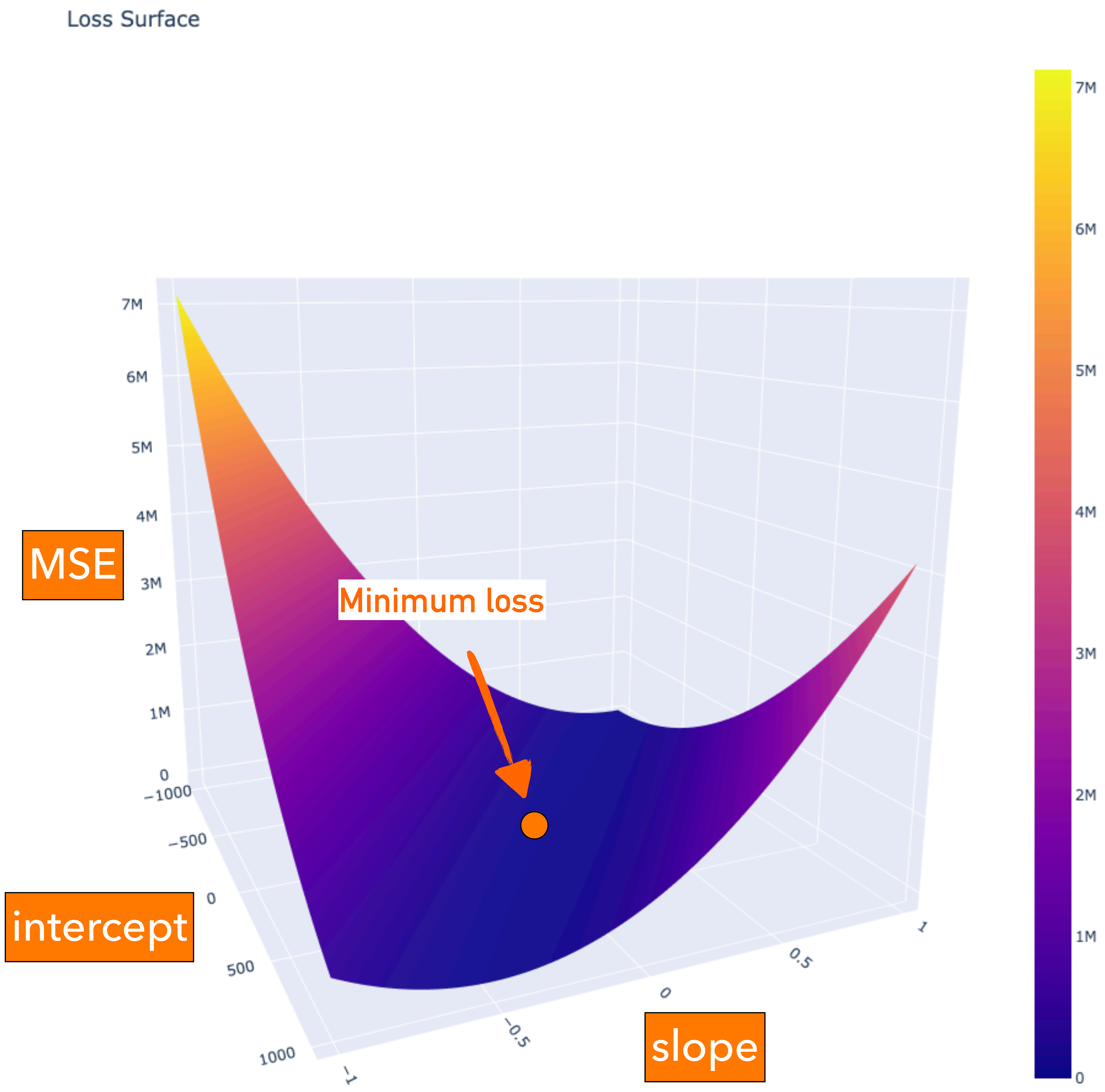
e.g., **Linear regression** is a type of **supervised learning algorithm:**

$$\begin{tikzcd}[column sep=small] & (x,\ y) \text{ - one training example} \arrow[dl] \arrow[d] \\ \textbf{feature} & \textbf{label} \end{tikzcd}$$

Recap - Loss Curve and Surface

How do we find a hypothesis that minimizes the loss? Differentiation!





Recap - Ordinary Least Squares

The minimization problem can be extended as

$$
\begin{aligned}
arg\min \mathrm{SSE} = arg\min_{\beta} f(\beta) &= arg\min_{\beta} (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) \\
&= arg\min_{\beta} (\mathbf{y}^T - \beta^T\mathbf{X}^T)(\mathbf{y} - \mathbf{X}\beta) \\
&= arg\min_{\beta} \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\beta - \beta^T\mathbf{X}^T\mathbf{y} + \beta^T\mathbf{X}^T\mathbf{X}\beta \\
&= arg\min_{\beta} \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}\beta + \beta^T\mathbf{X}^T\mathbf{X}\beta
\end{aligned}
$$

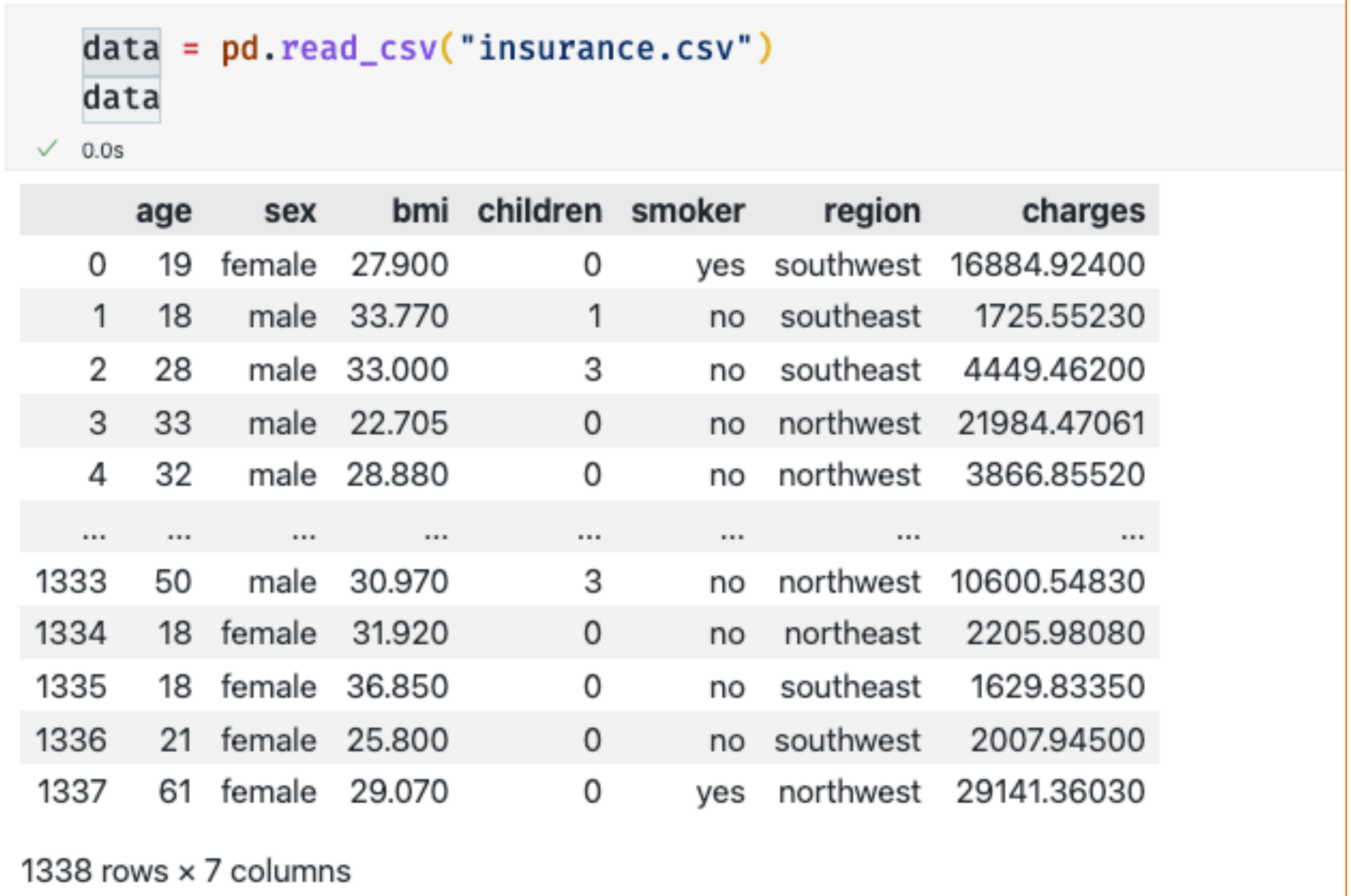
Find the partial derivatives of $f(\beta)$ with respect to β and set it to zero:

$$\frac{\partial f}{\partial \beta} = \beta^T\mathbf{X}^T\mathbf{X} - \mathbf{X}^T\mathbf{y} = 0$$

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \longleftarrow$$ This is the learning algorithm!

Medical Insurance Dataset

In this section, we will use a real-world dataset to demonstrate the multi-variable linear regression. The dataset contains information about medical insurance costs for 1338 people. The dataset is available on Kaggle.

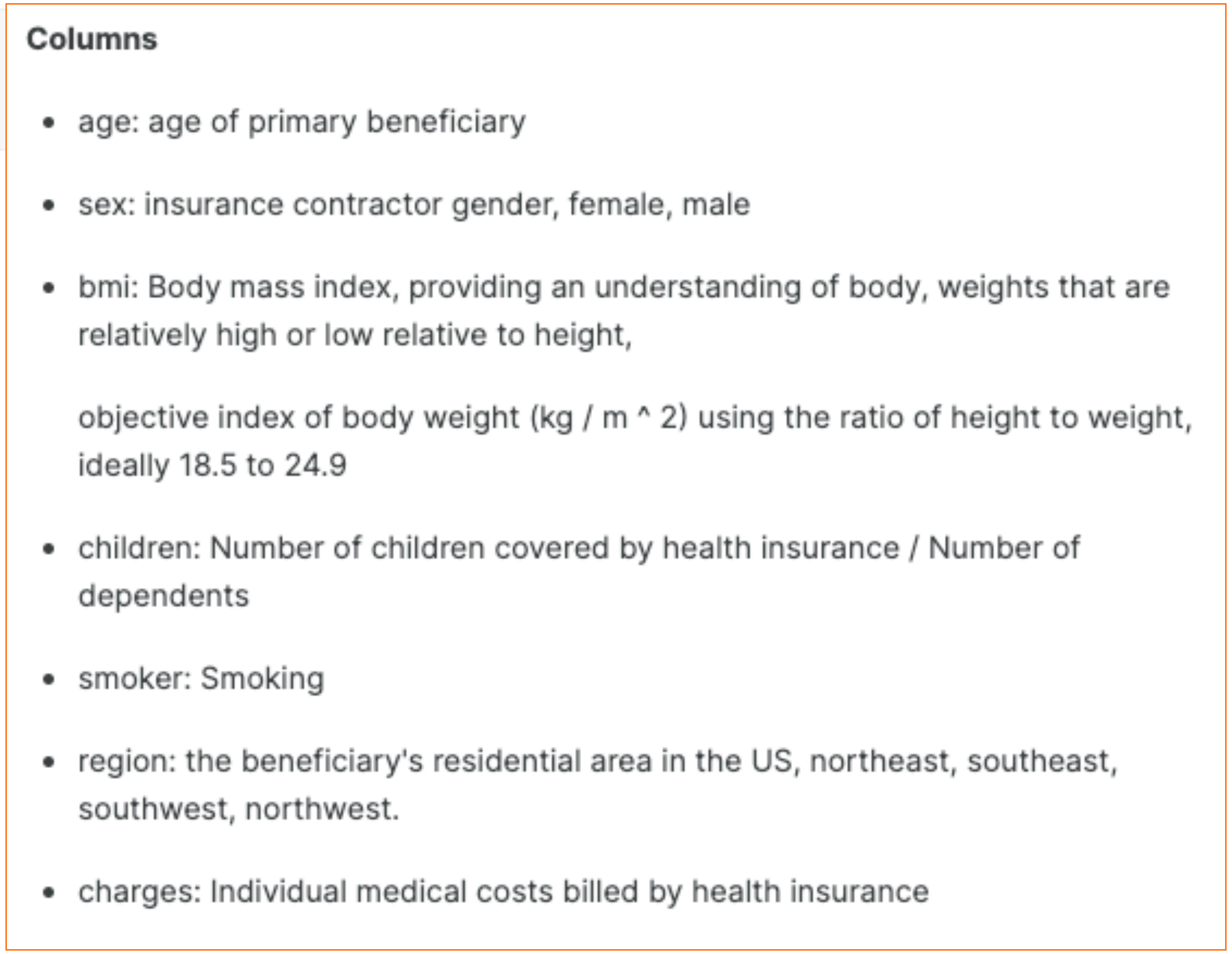


```
data = pd.read_csv("insurance.csv")
data
```

✓ 0.0s

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns



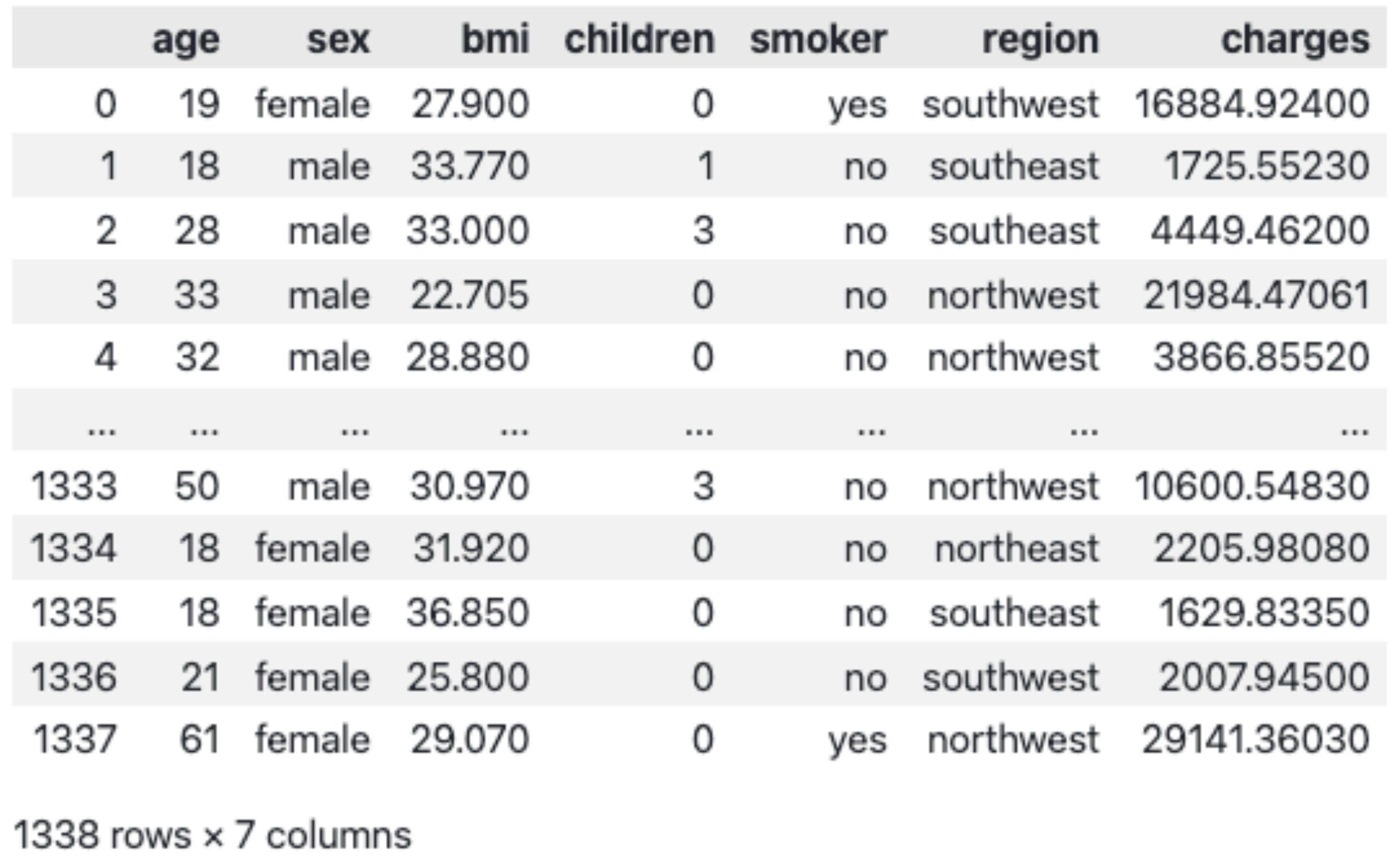
Columns

- age: age of primary beneficiary
- sex: insurance contractor gender, female, male
- bmi: Body mass index, providing an understanding of body, weights that are relatively high or low relative to height,

  objective index of body weight (kg / m ^ 2) using the ratio of height to weight, ideally 18.5 to 24.9
- children: Number of children covered by health insurance / Number of dependents
- smoker: Smoking
- region: the beneficiary's residential area in the US, northeast, southeast, southwest, northwest.
- charges: Individual medical costs billed by health insurance

