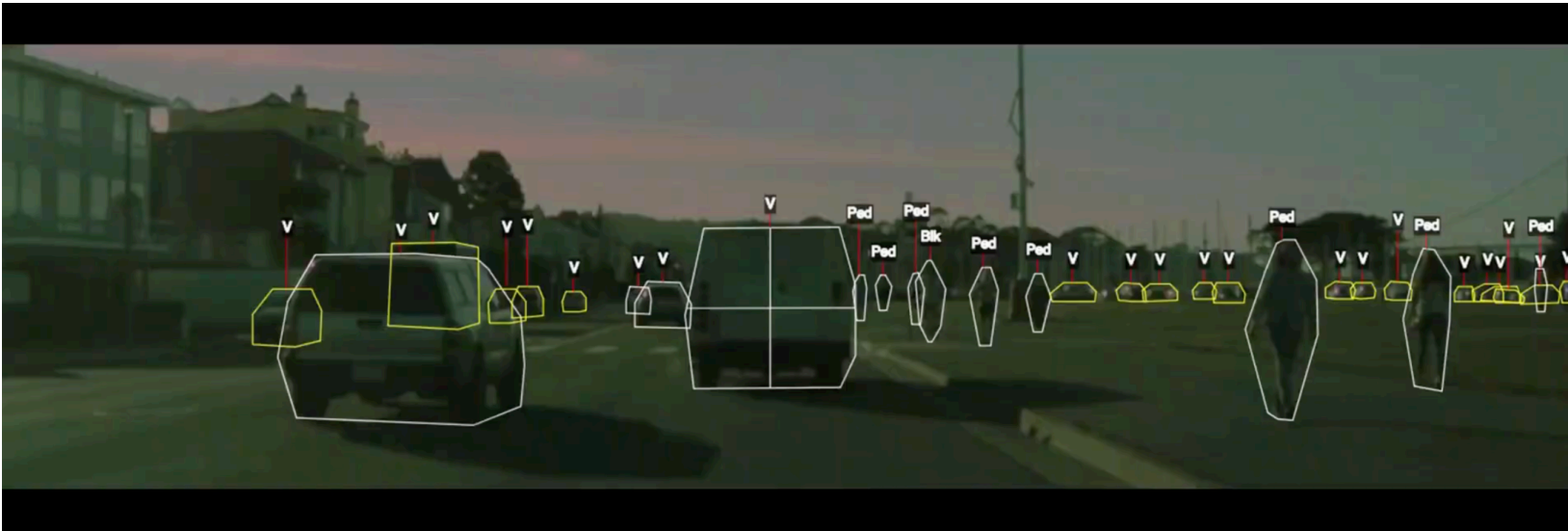


# **Lecture 2: Supervised Machine Learning**

# Recall: Supervised Learning

The most common approach to machine learning is supervised learning.



1. First, we collect a dataset of labeled training examples.
2. We train a model to output accurate predictions on this dataset.
3. When the model sees new, similar data, it will also be accurate.

# Part 1: An Example of a Supervised Machine Learning Problem

Let's start by dissecting a simple example of a supervised learning problem.

# A Recipe for Applying Supervised Learning

To apply supervised learning, we define a dataset and a learning algorithm.

$$\text{Dataset} + \text{Learning Algorithm} \rightarrow \text{Predictive Model}$$

The output is a predictive model that maps inputs to targets. For instance, it can predict targets on new inputs.

# A Supervised Learning Dataset

Let's start with a simple example of a supervised learning problem: predicting diabetes risk.

We start with a dataset of diabetes patients.

- **Input:** For each patient we have access to their BMI and an estimate of diabetes risk (from 0-400).
- **Output:** We are interested in understanding how BMI affects an individual's diabetes risk.

```

import numpy as np
import pandas as pd; pd.options.display.float_format = "{:,.2f}".format
from sklearn import datasets

# We will use the UCI Diabetes Dataset
# It's a toy dataset often used to demo ML algorithms.
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True, as_frame=True)

# Use only the BMI feature
diabetes_X = diabetes_X.loc[:, ['bmi']]

# The BMI is zero-centered and normalized; we recenter it for ease of presentation
diabetes_X = diabetes_X * 30 + 25

# Collect 20 data points
diabetes_X_train = diabetes_X.iloc[-20:]
diabetes_y_train = diabetes_y.iloc[-20:]

# Display some of the data points
pd.concat([diabetes_X_train, diabetes_y_train], axis=1).head()

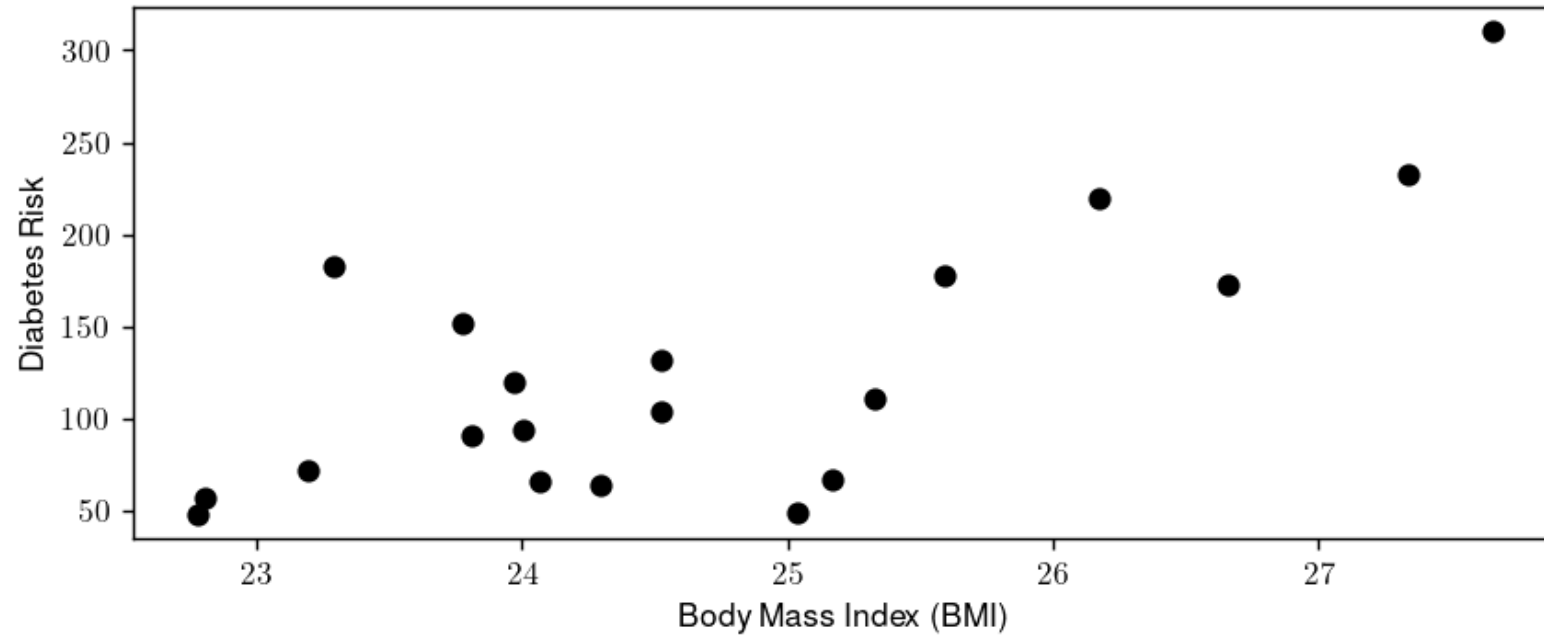
```