Data Overview

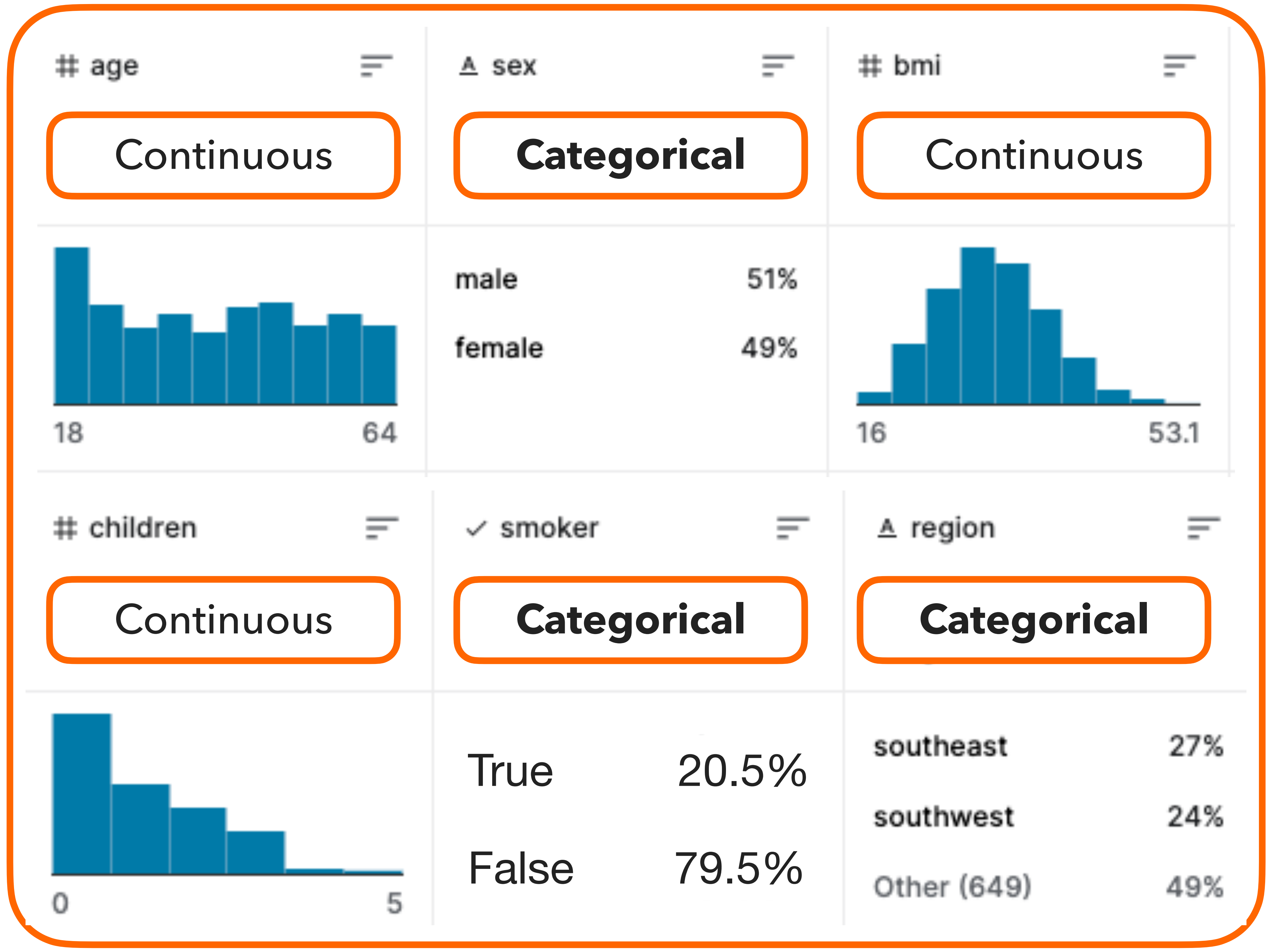
Check the features and labels



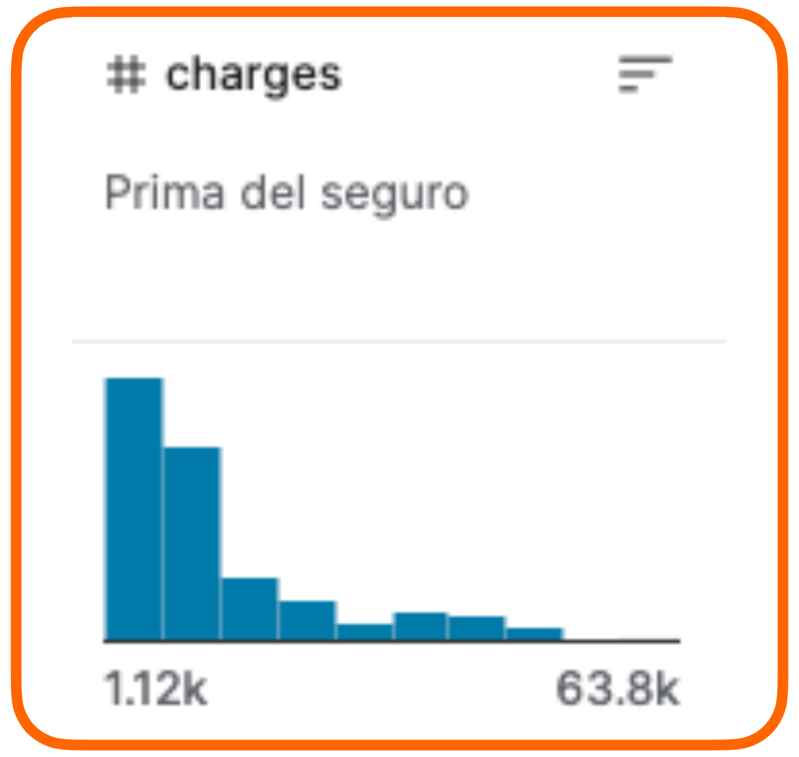
	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

Features



Labels



↓

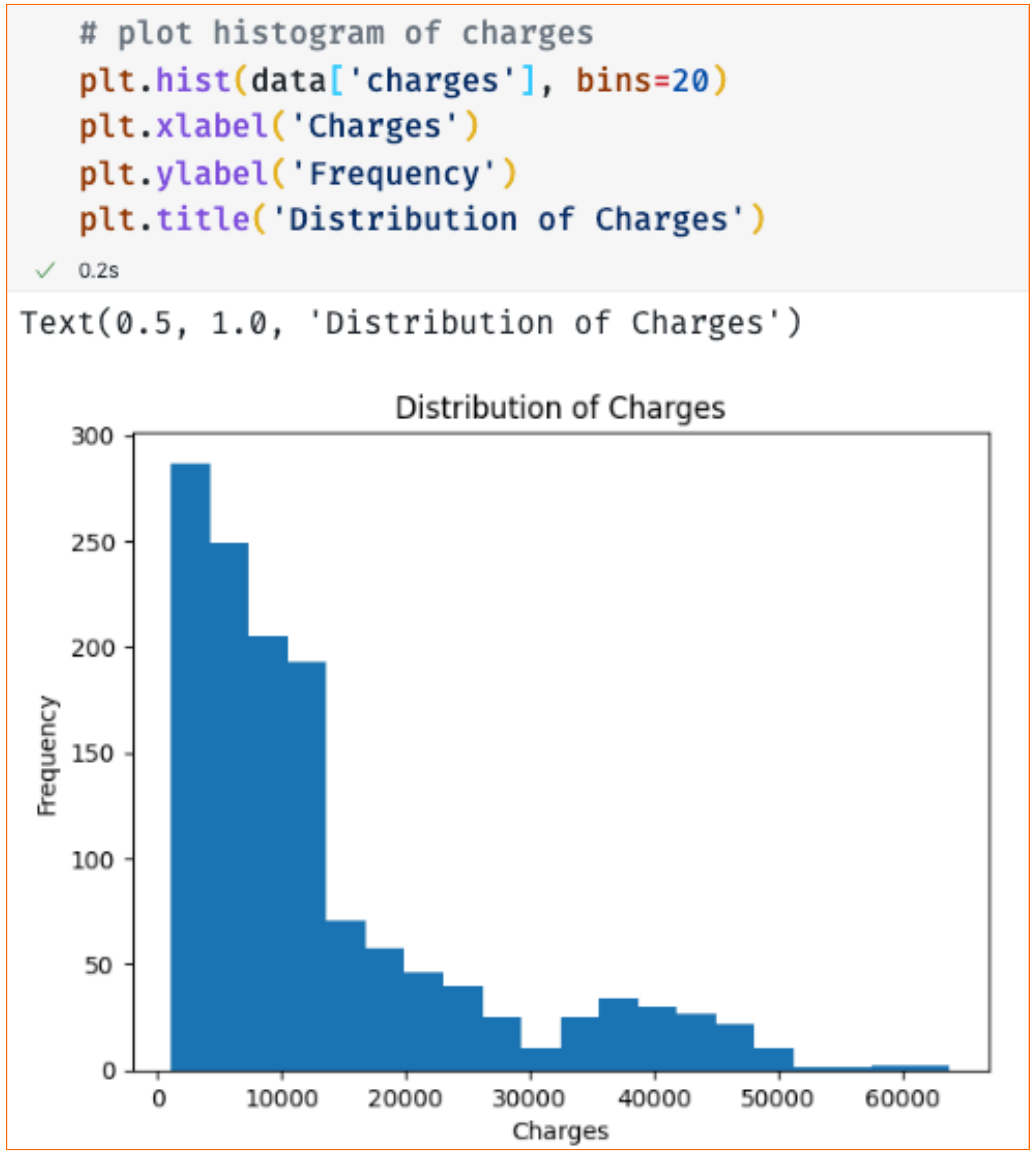
A skew distribution

Skew Distribution

Log-transformation

We want to avoid a **<u>skew</u>** label distribution as the model would tend to focus on that data-rich region and be **<u>less sensitive</u>** to **<u>rare</u>** cases (but important).

Original

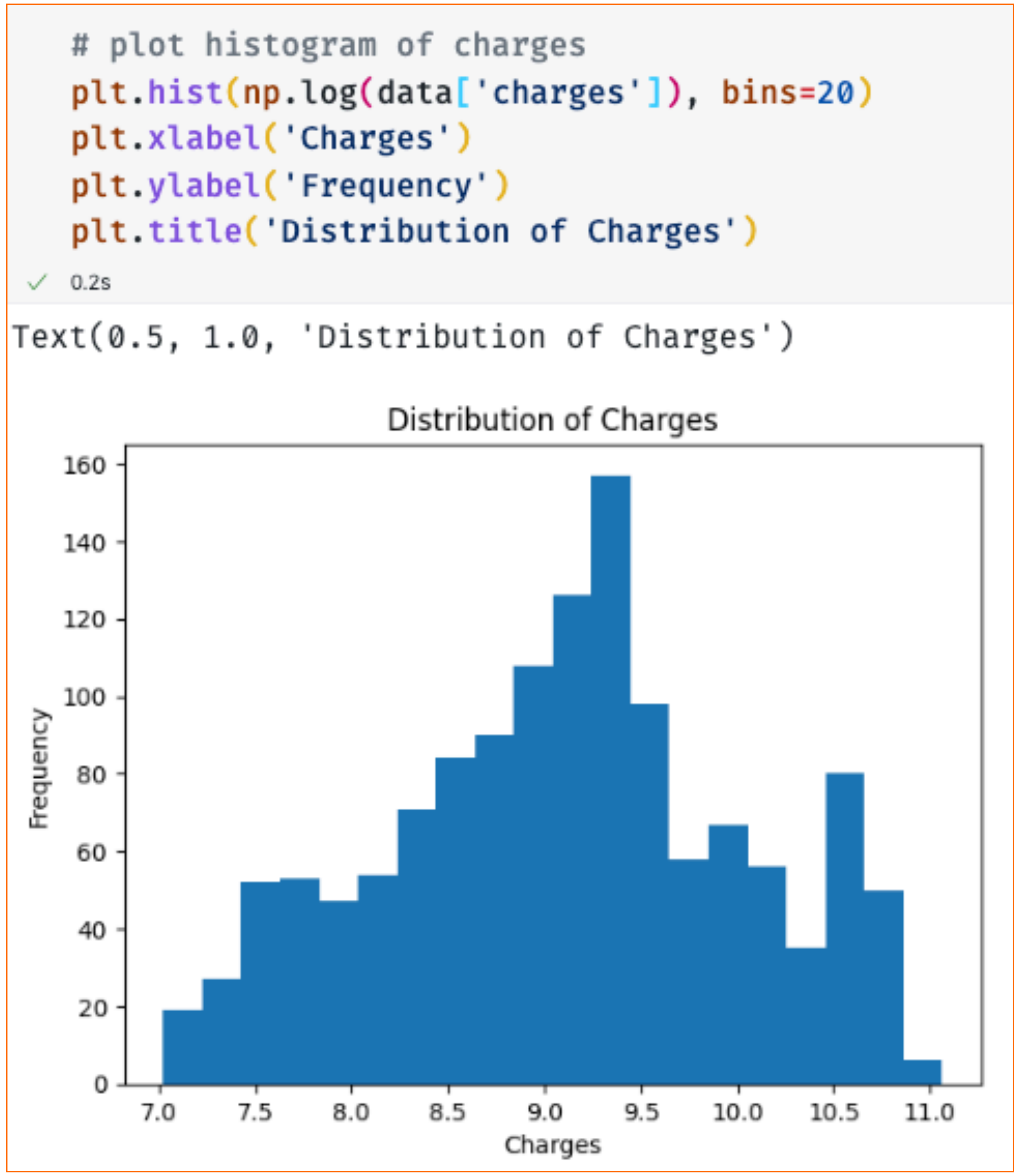


```
# plot histogram of charges
plt.hist(data['charges'], bins=20)
plt.xlabel('Charges')
plt.ylabel('Frequency')
plt.title('Distribution of Charges')
```

✓ 0.2s

```
Text(0.5, 1.0, 'Distribution of Charges')
```

Transformed



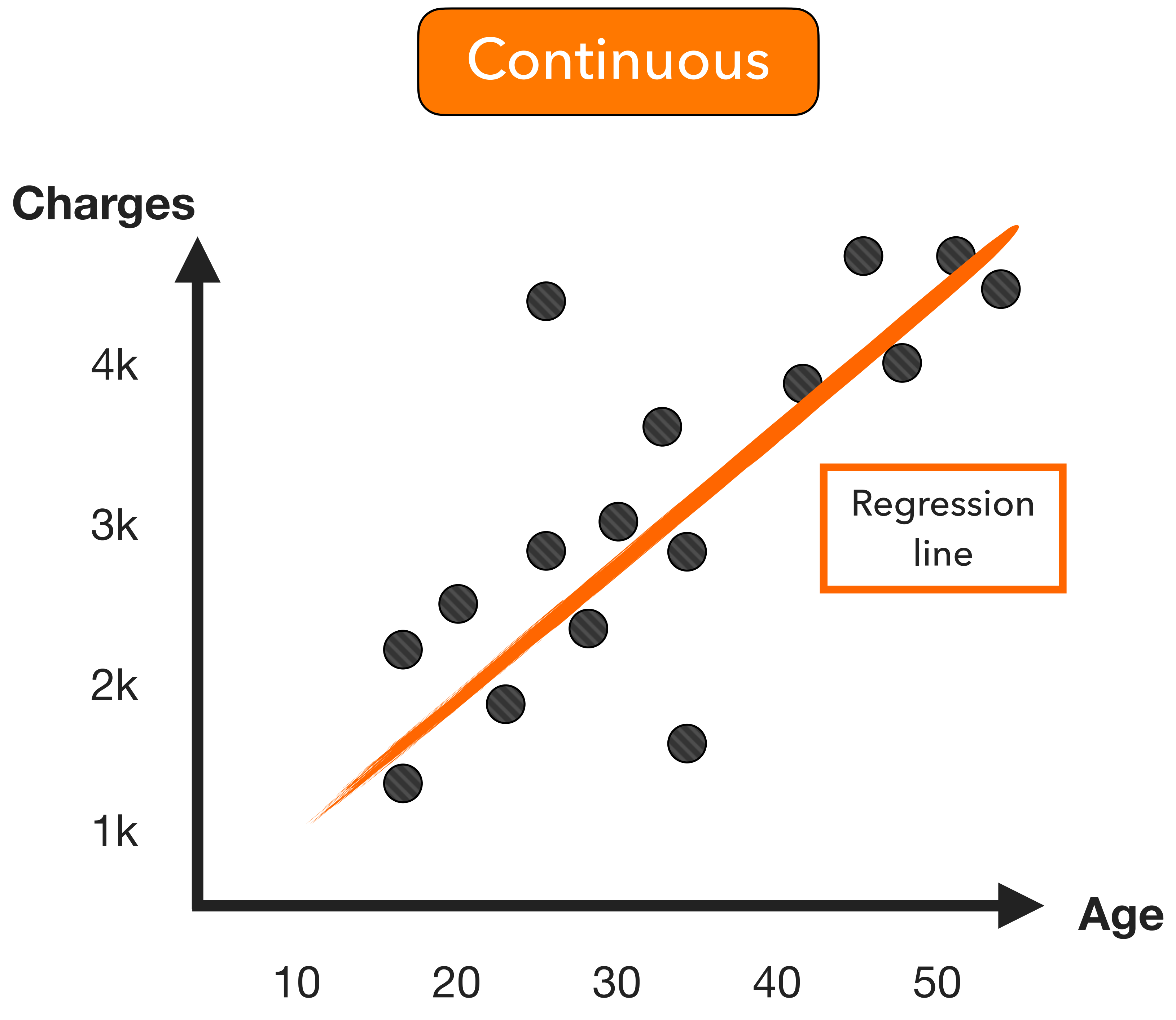
```
# plot histogram of charges
plt.hist(np.log(data['charges']), bins=20)
plt.xlabel('Charges')
plt.ylabel('Frequency')
plt.title('Distribution of Charges')
```