	bmi	target
<b>422</b>	27.34	233.00
<b>423</b>	23.81	91.00
<b>424</b>	25.33	111.00
<b>425</b>	23.78	152.00
<b>426</b>	23.97	120.00

We can also visualize this two-dimensional dataset.

```
import matplotlib.pyplot as plt
plt.rcParams.update({ "figure.figsize": [8, 3], "figure.dpi": 125, "text.usetex": True, "font.family": "Helvetica" })

plt.scatter(diabetes_X_train, diabetes_y_train, color='black')
plt.xlabel('Body Mass Index (BMI)')
plt.ylabel('Diabetes Risk');
```



# The Model Family

Let's assume that risk is a linear function of BMI. In other words, for some unknown  $\theta_0, \theta_1 \in \mathbb{R}$ , we have

$$y = \theta_1 \cdot x + \theta_0,$$

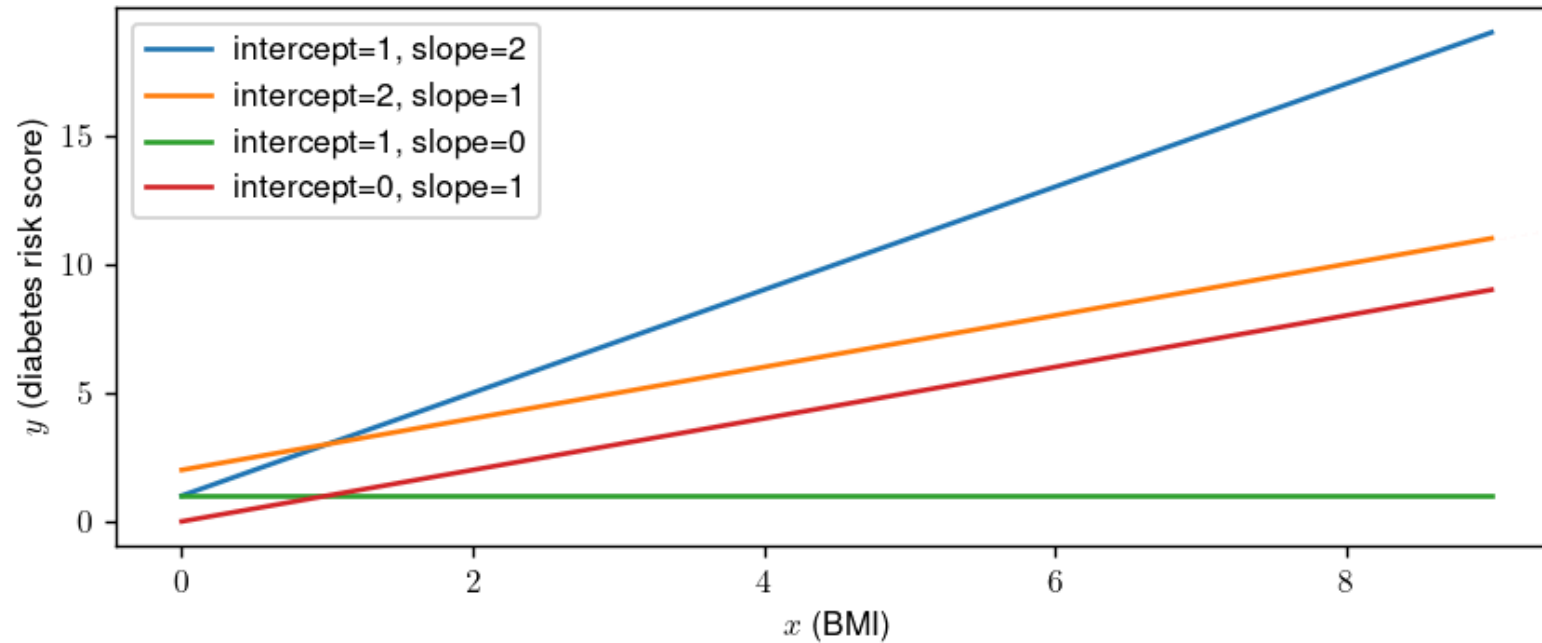
where  $x$  is the BMI (also called the independent variable), and  $y$  is the diabetes risk score (the dependent variable).

The parameters  $\theta_1, \theta_0$  are the slope and the intercept of the line relates  $x$  to  $y$ .



We can visualize this for a few values of  $\theta_1, \theta_0$ .

```
theta_list = [(1, 2), (2,1), (1,0), (0,1)]
for theta0, theta1 in theta_list:
    x = np.arange(10)
    y = theta1 * x + theta0
    plt.plot(x,y, label=f'intercept={theta0}, slope={theta1}')
plt.xlabel('$x$ (BMI)'); plt.ylabel('$y$ (diabetes risk score)'); plt.legend();
```



# A Supervised Learning Algorithm: The Optimizer

Given our assumption that  $x, y$  follow a linear relationship, the goal of a supervised learning algorithm is to find a good set of parameters consistent with the data.

This is an optimization problem.