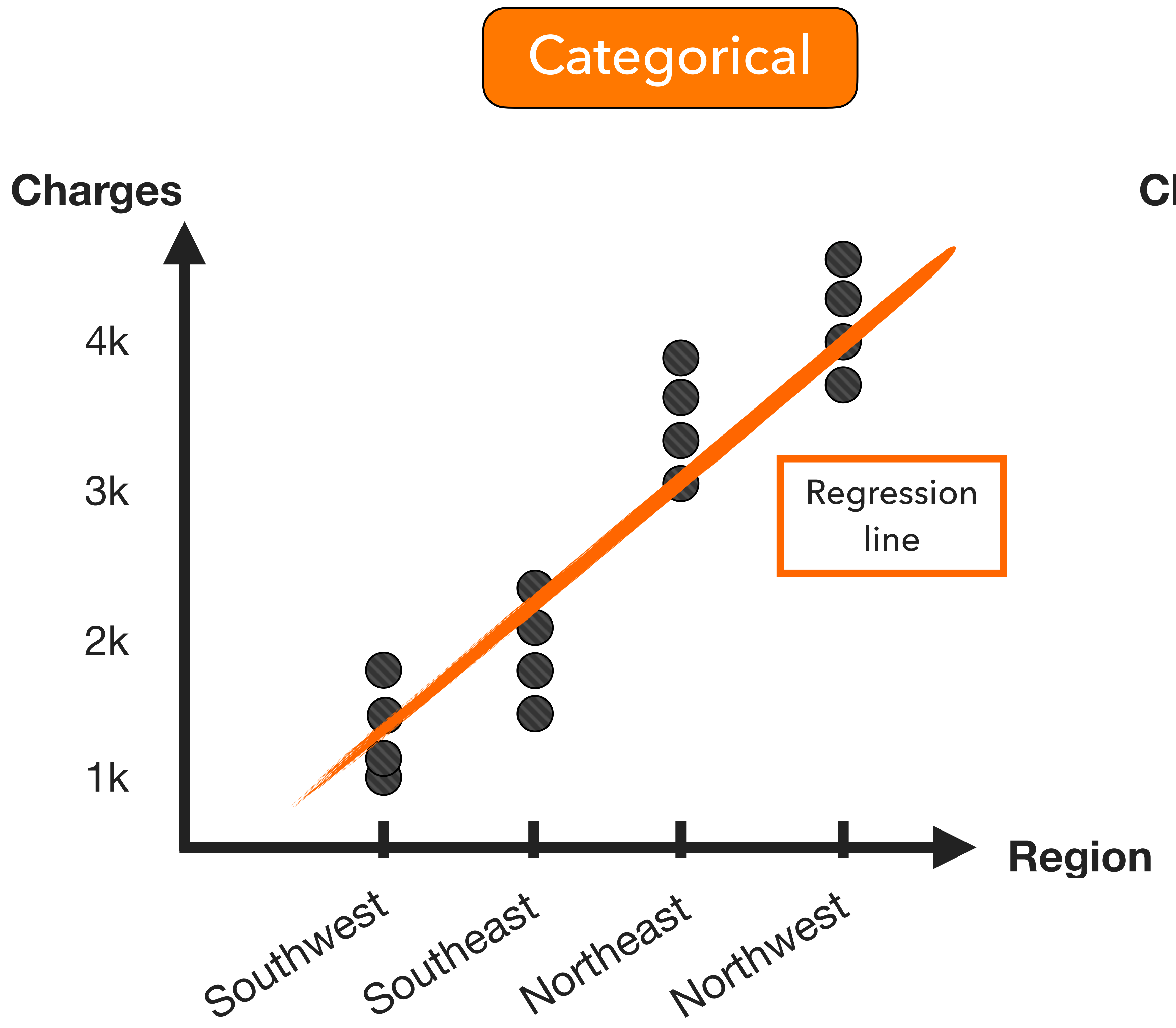
✓ 0.2s

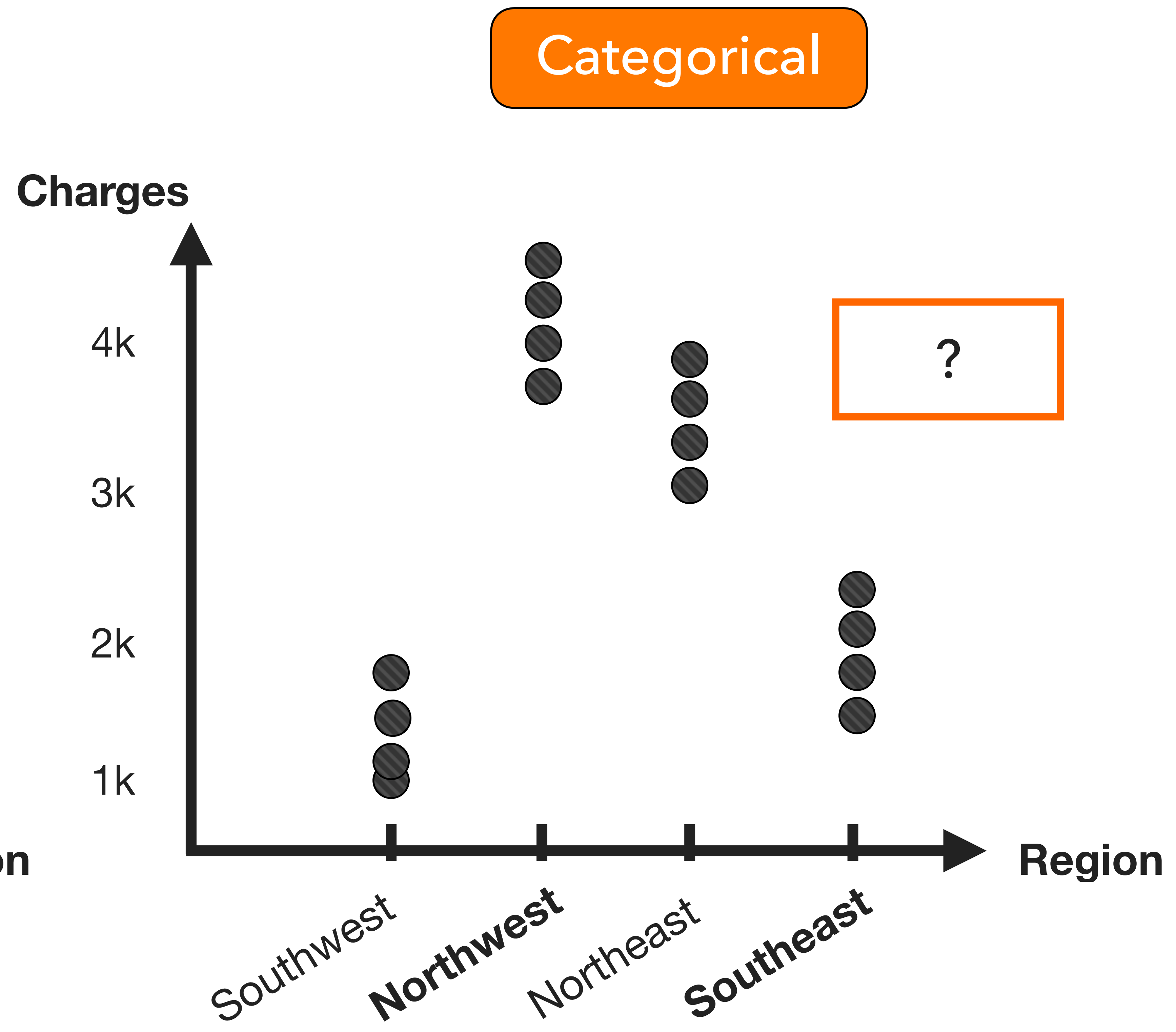
```
Text(0.5, 1.0, 'Distribution of Charges')
```

Categorical Variable

How does a linear model handle (understand) categorical variables?



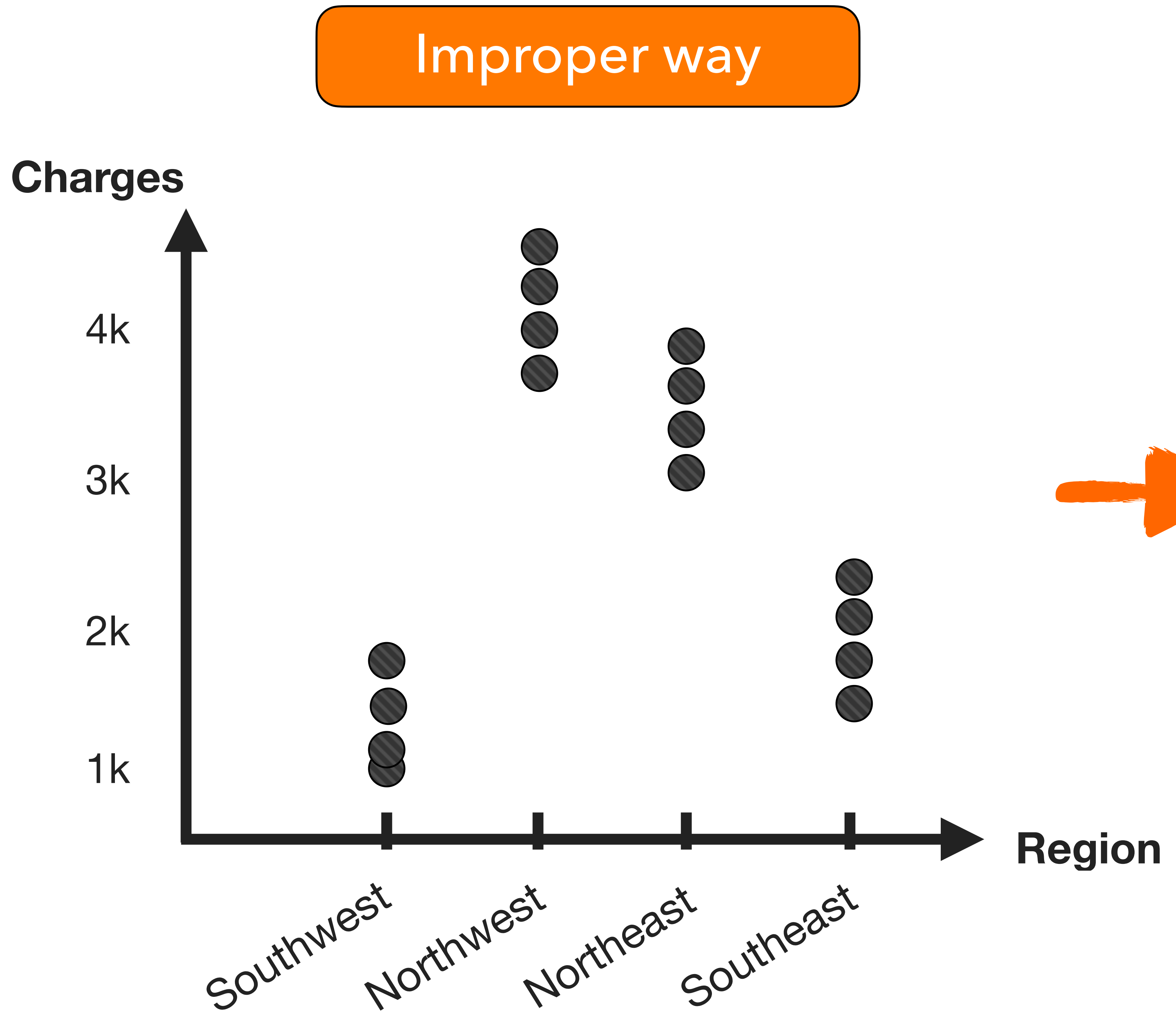


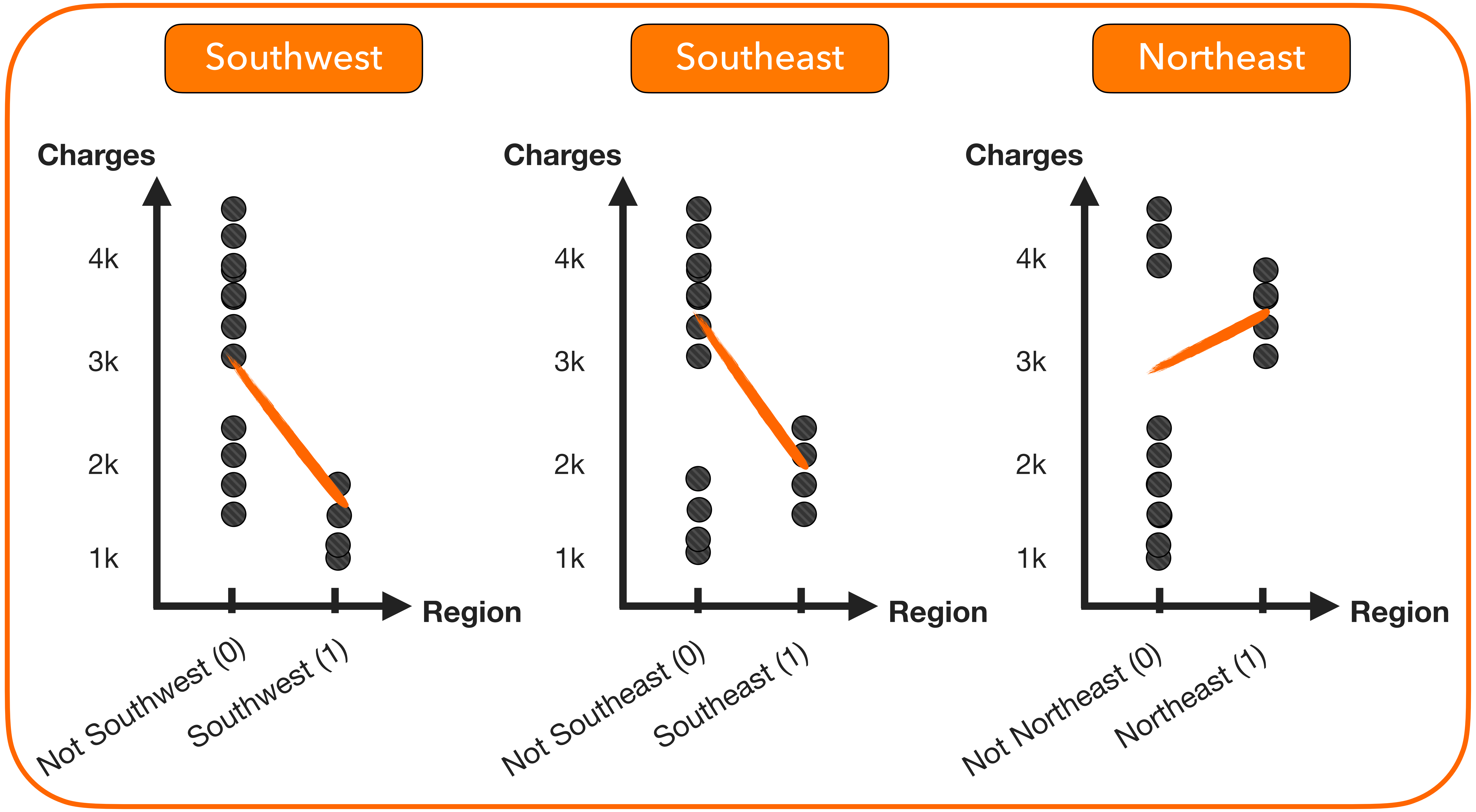


What if we change the order?
Will the hypothesis (e.g., slope) remain the same?

One-Hot Encoding (Dummy Variables)

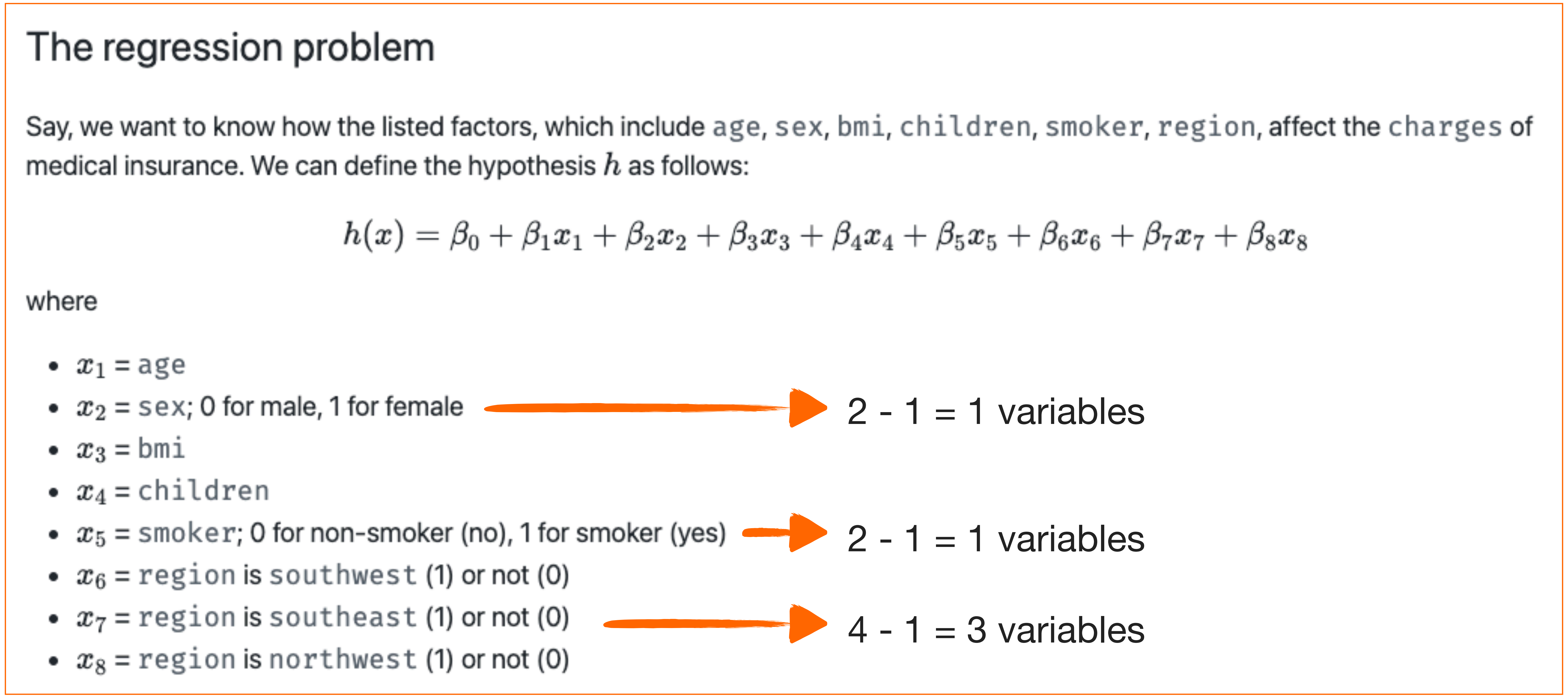
How does a linear model handle (understand) categorical variables?