For now, let's call the `sklearn.linear_model` library to find a  $\theta_1, \theta_0$  that fit the data well.

```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train.values)

# Make predictions on the training set
diabetes_y_train_pred_bmi_only = regr.predict(diabetes_X_train)

# The coefficients
print(f'Slope (theta1): {regr.coef_[0]:.2f}')
print(f'Intercept (theta0): {regr.intercept_:.2f}')
```

Slope (theta1): 37.38  
Intercept (theta0): -797.08

# A Supervised Learning Model (over scalar data)

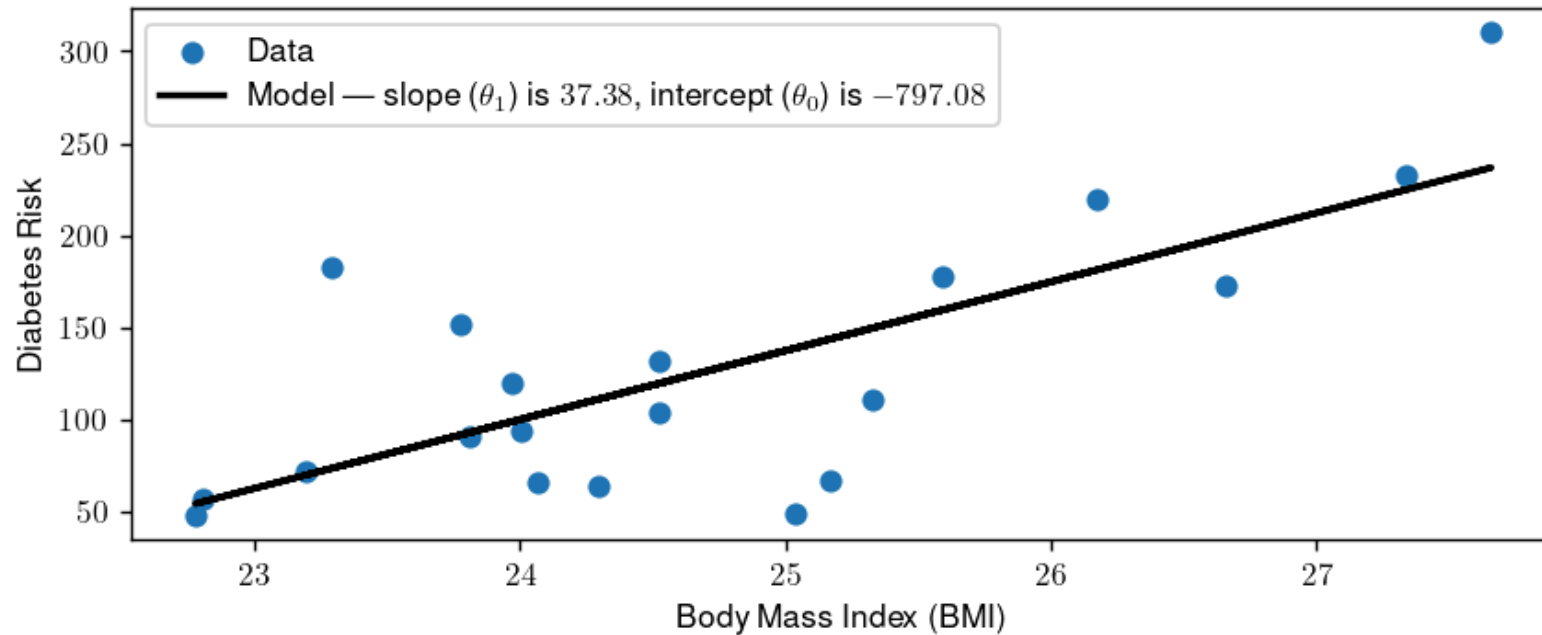
The supervised learning algorithm gave us a pair of parameters  $\theta_1^*, \theta_0^*$ . These define the *predictive model*  $f^*$ , defined as

$$f(x) = \theta_1^* \cdot x + \theta_0^*,$$

where again  $x$  is the BMI, and  $y$  is the diabetes risk score.

We can visualize the linear model that best fits our data.

```
plt.xlabel('Body Mass Index (BMI)')
plt.ylabel('Diabetes Risk')
plt.scatter(diabetes_X_train, diabetes_y_train, label='Data')
label = fr'Model --- slope ( $\theta_1$ ) is ${regr.coef_[0]:.2f}$, intercept ( $\theta_0$ ) is ${regr.intercept_:.2f}$'
plt.plot(diabetes_X_train, diabetes_y_train_pred_bmi_only, color='black', linewidth=2,
         label=label)
plt.legend();
```



# Making New Predictions

Given a new dataset of patients with a known BMI, we can use this model to estimate their diabetes risk.

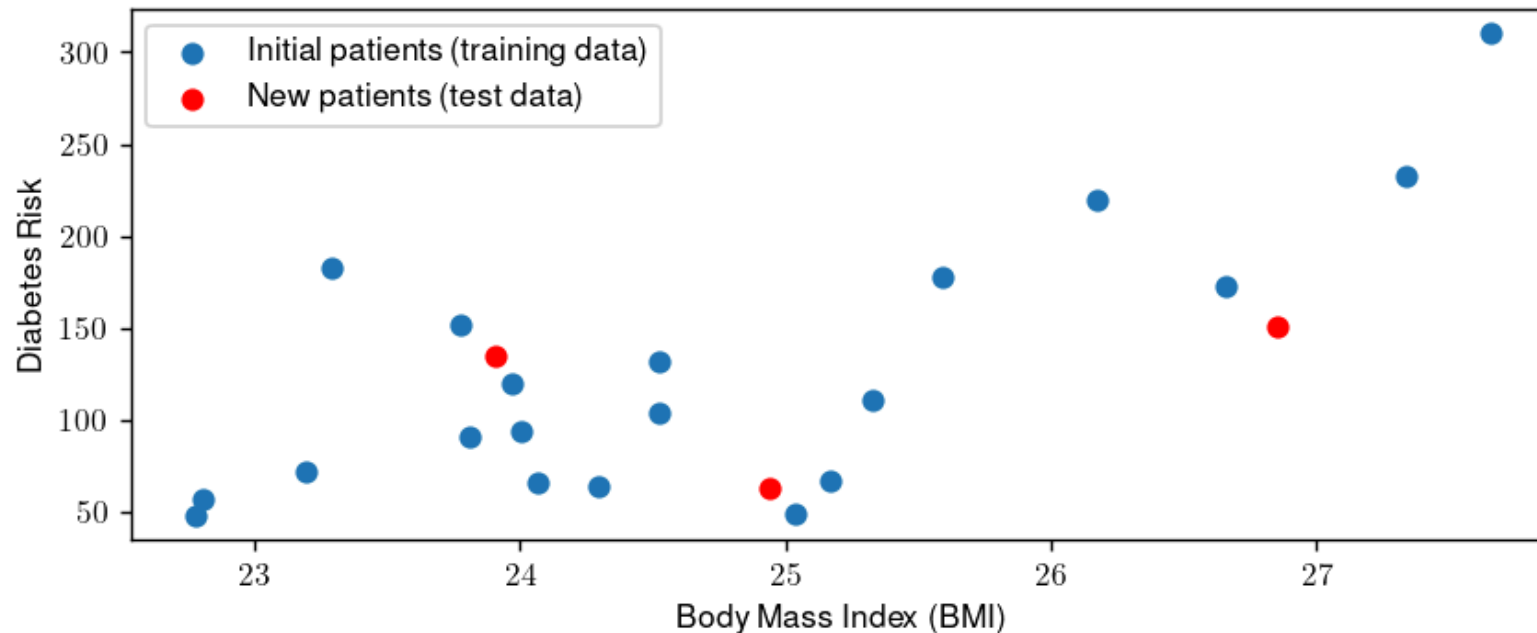
Given an  $x_{\text{new}}$ , we can output prediction  $y_{\text{new}}$  as

$$y_{\text{new}} = f(x_{\text{new}}) = \theta_1^* \cdot x_{\text{new}} + \theta_0.$$

Let's start by loading more data. We will load three new patients (shown in red below) that we haven't seen before.

```
# Collect 3 data points
test_indices = [0,4,7]
diabetes_X_test = diabetes_X.iloc[test_indices]
diabetes_y_test = diabetes_y.iloc[test_indices]

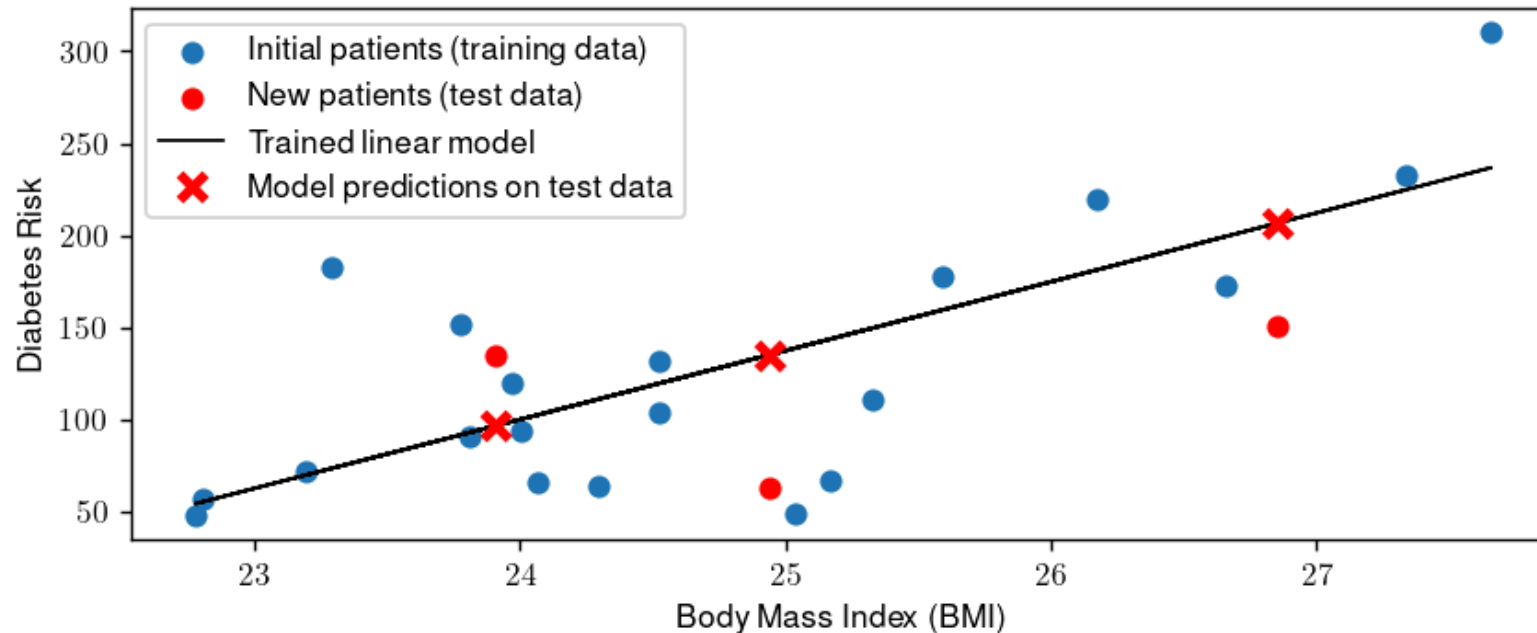
plt.scatter(diabetes_X_train, diabetes_y_train)
plt.scatter(diabetes_X_test, diabetes_y_test, color='red')
plt.xlabel('Body Mass Index (BMI)'); plt.ylabel('Diabetes Risk'); plt.legend(['Initial patients (training data)', 'New patients
```



Our linear model provides an estimate of the diabetes risk for these patients.

```
# generate predictions on the new patients
diabetes_y_test_pred_bmi_only = regr.predict(diabetes_X_test)

# visualize the results
plt.xlabel('Body Mass Index (BMI)')
plt.ylabel('Diabetes Risk')
plt.scatter(diabetes_X_train, diabetes_y_train)
plt.scatter(diabetes_X_test, diabetes_y_test, color='red', marker='o')
plt.plot(diabetes_X_train, diabetes_y_train_pred_bmi_only, color='black', linewidth=1)
plt.plot(diabetes_X_test, diabetes_y_test_pred_bmi_only, 'x', color='red', mew=3, markersize=8)
plt.legend(['Initial patients (training data)', 'New patients (test data)',
           "Trained linear model", "Model predictions on test data"]);
```

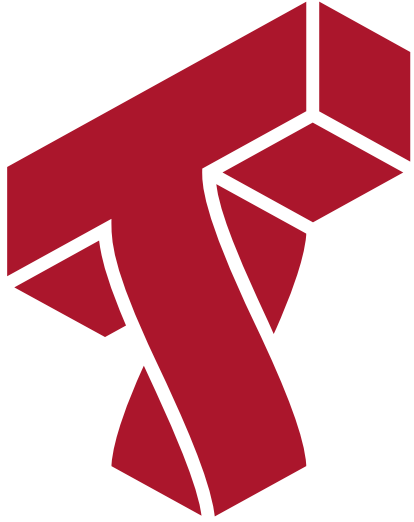




# Applications of Supervised Learning

Many of the most important applications of machine learning are supervised:

- Classifying medical images.
- Translating between pairs of languages.
- Detecting objects in a self-driving car.