We use the fourth category, "Northwest," as a reference category (intercept) in our regression model. This means that we only need to create three dummy variables to represent the other three categories. The reference category is represented by the intercept term in the regression model.

The Regression Model



The regression problem

Say, we want to know how the listed factors, which include `age`, `sex`, `bmi`, `children`, `smoker`, `region`, affect the `charges` of medical insurance. We can define the hypothesis h as follows:

$$h(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6 + \beta_7 x_7 + \beta_8 x_8$$

where

- x_1 = `age`
- x_2 = `sex`; 0 for male, 1 for female $\rightarrow$ 2 - 1 = 1 variables
- x_3 = `bmi`
- x_4 = `children`
- x_5 = `smoker`; 0 for non-smoker (no), 1 for smoker (yes) $\rightarrow$ 2 - 1 = 1 variables
- x_6 = `region` is `southwest` (1) or not (0)
- x_7 = `region` is `southeast` (1) or not (0) $\rightarrow$ 4 - 1 = 3 variables
- x_8 = `region` is `northwest` (1) or not (0)

Matrix form

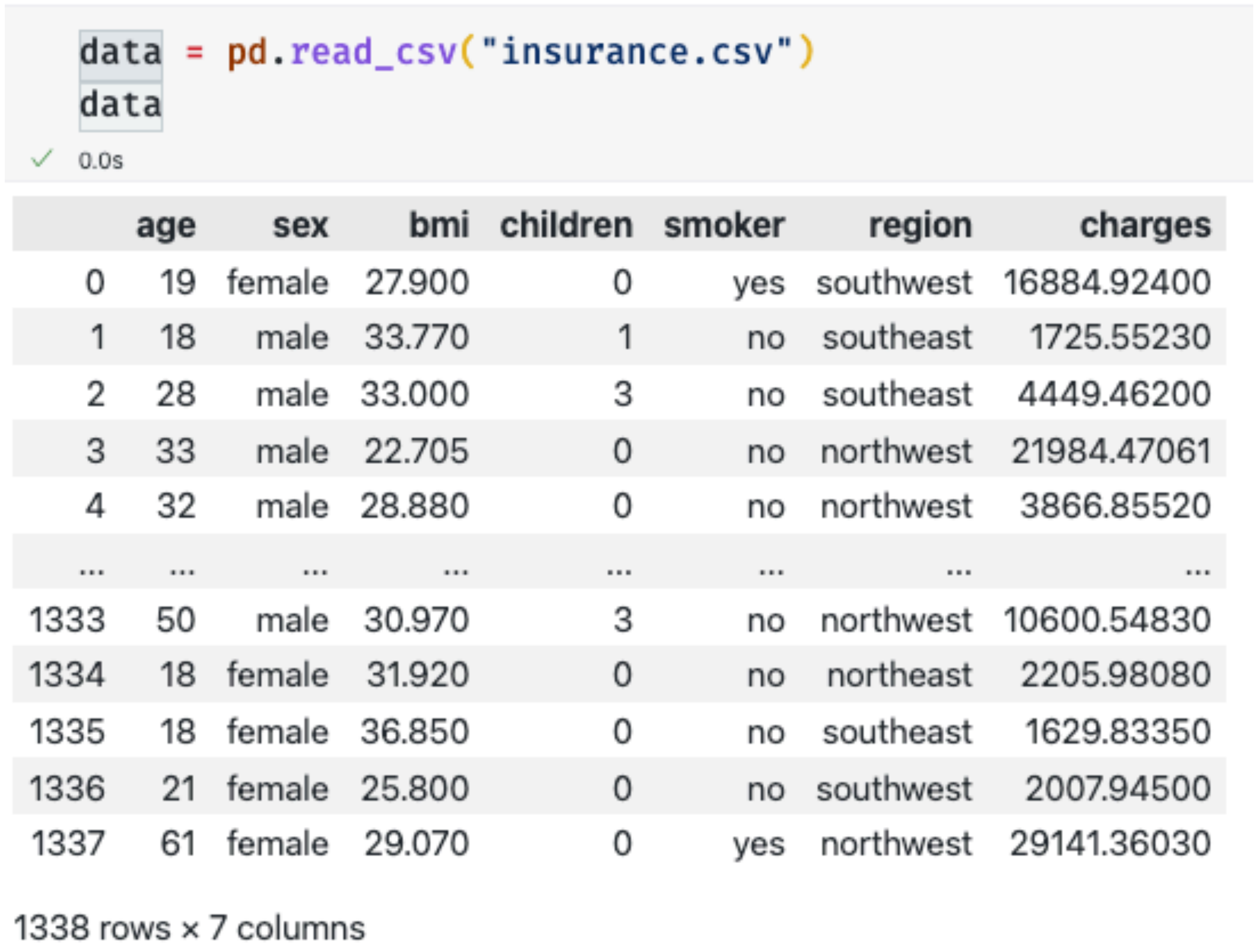
$$\underset{N \times 1}{y} = \underset{N \times 9}{X}\,\underset{9 \times 1}{\beta}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & X_{11} & X_{12} & X_{13} & X_{14} & X_{15} & X_{16} & X_{17} & X_{18} \\ 1 & X_{21} & X_{22} & X_{23} & X_{24} & X_{25} & X_{26} & X_{27} & X_{28} \\ 1 & X_{31} & X_{32} & X_{33} & X_{34} & X_{35} & X_{36} & X_{37} & X_{38} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ 1 & X_{n1} & X_{n2} & X_{n3} & X_{n4} & X_{n5} & X_{n6} & X_{n7} & X_{n8} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \\ \beta_6 \\ \beta_7 \\ \beta_8 \end{bmatrix}$$

Make X and Y

Dummy variables

$$X\beta = y$$

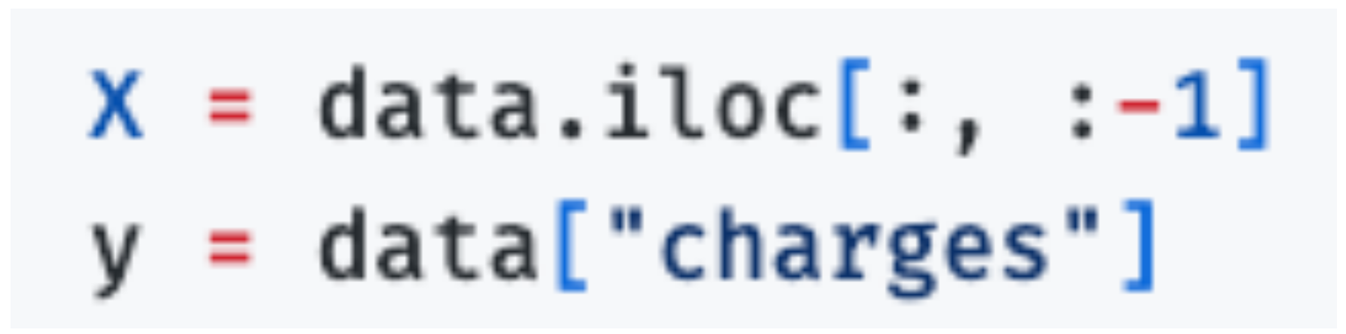


```python
data = pd.read_csv("insurance.csv")
data
```

✓ 0.0s

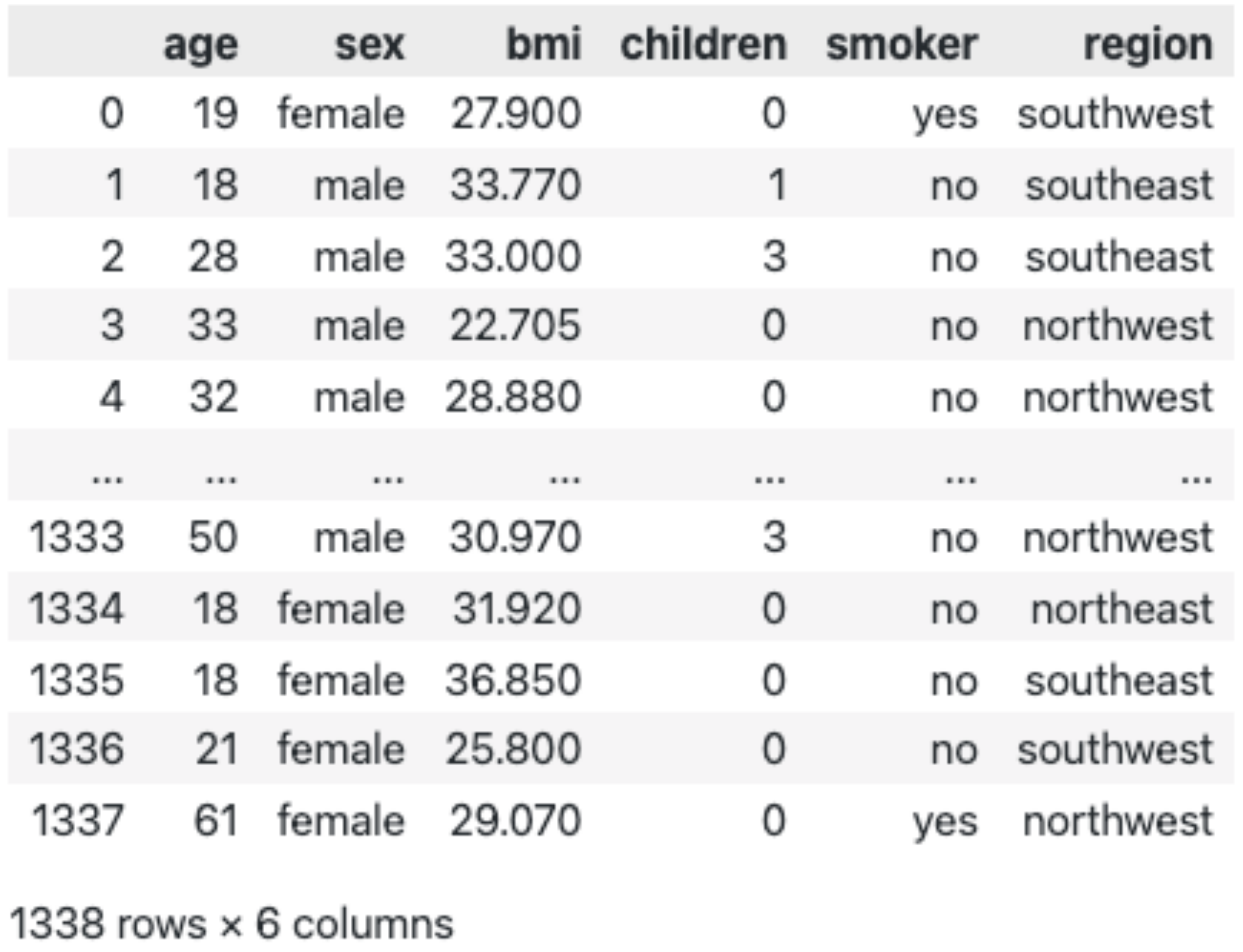
	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns



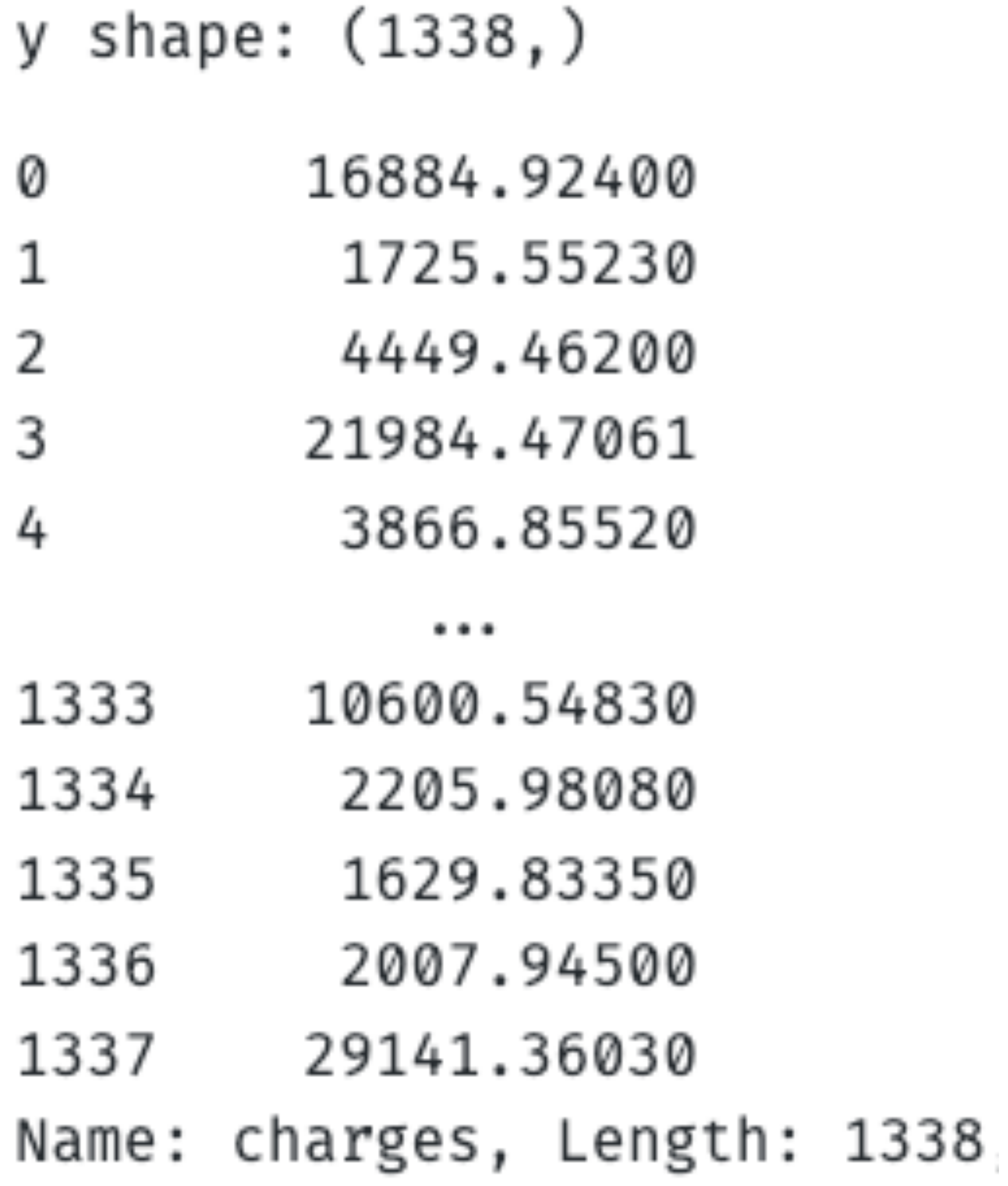
```python
X = data.iloc[:, :-1]
y = data["charges"]
```

```
X shape: (1338, 6)
```



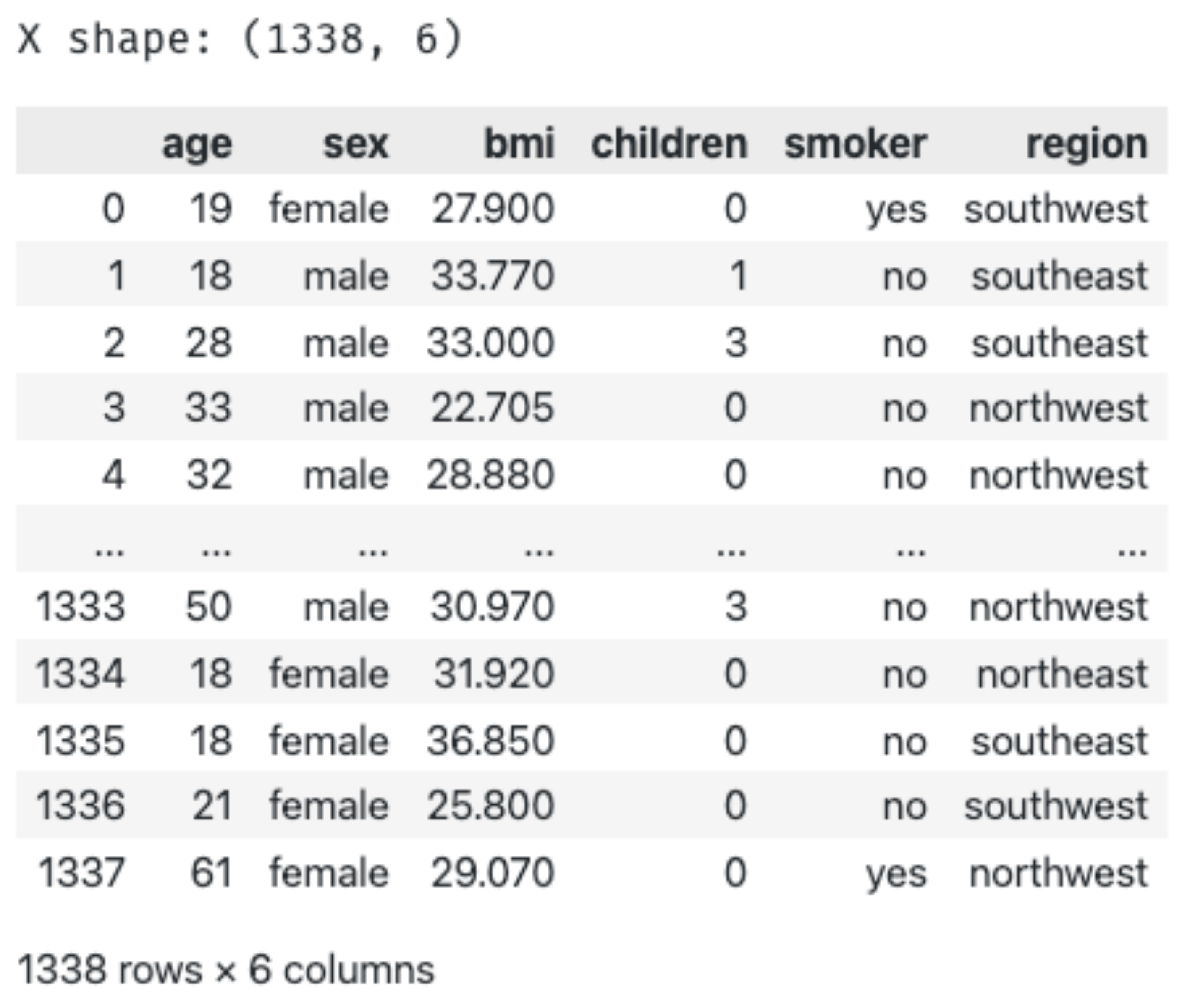
	age	sex	bmi	children	smoker	region
0	19	female	27.900	0	yes	southwest
1	18	male	33.770	1	no	southeast
2	28	male	33.000	3	no	southeast
3	33	male	22.705	0	no	northwest
4	32	male	28.880	0	no	northwest
...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest
1334	18	female	31.920	0	no	northeast
1335	18	female	36.850	0	no	southeast
1336	21	female	25.800	0	no	southwest
1337	61	female	29.070	0	yes	northwest

1338 rows × 6 columns



```
y shape: (1338,)

0       16884.92400
1        1725.55230
2        4449.46200
3       21984.47061
4        3866.85520
           ...
1333    10600.54830
1334     2205.98080
1335     1629.83350
1336     2007.94500
1337    29141.36030
Name: charges, Length: 1338
```

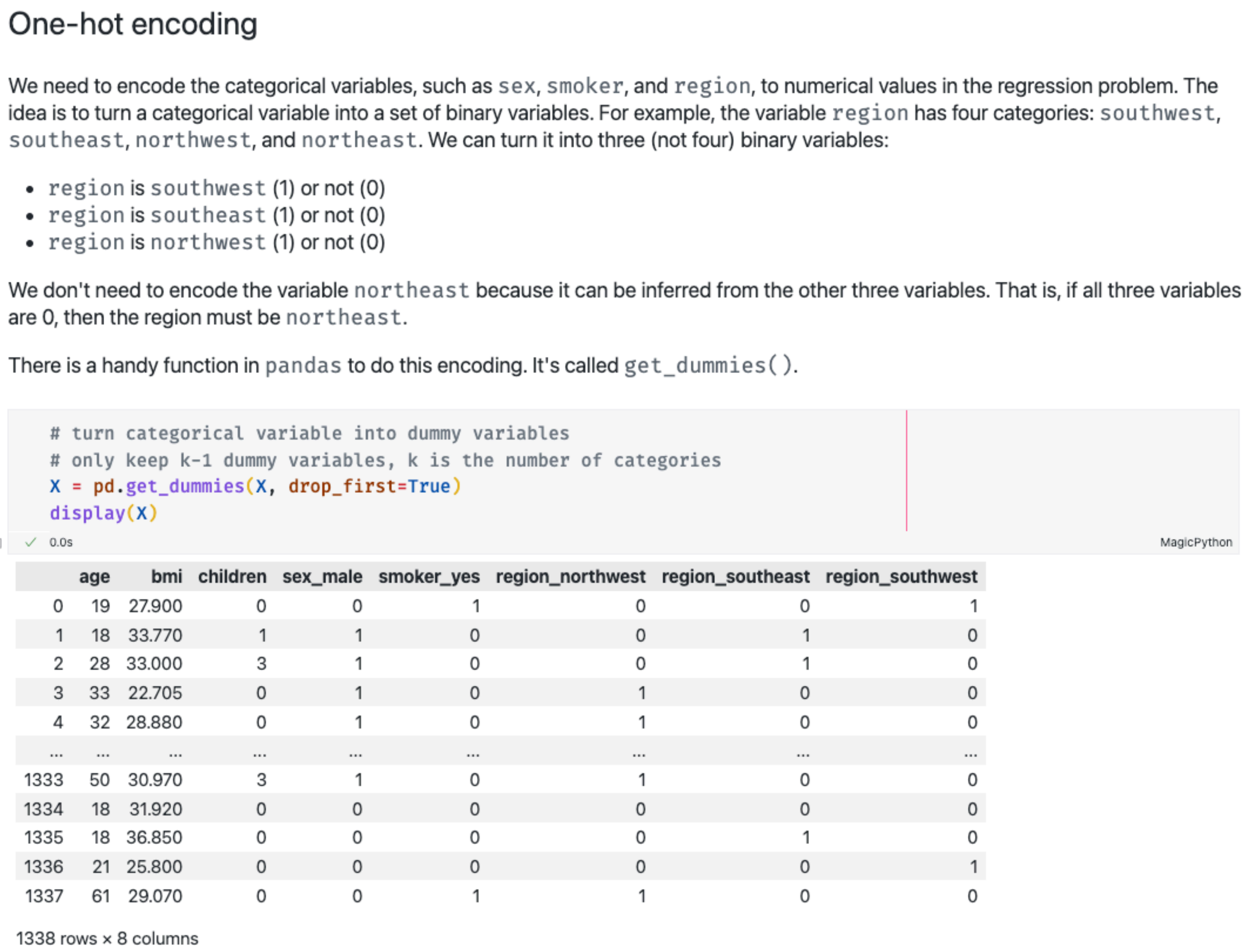
Make X and Y



```
X shape: (1338, 6)
```

	age	sex	bmi	children	smoker	region
0	19	female	27.900	0	yes	southwest
1	18	male	33.770	1	no	southeast
2	28	male	33.000	3	no	southeast
3	33	male	22.705	0	no	northwest
4	32	male	28.880	0	no	northwest
...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest
1334	18	female	31.920	0	no	northeast
1335	18	female	36.850	0	no	southeast
1336	21	female	25.800	0	no	southwest
1337	61	female	29.070	0	yes	northwest

1338 rows × 6 columns



One-hot encoding

We need to encode the categorical variables, such as `sex`, `smoker`, and `region`, to numerical values in the regression problem. The idea is to turn a categorical variable into a set of binary variables. For example, the variable `region` has four categories: `southwest`, `southeast`, `northwest`, and `northeast`. We can turn it into three (not four) binary variables:

- `region` is `southwest` (1) or not (0)
- `region` is `southeast` (1) or not (0)
- `region` is `northwest` (1) or not (0)

We don't need to encode the variable `northeast` because it can be inferred from the other three variables. That is, if all three variables are 0, then the region must be `northeast`.

There is a handy function in `pandas` to do this encoding. It's called `get_dummies()`.

```
# turn categorical variable into dummy variables
# only keep k-1 dummy variables, k is the number of categories
X = pd.get_dummies(X, drop_first=True)
display(X)
```

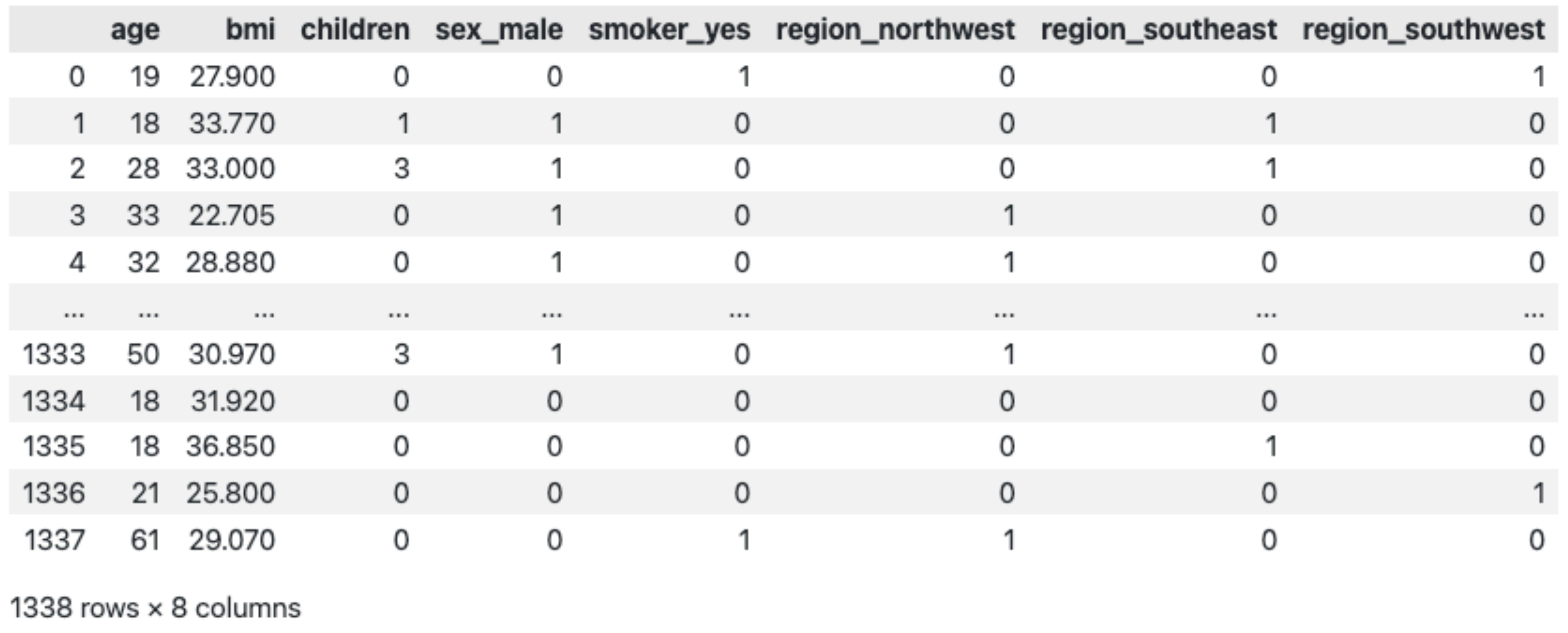
✓ 0.0s MagicPython

	age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	region_southwest
0	19	27.900	0	0	1	0	0	1
1	18	33.770	1	1	0	0	1	0
2	28	33.000	3	1	0	0	1	0
3	33	22.705	0	1	0	1	0	0
4	32	28.880	0	1	0	1	0	0
...	...	...	...	...	...	...	...	...
1333	50	30.970	3	1	0	1	0	0
1334	18	31.920	0	0	0	0	0	0
1335	18	36.850	0	0	0	0	1	0
1336	21	25.800	0	0	0	0	0	1
1337	61	29.070	0	0	1	1	0	0

1338 rows × 8 columns

Make X and Y

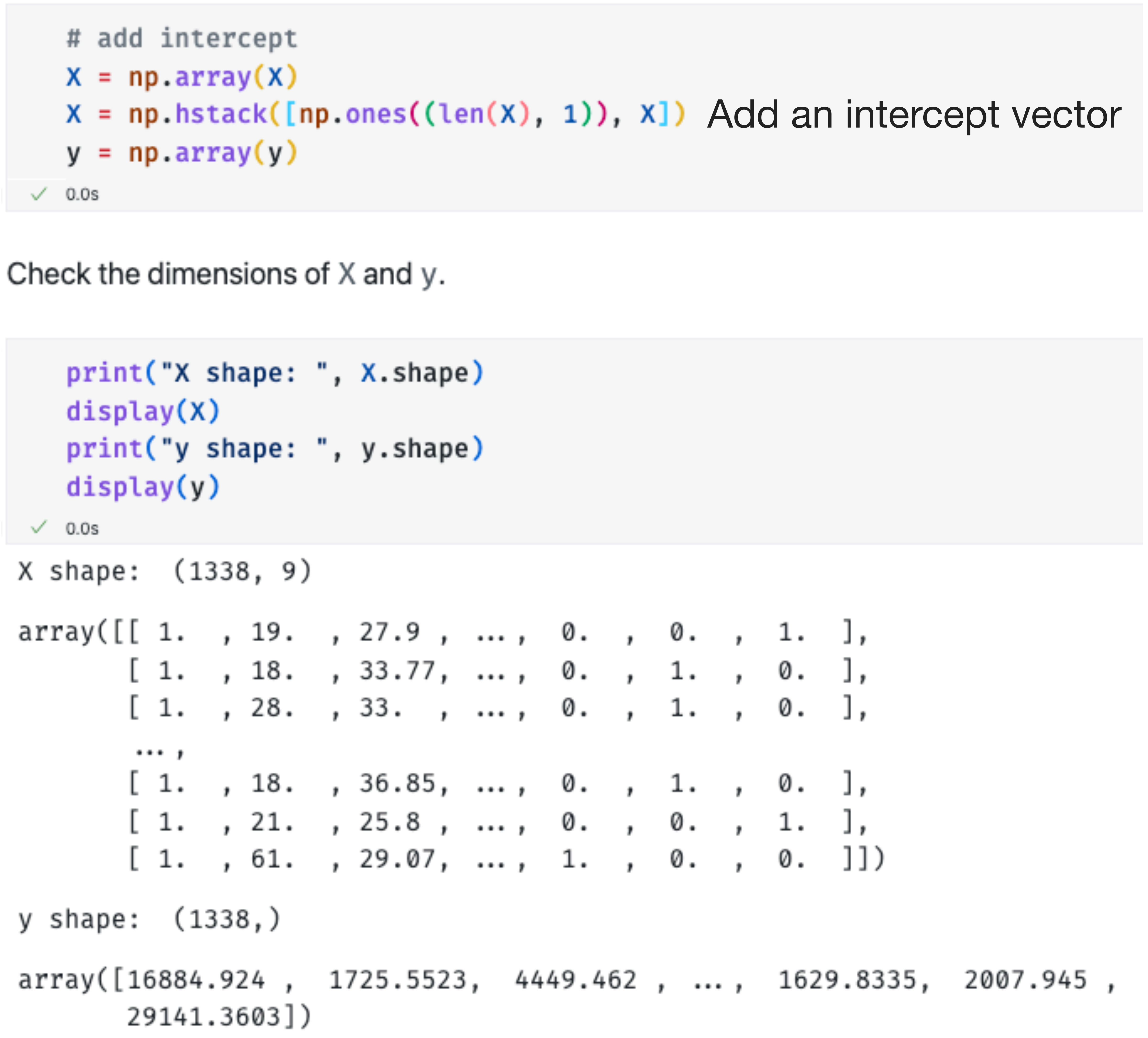
Dataframe



	age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	region_southwest
0	19	27.900	0	0	1	0	0	1
1	18	33.770	1	1	0	0	1	0
2	28	33.000	3	1	0	0	1	0
3	33	22.705	0	1	0	1	0	0
4	32	28.880	0	1	0	1	0	0
...	...	...	...	...	...	...	...	...
1333	50	30.970	3	1	0	1	0	0
1334	18	31.920	0	0	0	0	0	0
1335	18	36.850	0	0	0	0	1	0
1336	21	25.800	0	0	0	0	0	1
1337	61	29.070	0	0	1	1	0	0

1338 rows × 8 columns

Numpy Array



```
# add intercept
X = np.array(X)
X = np.hstack([np.ones((len(X), 1)), X])
y = np.array(y)
```

✓ 0.0s

Add an intercept vector

Check the dimensions of X and y.