## Part 2: Anatomy of a Supervised Learning Problem: The Dataset

We have seen an example of supervised machine learning for making predictions from a scalar value (the BMI).

Let's now examine more closely the components of a supervised learning problem,

# Recall: Three Components of a Supervised Machine Learning Problem

To apply supervised learning, we define a dataset and a learning algorithm.

$$\text{Dataset} + \text{Learning Algorithm} \rightarrow \text{Predictive Model}$$

The output is a predictive model that maps inputs to targets. For instance, it can predict targets on new inputs.

# A Supervised Learning Dataset

Let's dive deeper into what's a supervised learning dataset.

We will again use the UCI Diabetes Dataset as our example.

The UCI dataset contains many additional data columns besides `bmi`, including age, sex, and blood pressure. We can ask `sklearn` to give us more information about this dataset.

```
# Load the diabetes dataset
from sklearn import datasets
diabetes = datasets.load_diabetes(as_frame=True)
print(diabetes.DESCR)
```

```
.. _diabetes_dataset:
```

```
Diabetes dataset
-----
```

```
Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.
```

```
**Data Set Characteristics:**
```

```
:Number of Instances: 442
```

```
:Number of Attributes: First 10 columns are numeric predictive values
```

```
:Target: Column 11 is a quantitative measure of disease progression one year after baseline
```

```
:Attribute Information:
```

- age age in years
- sex
- bmi body mass index

# A Supervised Learning Dataset: Notation

We say that a training dataset of size  $n$  (e.g.,  $n$  patients) is a set

$$\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid i = 1, 2, \dots, n\}$$

Each  $x^{(i)}$  denotes an input (e.g., the measurements for patient  $i$ ), and each  $y^{(i)} \in \mathcal{Y}$  is a target (e.g., the diabetes risk).

Together,  $(x^{(i)}, y^{(i)})$  form a *training example*.

We can look at the diabetes dataset in this form.

```
# Load the diabetes dataset
diabetes_X, diabetes_y = diabetes.data, diabetes.target

# Print part of the dataset
diabetes_X.head()
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6
0	0.04	0.05	0.06	0.02	-0.04	-0.03	-0.04	-0.00	0.02	-0.02
1	-0.00	-0.04	-0.05	-0.03	-0.01	-0.02	0.07	-0.04	-0.07	-0.09
2	0.09	0.05	0.04	-0.01	-0.05	-0.03	-0.03	-0.00	0.00	-0.03
3	-0.09	-0.04	-0.01	-0.04	0.01	0.02	-0.04	0.03	0.02	-0.01
4	0.01	-0.04	-0.04	0.02	0.00	0.02	0.01	-0.00	-0.03	-0.05

# Training Dataset: Inputs

More precisely, an input  $x^{(i)} \in \mathcal{X}$  is a  $d$ -dimensional vector of the form

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix}$$

For example, it could be the values of the  $d$  features for patient  $i$ .

The set  $\mathcal{X}$  is called the feature space. Often, we have,  $\mathcal{X} = \mathbb{R}^d$ .



Let's look at data for one patient.

```
diabetes_X.iloc[0]
```

```
age    0.04
sex    0.05
bmi    0.06
bp     0.02
s1    -0.04
s2    -0.03
s3    -0.04
s4    -0.00
s5     0.02
s6    -0.02
Name: 0, dtype: float64
```

# Training Dataset: Attributes

We refer to the numerical variables describing the patient as *attributes*. Examples of attributes include:

- The age of a patient.
- The patient's gender.
- The patient's BMI.

Note that these attributes in the above example have been mean-centered at zero and re-scaled to have a variance of one.

# Training Dataset: Features

Often, an input object has many attributes, and we want to use these attributes to define more complex descriptions of the input.

- Is the patient old and a man? (Useful if old men are at risk).
- Is the BMI above the obesity threshold?

We call these custom attributes *features*.

Let's create an "old man" feature.

```
diabetes_X['old_man'] = (diabetes_X['sex'] > 0) & (diabetes_X['age'] > 0.05)
diabetes_X.head()
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	old_man
0	0.04	0.05	0.06	0.02	-0.04	-0.03	-0.04	-0.00	0.02	-0.02	False
1	-0.00	-0.04	-0.05	-0.03	-0.01	-0.02	0.07	-0.04	-0.07	-0.09	False
2	0.09	0.05	0.04	-0.01	-0.05	-0.03	-0.03	-0.00	0.00	-0.03	True
3	-0.09	-0.04	-0.01	-0.04	0.01	0.02	-0.04	0.03	0.02	-0.01	False
4	0.01	-0.04	-0.04	0.02	0.00	0.02	0.01	-0.00	-0.03	-0.05	False

# Training Dataset: Features

We may denote features via a function  $\phi : \mathcal{X} \rightarrow \mathbb{R}^p$  that takes an input  $x^{(i)} \in \mathcal{X}$  and outputs a  $p$ -dimensional vector

$$\phi(x^{(i)}) = \begin{bmatrix} \phi(x^{(i)})_1 \\ \phi(x^{(i)})_2 \\ \vdots \\ \phi(x^{(i)})_p \end{bmatrix}$$

We say that  $\phi(x^{(i)})$  is a *featurized* input, and each  $\phi(x^{(i)})_j$  is a *feature*.

# Features vs Attributes

In practice, the terms attribute and features are often used interchangeably. Most authors refer to  $x^{(i)}$  as a vector of features.