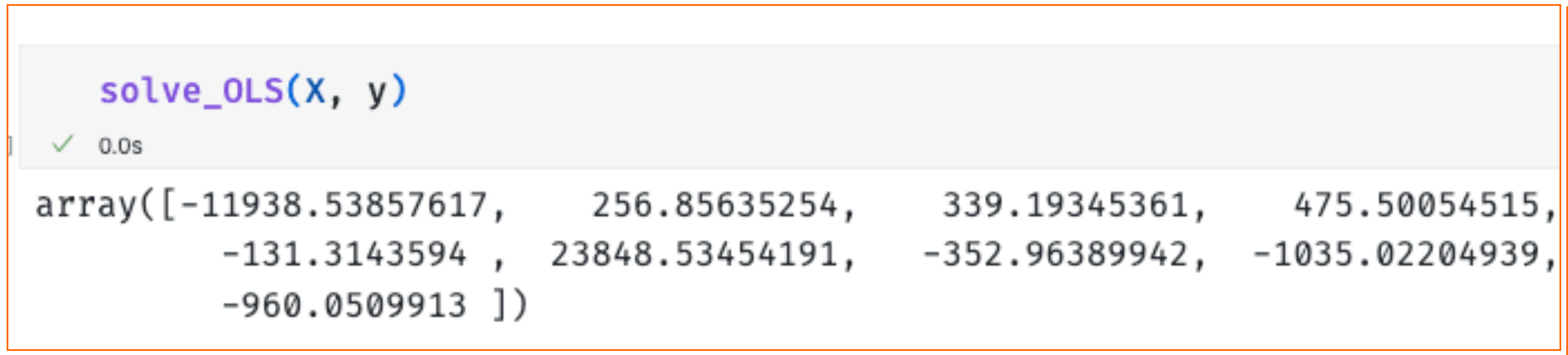
```
print("X shape: ", X.shape)
display(X)
print("y shape: ", y.shape)
display(y)
```

✓ 0.0s

```
X shape:  (1338, 9)

array([[ 1.   , 19.   , 27.9  , ...,  0.   ,  0.   ,  1.   ],
       [ 1.   , 18.   , 33.77 , ...,  0.   ,  1.   ,  0.   ],
       [ 1.   , 28.   , 33.   , ...,  0.   ,  1.   ,  0.   ],
       ...,
       [ 1.   , 18.   , 36.85 , ...,  0.   ,  1.   ,  0.   ],
       [ 1.   , 21.   , 25.8  , ...,  0.   ,  0.   ,  1.   ],
       [ 1.   , 61.   , 29.07 , ...,  1.   ,  0.   ,  0.   ]])

y shape:  (1338,)

array([16884.924 ,  1725.5523,  4449.462 , ...,  1629.8335,  2007.945 ,
       29141.3603])
```

Solve the OLS Problem

Manual Solution



```
solve_OLS(X, y)
```

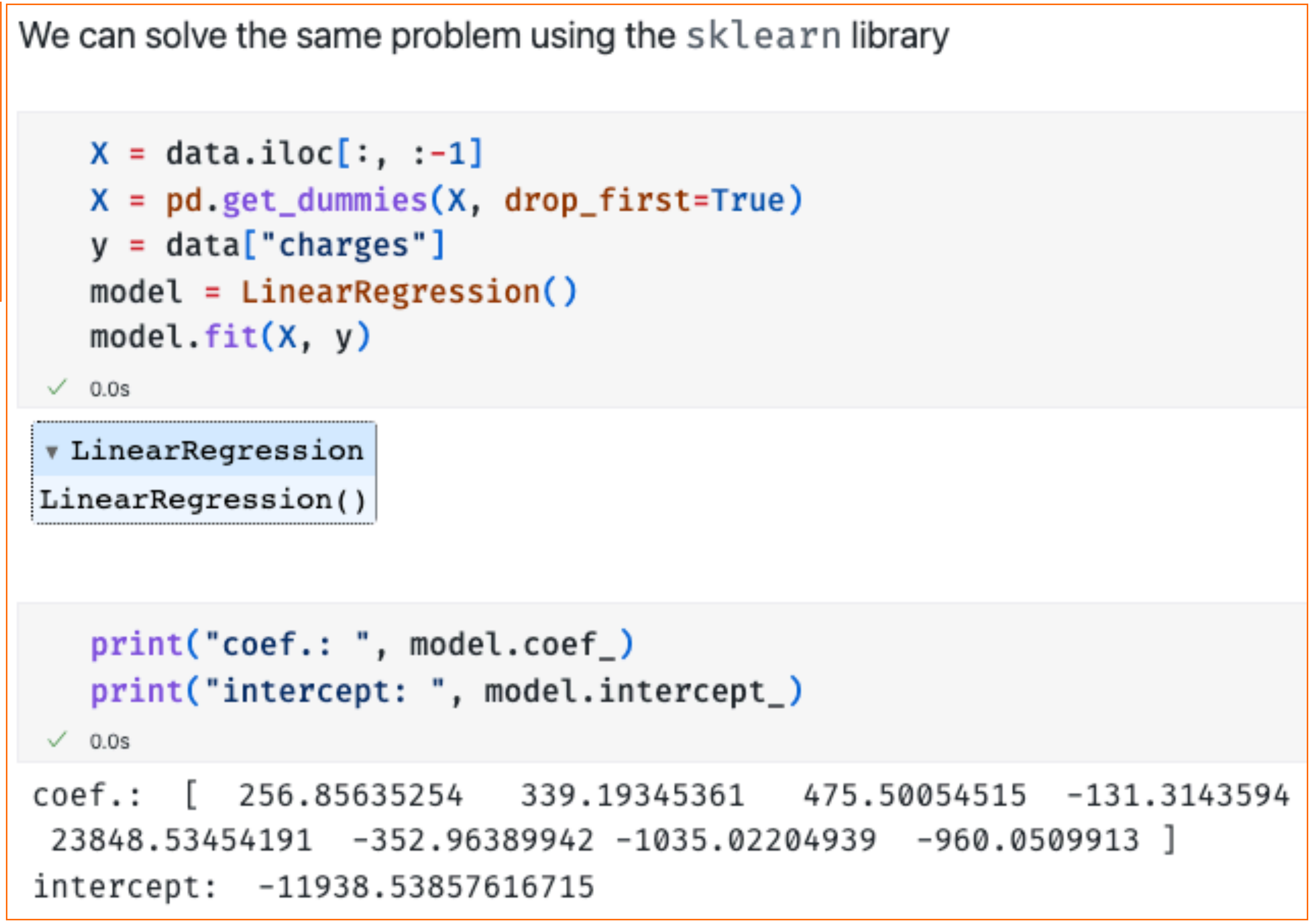
✓ 0.0s

```
array([-11938.53857617,     256.85635254,     339.19345361,     475.50054515,
         -131.3143594 ,   23848.53454191,    -352.96389942,   -1035.02204939,
         -960.0509913 ])
```

Beta vector

$$\begin{array}{ll} \begin{bmatrix} -11938.54 \\ 256.86 \\ 339.19 \\ 475.5 \\ -131.31 \\ 23848.53 \\ -352.96 \\ -1035.02 \\ -960.05 \end{bmatrix} & \begin{array}{l} \rightarrow \text{Intercept} \\ \\ \\ \rightarrow \text{Eight coefficients} \\ \\ \\ \end{array} \end{array}$$

Sklearn



We can solve the same problem using the `sklearn` library

```
X = data.iloc[:, :-1]
X = pd.get_dummies(X, drop_first=True)
y = data["charges"]
model = LinearRegression()
model.fit(X, y)
```

✓ 0.0s

```
▾ LinearRegression
LinearRegression()
```

```
print("coef.: ", model.coef_)
print("intercept: ", model.intercept_)
```

✓ 0.0s

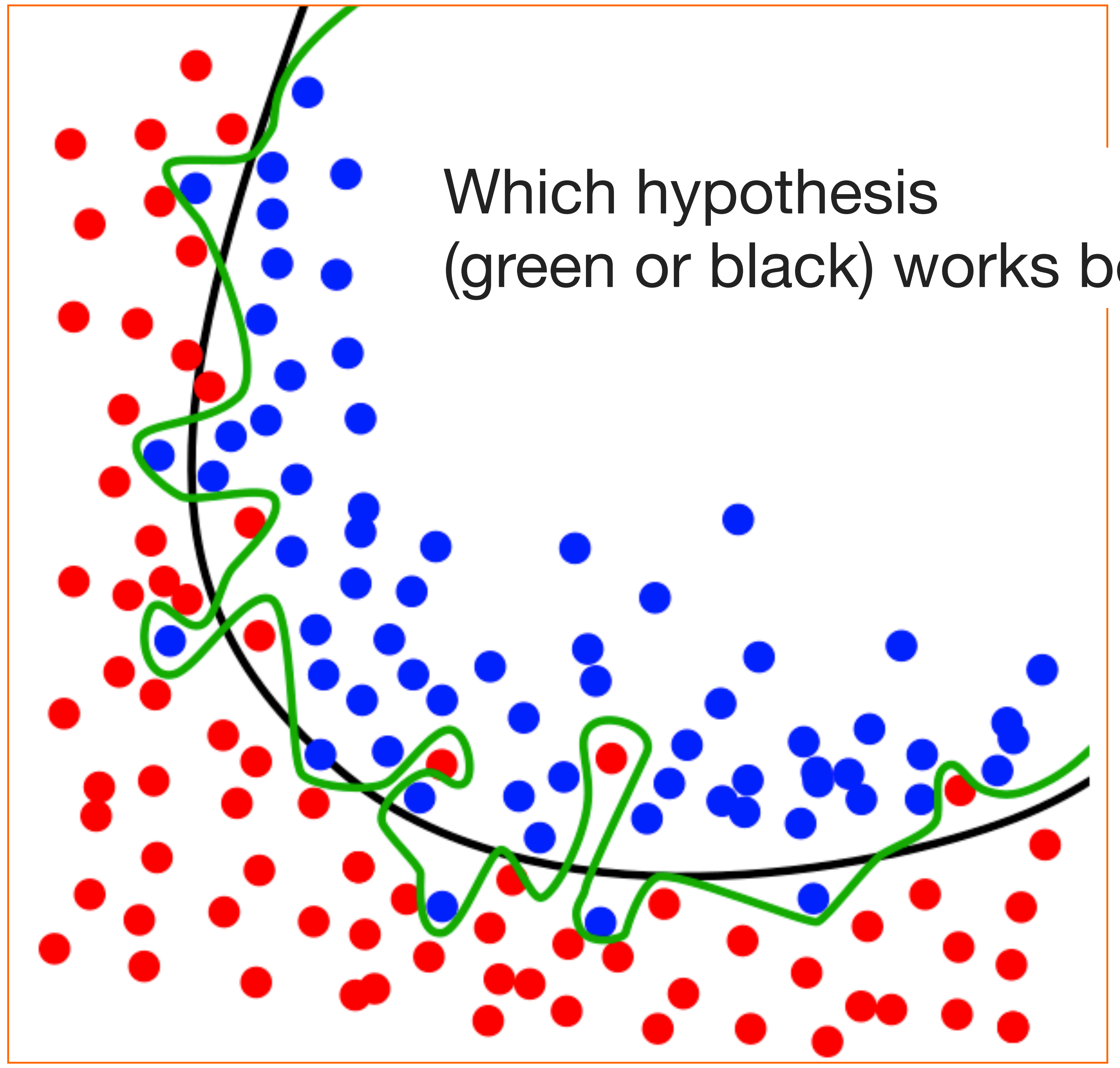
```
coef.:  [   256.85635254    339.19345361    475.50054515   -131.3143594
 23848.53454191   -352.96389942 -1035.02204939   -960.0509913 ]
intercept:  -11938.53857616715
```

Model Evaluation

Overfitting

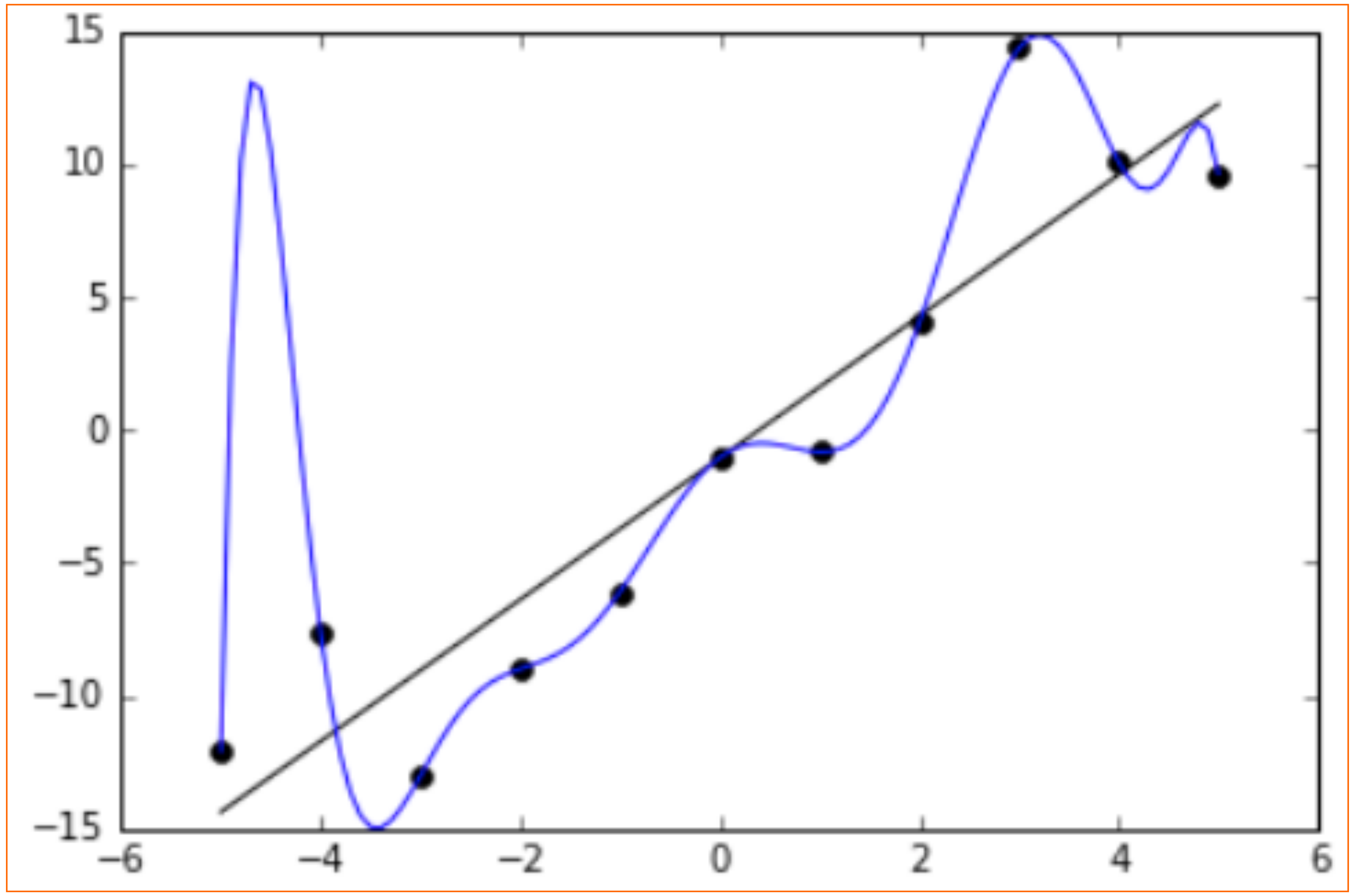
Intuitively, as we increase the number of variables in the model, the model will be more accurate. However, this is not always the case. There is a chance that a model with more variables may lose its predictability on new data. This problem is called overfitting.

Classification

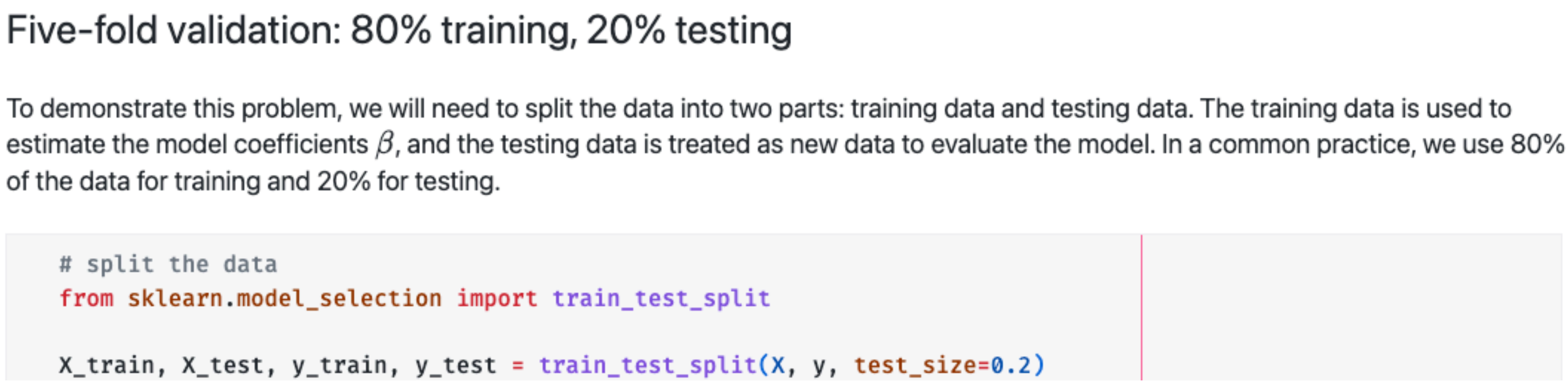


Which hypothesis
(green or black) works better?