We will follow this convention and use the term "attribute" only when there is ambiguity between features and attributes.

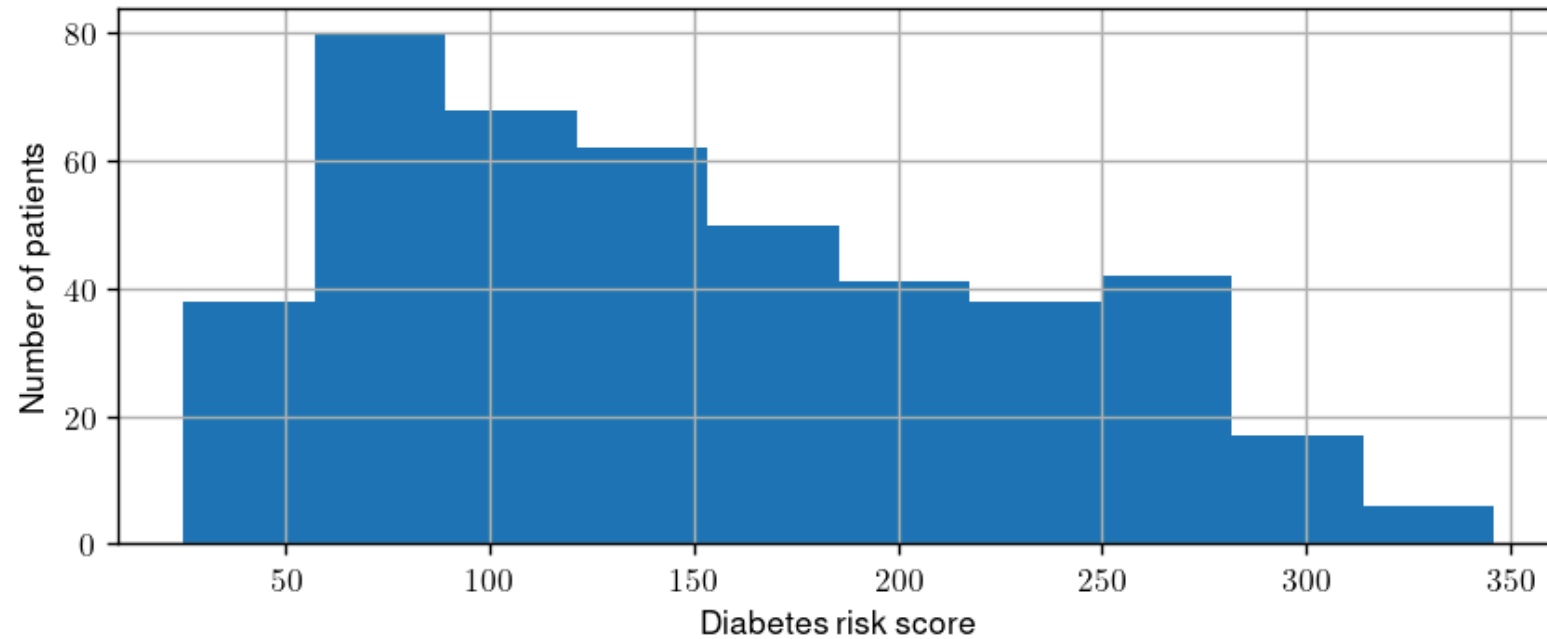
# Training Dataset: Targets

For each patient, we are interested in predicting a quantity of interest, the *target*. In our example, this is the patient's diabetes risk.

Formally, when  $(x^{(i)}, y^{(i)})$  form a *training example*, each  $y^{(i)} \in \mathcal{Y}$  is a target. We call  $\mathcal{Y}$  the target space.

We plot the distribution of risk scores below.

```
plt.xlabel('Diabetes risk score')  
plt.ylabel('Number of patients')  
diabetes_y.hist();
```

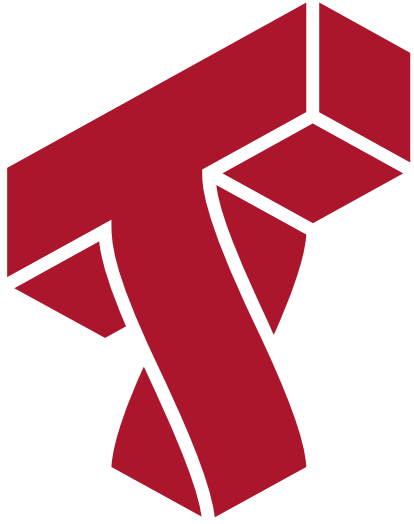




# Targets: Regression vs. Classification

We distinguish between two broad types of supervised learning problems that differ in the form of the target variable.

1. **Regression:** The target variable  $y$  is continuous. We are fitting a curve in a high-dimensional feature space that approximates the shape of the dataset.
2. **Classification:** The target variable  $y$  is discrete. Each discrete value corresponds to a *class* and we are looking for a hyperplane that separates the different classes.



# Part 3: Anatomy of a Supervised Learning Problem: The Learning Algorithm

Let's now look at what a general supervised learning algorithm looks like.

# The Components of a Supervised Machine Learning Algorithm

We can also define the high-level structure of a supervised learning algorithm as consisting of three components:

- A **model class**: the set of possible models we consider.
- An **objective** function, which defines how good a model is.
- An **optimizer**, which finds the best predictive model in the model class according to the objective function.