Regression



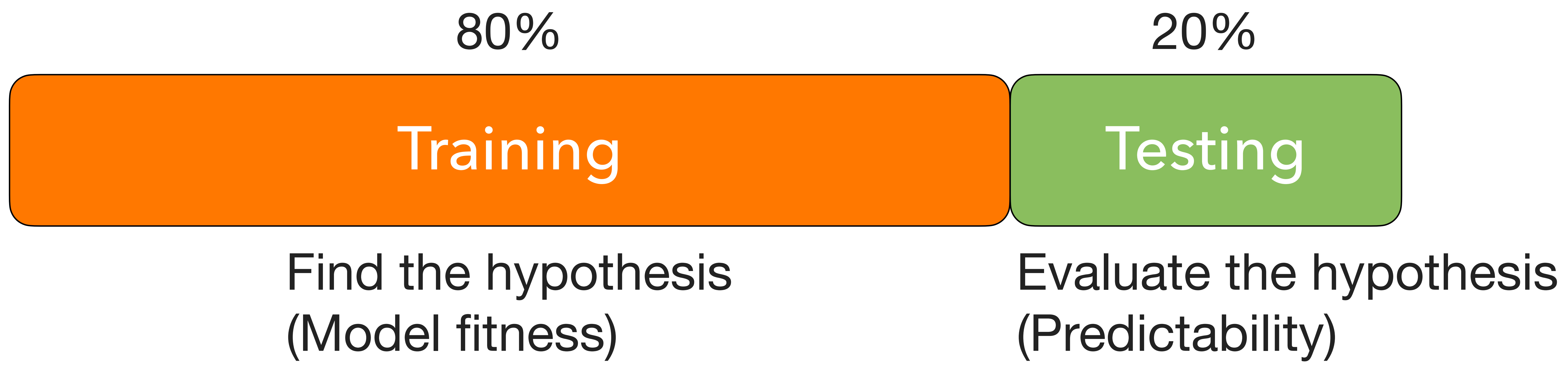
Cross Validation



Five-fold validation: 80% training, 20% testing

To demonstrate this problem, we will need to split the data into two parts: training data and testing data. The training data is used to estimate the model coefficients β, and the testing data is treated as new data to evaluate the model. In a common practice, we use 80% of the data for training and 20% for testing.

```
# split the data
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

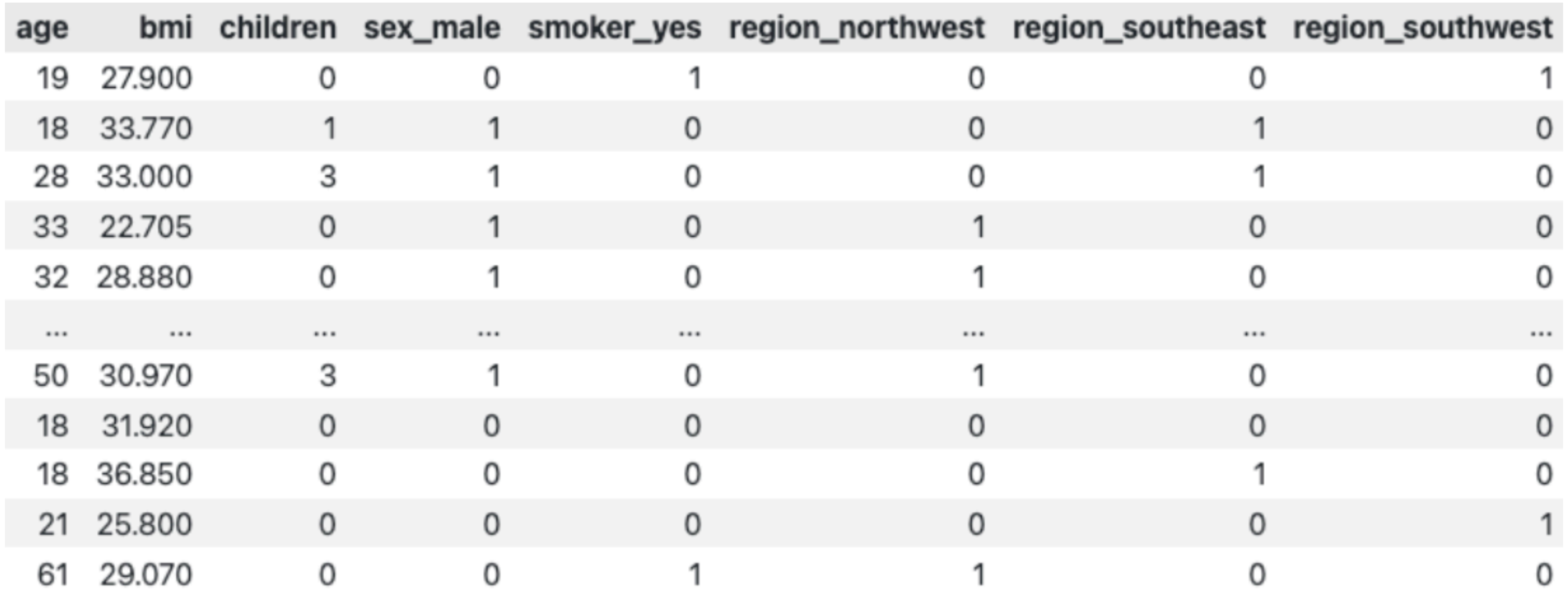


Model Selection

Do we need all the available variables?

We can design several models with different combinations of variables:

- Model A, full model: all eight variables
- Model B, reduced model: `age`, `bmi`, `children`, `smoker_yes`, `region_southeast`, `region_southwest`
- Model C, reduced model: `age`, `bmi`, `smoker_yes`, `region_southeast`
- Model D, reduced model: `age`, `smoker_yes`

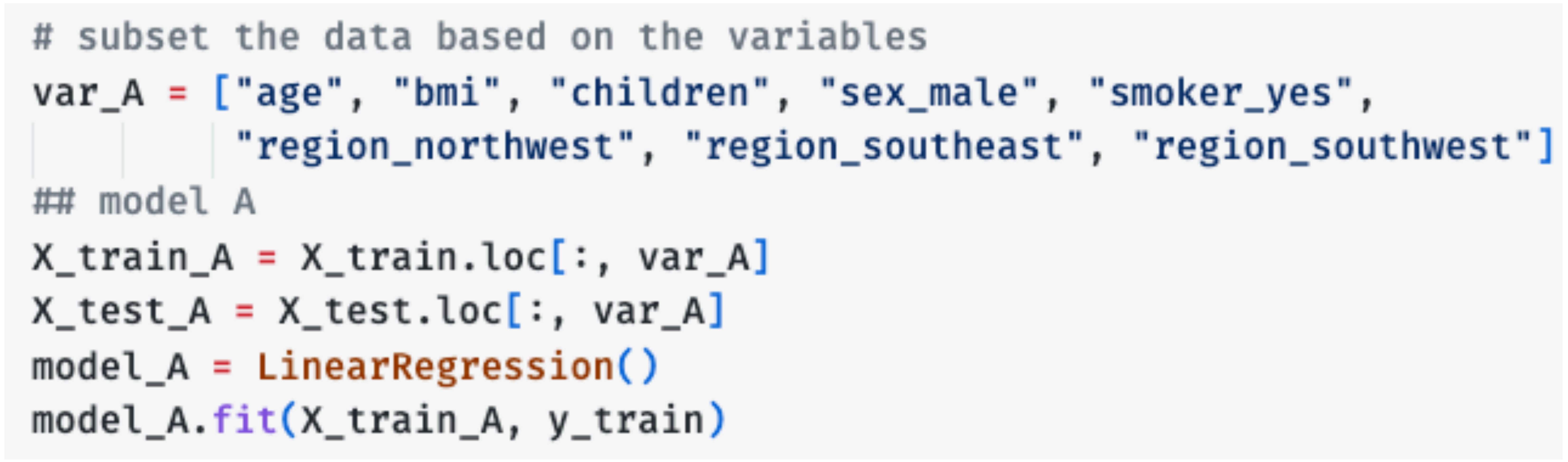


age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	region_southwest
19	27.900	0	0	1	0	0	1
18	33.770	1	1	0	0	1	0
28	33.000	3	1	0	0	1	0
33	22.705	0	1	0	1	0	0
32	28.880	0	1	0	1	0	0
...	...	...	...	...	...	...	...
50	30.970	3	1	0	1	0	0
18	31.920	0	0	0	0	0	0
18	36.850	0	0	0	0	1	0
21	25.800	0	0	0	0	0	1
61	29.070	0	0	1	1	0	0

Model Selection

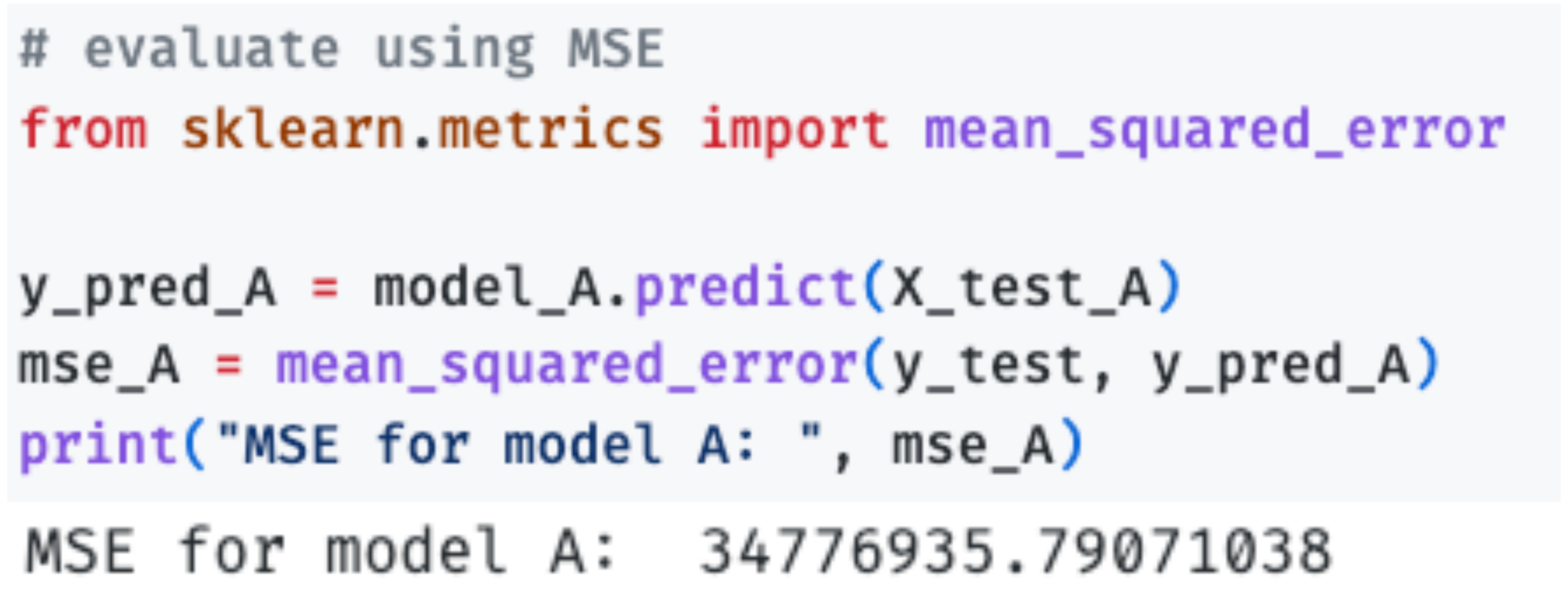
Python implementation

Fit the model (training)



```
# subset the data based on the variables
var_A = ["age", "bmi", "children", "sex_male", "smoker_yes",
         "region_northwest", "region_southeast", "region_southwest"]
## model A
X_train_A = X_train.loc[:, var_A]
X_test_A = X_test.loc[:, var_A]
model_A = LinearRegression()
model_A.fit(X_train_A, y_train)
```

Predict on the new data (testing)



```
# evaluate using MSE
from sklearn.metrics import mean_squared_error

y_pred_A = model_A.predict(X_test_A)
mse_A = mean_squared_error(y_test, y_pred_A)
print("MSE for model A: ", mse_A)
```

```
MSE for model A:  34776935.79071038
```

Iterate the Process

Iterate the validation for 100 times

The evaluation results can be biased due to the data splitting. To reduce the bias, we can repeat the splitting and evaluation for multiple times. In this example, we will repeat the splitting and evaluation for 100 times.

We need to define several functions to reduce the code repetition:

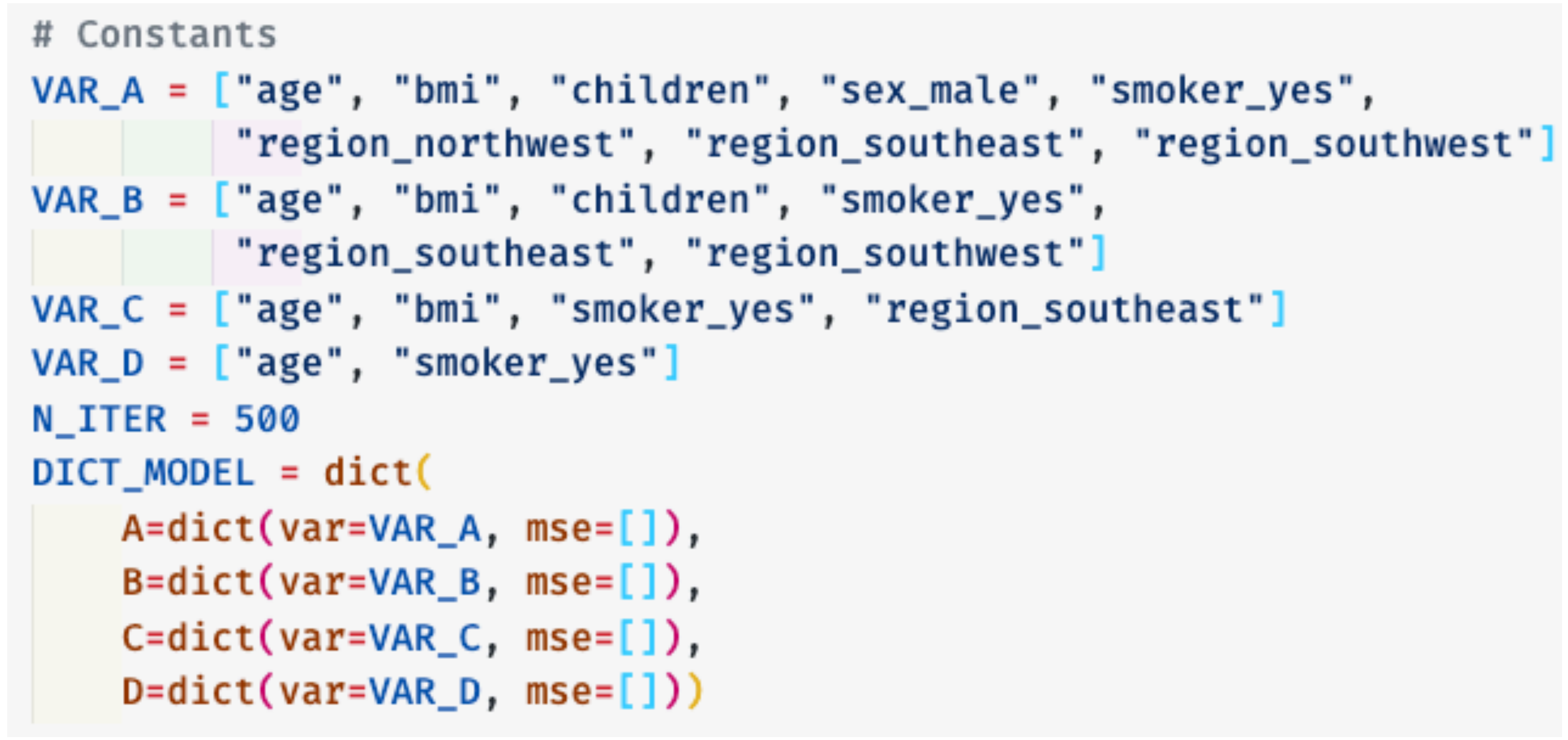
- `subset_X()`: subset the X matrix based on the variable names
- `split_data()`: split the data into training and testing
- `fit_eval_model()`: fit and evaluate the model, return MSE as the evaluation metric

```python
def subset_X(data, var):
    X = data.loc[:, var]
    return X
```

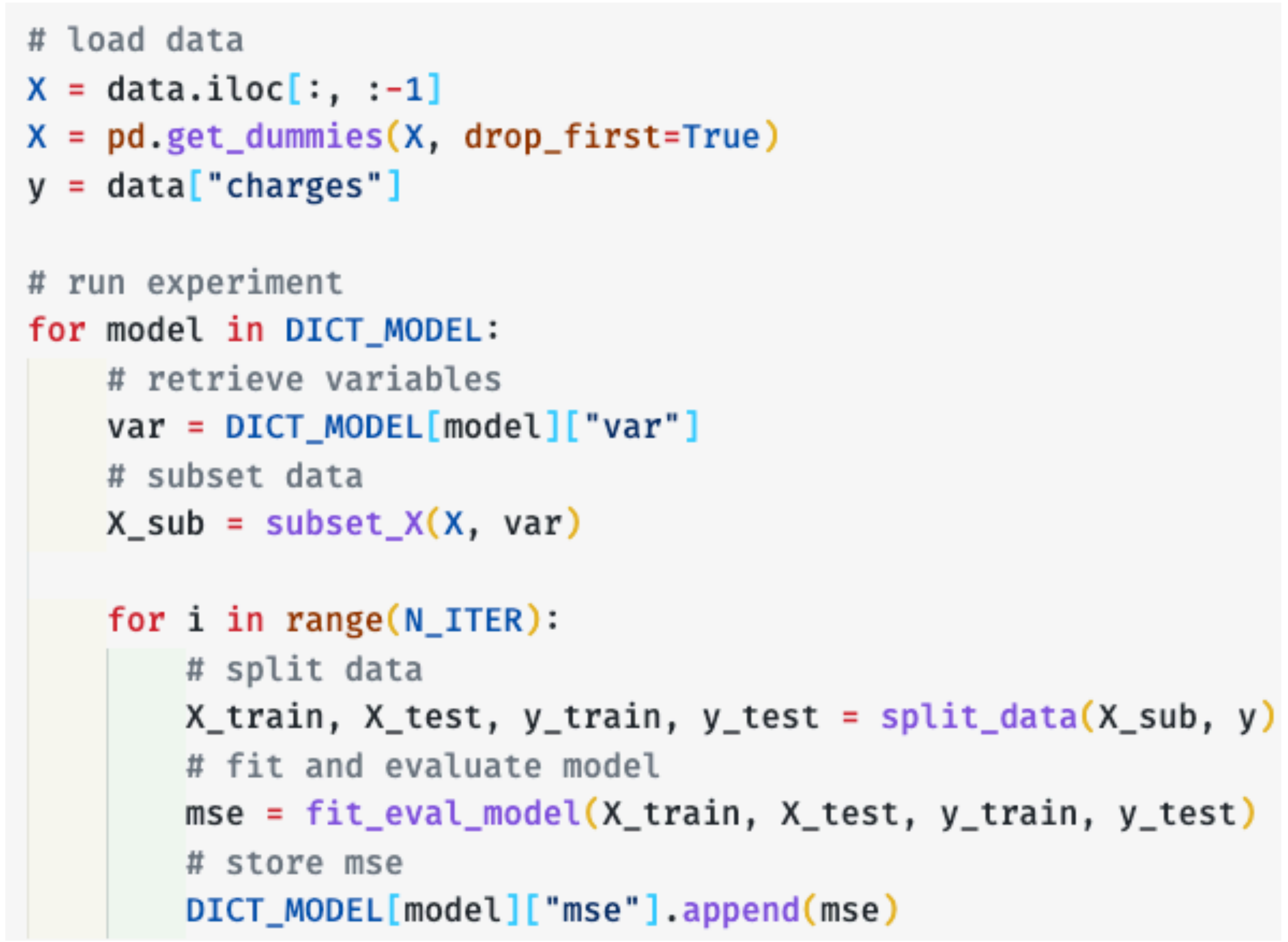
```python
def split_data(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    return X_train, X_test, y_train, y_test
```

```python
def fit_eval_model(X_train, X_test, y_train, y_test):
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    return mse
```

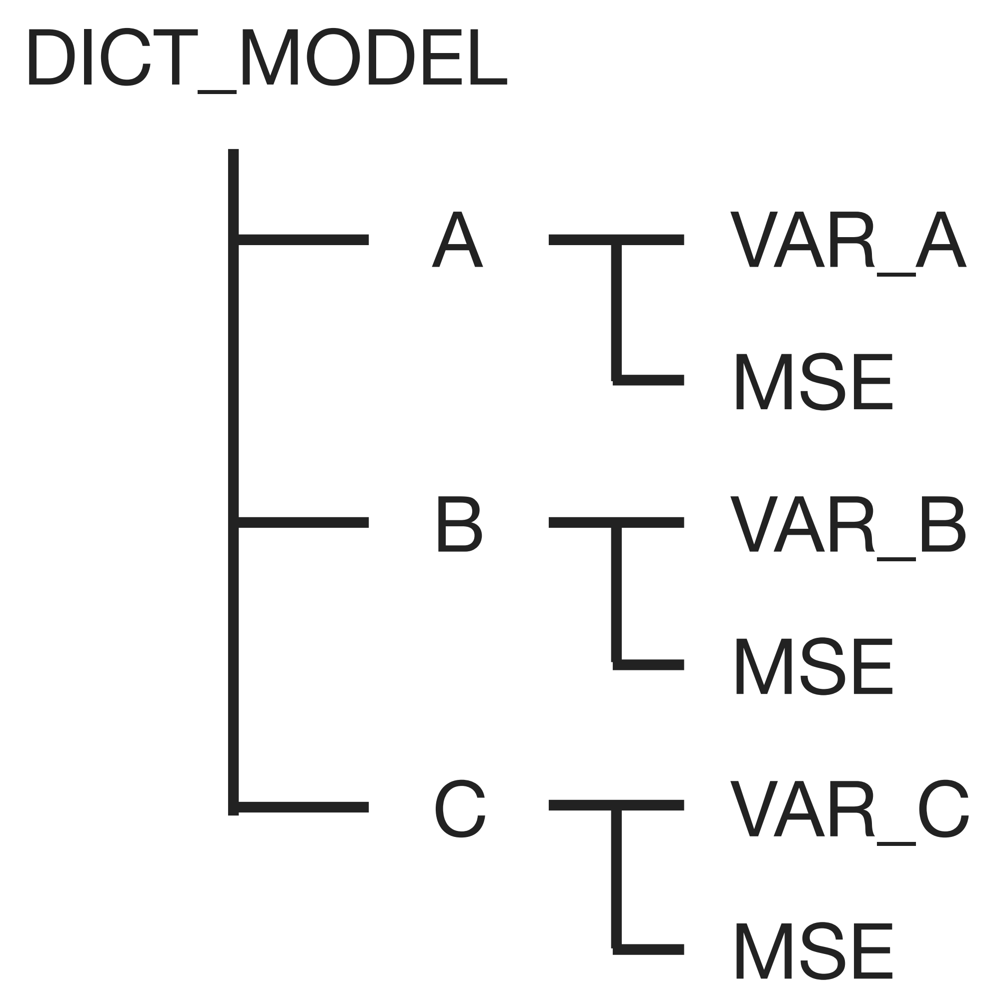
Iterate the Process



```
# Constants
VAR_A = ["age", "bmi", "children", "sex_male", "smoker_yes",
         "region_northwest", "region_southeast", "region_southwest"]
VAR_B = ["age", "bmi", "children", "smoker_yes",
         "region_southeast", "region_southwest"]
VAR_C = ["age", "bmi", "smoker_yes", "region_southeast"]
VAR_D = ["age", "smoker_yes"]
N_ITER = 500
DICT_MODEL = dict(
    A=dict(var=VAR_A, mse=[]),
    B=dict(var=VAR_B, mse=[]),
    C=dict(var=VAR_C, mse=[]),
    D=dict(var=VAR_D, mse=[]))
```



```
# load data
X = data.iloc[:, :-1]
X = pd.get_dummies(X, drop_first=True)
y = data["charges"]

# run experiment
for model in DICT_MODEL:
    # retrieve variables
    var = DICT_MODEL[model]["var"]
    # subset data
    X_sub = subset_X(X, var)

    for i in range(N_ITER):
        # split data
        X_train, X_test, y_train, y_test = split_data(X_sub, y)
        # fit and evaluate model
        mse = fit_eval_model(X_train, X_test, y_train, y_test)
        # store mse
        DICT_MODEL[model]["mse"].append(mse)
```

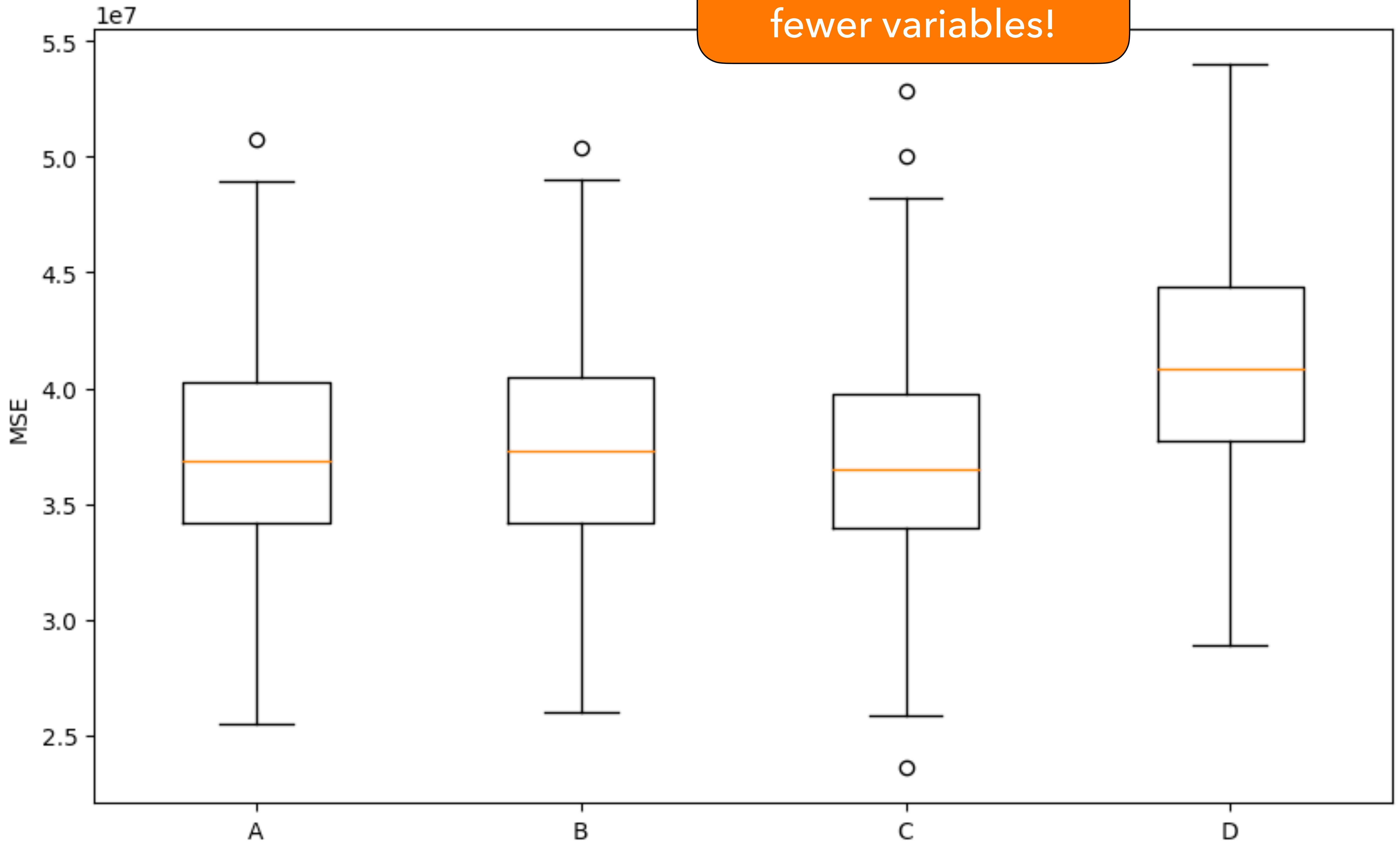


Iterate the Process

We can design several models with different combinations of variables:

- Model A, full model: all eight variables
- Model B, reduced model: `age`, `bmi`, `children`, `smoker_yes`, `region_southeast`, `region_southwest`
- Model C, reduced model: `age`, `bmi`, `smoker_yes`, `region_southeast`
- Model D, reduced model: `age`, `smoker_yes`

Smaller error with fewer variables!



AIC and BIC

There are two common model selection criteria: Akalke's Information Criterio (AIC) and Bayesian Information Criterion (BIC). The idea of AIC and BIC is to balance the goodness of fit and the complexity of the model. The lower the AIC and BIC, the better the model. The AIC and BIC are defined as:

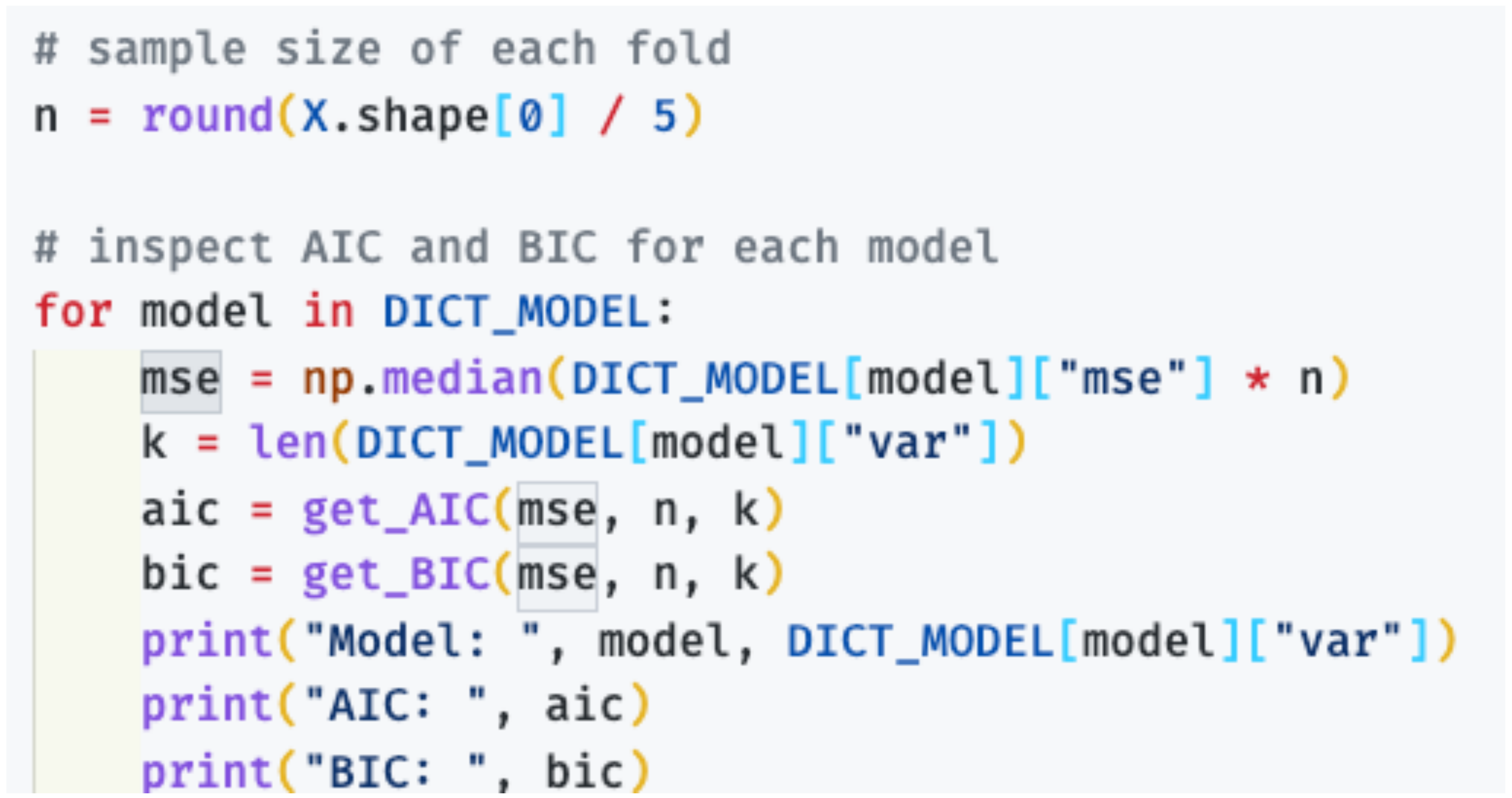
$$AIC = -2\log(L) + 2k$$
$$BIC = -2\log(L) + k\log(n)$$

where L is the likelihood of the model, k is the number of parameters, and n is the number of observations. The difference between AIC and BIC is that BIC penalizes the model complexity more than AIC. Empirically, we can replace L with the residual sum of squares (RSS):

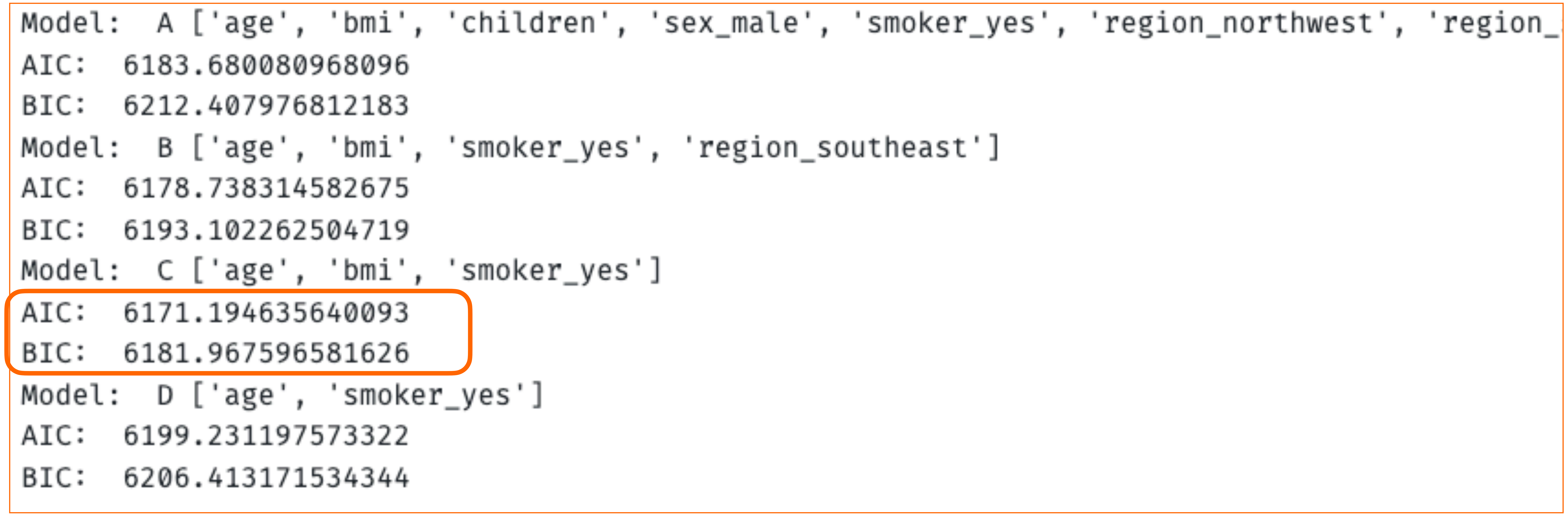
$$AIC = n\log(RSS) + 2k$$
$$BIC = n\log(RSS) + k\log(n)$$

```python
def get_AIC(MSE, n, k):
    RSS = MSE * n
    AIC = n * np.log(RSS) + 2 * k
    return AIC

def get_BIC(MSE, n, k):
    RSS = MSE * n
    BIC = n * np.log(RSS) + k * np.log(n)
    return BIC
```

AIC and BIC



```python
# sample size of each fold
n = round(X.shape[0] / 5)

# inspect AIC and BIC for each model
for model in DICT_MODEL:
    mse = np.median(DICT_MODEL[model]["mse"] * n)
    k = len(DICT_MODEL[model]["var"])
    aic = get_AIC(mse, n, k)
    bic = get_BIC(mse, n, k)
    print("Model: ", model, DICT_MODEL[model]["var"])
    print("AIC: ", aic)
    print("BIC: ", bic)
```



```
Model:  A ['age', 'bmi', 'children', 'sex_male', 'smoker_yes', 'region_northwest', 'region_
AIC:  6183.680080968096
BIC:  6212.407976812183
Model:  B ['age', 'bmi', 'smoker_yes', 'region_southeast']
AIC:  6178.738314582675
BIC:  6193.102262504719
Model:  C ['age', 'bmi', 'smoker_yes']
AIC:  6171.194635640093
BIC:  6181.967596581626
Model:  D ['age', 'smoker_yes']
AIC:  6199.231197573322
BIC:  6206.413171534344
```