Let's look again at our diabetes dataset for an example.

```
# Load the diabetes dataset
diabetes = datasets.load_diabetes(as_frame=True)
diabetes_X, diabetes_y = diabetes.data, diabetes.target

# Print part of the dataset
diabetes_X.head()
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6
0	0.04	0.05	0.06	0.02	-0.04	-0.03	-0.04	-0.00	0.02	-0.02
1	-0.00	-0.04	-0.05	-0.03	-0.01	-0.02	0.07	-0.04	-0.07	-0.09
2	0.09	0.05	0.04	-0.01	-0.05	-0.03	-0.03	-0.00	0.00	-0.03
3	-0.09	-0.04	-0.01	-0.04	0.01	0.02	-0.04	0.03	0.02	-0.01
4	0.01	-0.04	-0.04	0.02	0.00	0.02	0.01	-0.00	-0.03	-0.05

# Model Class: Notation

Formally, the model (or *hypothesis*) class is a set

$$\mathcal{M} \subseteq \{f \mid f : \mathcal{X} \rightarrow \mathcal{Y}\}$$

of possible models that map input features to targets.

When the models  $f_\theta$  are parametrized by *parameters*  $\theta \in \Theta$  living in some set  $\Theta$ . Thus we can also write

$$\mathcal{M} = \{f_\theta \mid \theta \in \Theta\}.$$

# Model Class: Example with vector-valued features

One simple approach is to assume that  $x$  and  $y$  are related by a linear model of the form

$$f(x) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \dots + \theta_d \cdot x_d$$

where  $x$  is a featurized input and  $y$  is the target.

The  $\theta_j$  are the *parameters* of the model,  $\Theta = \mathbb{R}^{d+1}$ , and

$$\mathcal{M} = \{\theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \dots + \theta_d \cdot x_d \mid \theta \in \mathbb{R}^{d+1}\}$$

# Objectives: Notation

To capture this intuition, we define an *objective function* (also called a *loss function*)

$$J(f) : \mathcal{M} \rightarrow [0, \infty),$$

which describes the extent to which  $f$  "fits" the data  $\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid i = 1, 2, \dots, n\}$ .

When  $f$  is parametrized by  $\theta \in \Theta$ , the objective becomes a function  $J(\theta) : \Theta \rightarrow [0, \infty)$ .

Error-based objective functions for regression are typically lower-bounded by 0, but in general may take any real value (e.g., for classification)

# Objective: Examples

What would be some possible objective functions? We will see many, but here are a few examples:

- Mean squared error:

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^n \left( f_{\theta}(x^{(i)}) - y^{(i)} \right)^2$$

- Absolute (L1) error:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n \left| f_{\theta}(x^{(i)}) - y^{(i)} \right|$$

These are defined for a dataset  $\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid i = 1, 2, \dots, n\}$ .



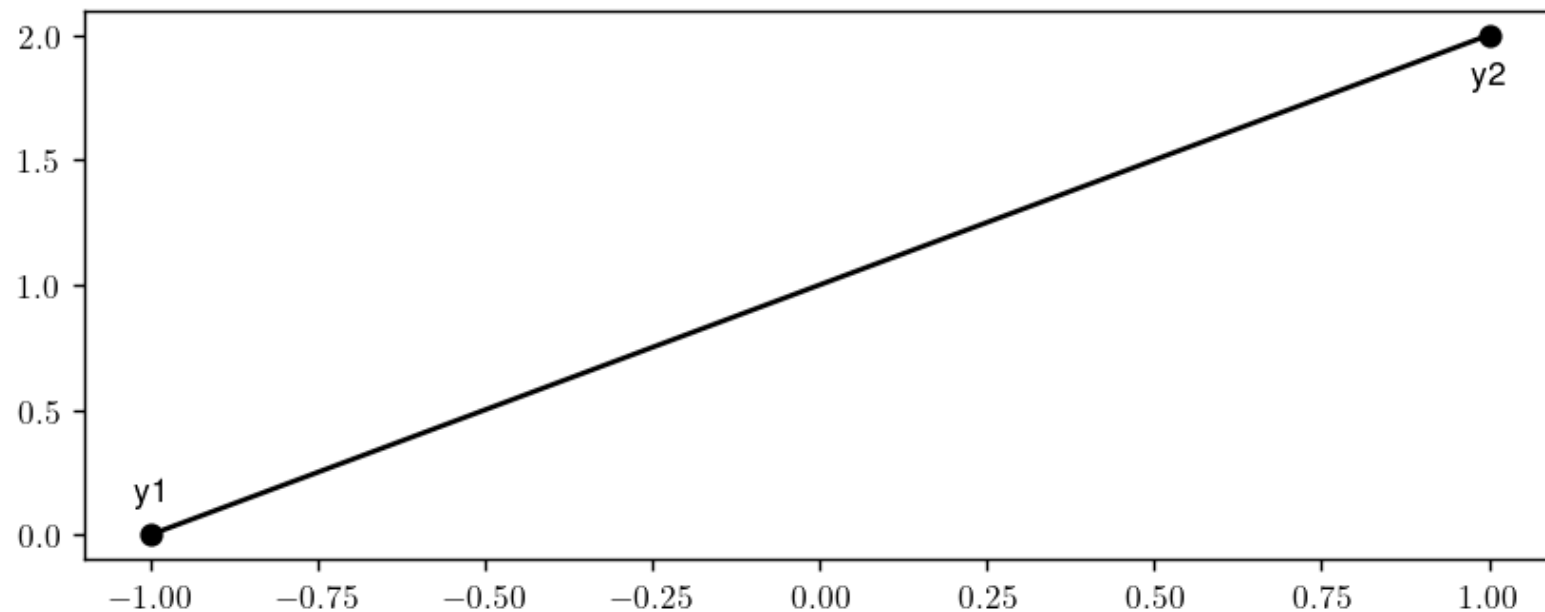
```
from sklearn.metrics import mean_squared_error, mean_absolute_error

y1 = np.array([-1, 0])
y2 = np.array([1, 2])

plt.scatter(*y1, color='k')
plt.scatter(*y2, color='k')
plt.plot([y1[0], y2[0]], [y1[1], y2[1]], color='k')
plt.annotate('y1', (y1[0], y1[1]), textcoords="offset points", xytext=(0,10), ha='center')
plt.annotate('y2', (y2[0], y2[1]), textcoords="offset points", xytext=(0,-15), ha='center')

print('Mean squared error: %.2f' % mean_squared_error(y1, y2))
print('Mean absolute error: %.2f' % mean_absolute_error(y1, y2))
```

Mean squared error: 4.00  
Mean absolute error: 2.00



# Optimizer: Notation

At a high-level an optimizer takes an objective  $J$  and a model class  $\mathcal{M}$  and finds a model  $f \in \mathcal{M}$  with the smallest value of the objective  $J$ .

$$\min_{f \in \mathcal{M}} J(f)$$

Intuitively, this is the function that bests "fits" the data on the training dataset.

When  $f$  is parametrized by  $\theta \in \Theta$ , the optimizer minimizes a function  $J(\theta)$  over all  $\theta \in \Theta$ .

# Optimizer: Example

We will see that behind the scenes, the

`sklearn.linear_models.LinearRegression` algorithm optimizes the MSE loss. When  $\mathcal{X} = \mathbb{R}$ ,  $\Theta = \mathbb{R}^2$  for the intercept and slope terms.

$$\min_{\theta \in \mathbb{R}^2} \frac{1}{2n} \sum_{i=1}^n \left( f_{\theta}(x^{(i)}) - y^{(i)} \right)^2$$

We can easily measure the quality of the fit on the training set and the test set.

Let's use `sklearn` to fit a linear model with the MSE loss to our diabetes dataset, which now takes multi-dimensional features instead of just the scalar-valued BMI.

```
# Collect 20 data points for training
diabetes_X_train = diabetes_X.iloc[-20:]
diabetes_y_train = diabetes_y.iloc[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train.values)

# Make predictions on the training set
diabetes_y_train_pred = regr.predict(diabetes_X_train)

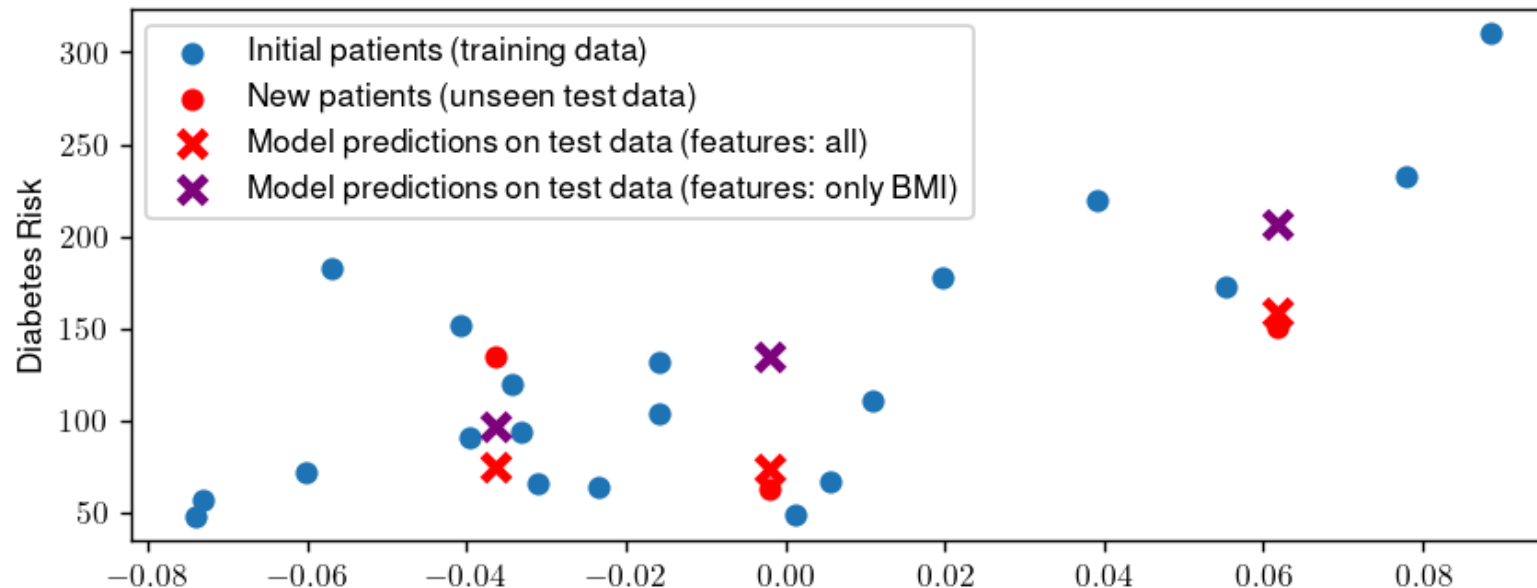
# Collect 3 data points for testing
diabetes_X_test = diabetes_X.iloc[test_indices]
diabetes_y_test = diabetes_y.iloc[test_indices]

# generate predictions on the new patients
diabetes_y_test_pred = regr.predict(diabetes_X_test)
```

Let's visualize the predictions on the same data, and compare to the model only trained on the BMI features.

```
# generate predictions on the new patients
diabetes_y_test_pred = regr.predict(diabetes_X_test)

# visualize the results
plt.xlabel('Body Mass Index (BMI)')
plt.ylabel('Diabetes Risk')
l = plt.scatter(diabetes_X_train.loc[:, ['bmi']], diabetes_y_train)
plt.scatter(diabetes_X_test.loc[:, ['bmi']], diabetes_y_test, color='red', marker='o')
plt.plot(diabetes_X_test.loc[:, ['bmi']], diabetes_y_test_pred, 'x', color='red', mew=3, markersize=8)
plt.plot(diabetes_X_test.loc[:, ['bmi']], diabetes_y_test_pred_bmi_only, 'x', color='purple', mew=3, markersize=8)
plt.legend(['Initial patients (training data)', 'New patients (unseen test data)',
           "Model predictions on test data (features: all)", "Model predictions on test data (features: only BMI)" ]);
```



# Evaluating the learned models

We lastly look at the MSE of the models we have fit in this lecture on the training and test datasets. Some trends:

1. The train MSE is lower than the test MSE (not a great predictor, but it's a hard task)
2. The model using all the features outperforms the model predicting only from the BMI
3. Both models are better than random noise

```
from sklearn.metrics import mean_squared_error

print('Training set mean squared error of the linear model using all features: %.2f'
      % mean_squared_error(diabetes_y_train, diabetes_y_train_pred))
print('Training set mean squared error of the linear model using only the BMI as a feature: %.2f'
      % mean_squared_error(diabetes_y_train, diabetes_y_train_pred_bmi_only))
print('-'*80)
print('Test set mean squared error of the linear model using all features: %.2f'
      % mean_squared_error(diabetes_y_test, diabetes_y_test_pred))
```

# Summary: Components of a Supervised Machine Learning Problem

To apply supervised learning, we define a dataset and a learning algorithm.

$$\underbrace{\text{Dataset}}_{\text{Features, Attributes, Targets}} + \underbrace{\text{Learning Algorithm}}_{\text{Model Class} + \text{Objective} + \text{Optimizer}} \rightarrow \text{Predictive Model}$$

The output is a predictive model that maps inputs to targets. For instance, it can predict targets on new inputs.

**Next class:** implementing and optimizing regression models ourselves without `sklearn`