Overview

This tutorial will focus on Linear Regression

We will include some additional topics, including:

- Basic formulation of supervised learning
- Basic linear regression model
- Train/test set split
- Evaluation of regression models

The lecture relies on the the following proficiencies and tools:

- Python programming
- Vector computations via the numpy module
- Data handling using the pandas module
- Plotting using matplotlib
- Training and using Machine Learning model via scikit-learn

You will need them only if you plan to handle these tasks yourself

Linear Regression

Our Target Problem

Let's assume we want to estimate real-estate prices in Taiwan

Loading the Data

Data for this problem is available (in csv format) from the `data` folder

In [2]:
```
!ls data
```

```
lr_test.txt  lr_train.txt  real_estate.csv  weather.csv
```

We will load the data via a Python library, called pandas

In [3]:
```
data = pd.read_csv('data/real_estate.csv', sep=',')
data.head() # Head returns the first 5 elements
```

Out[3]:

	house age	dist to MRT	#stores	latitude	longitude	price per area
0	14.8	393.2606	6	24.96172	121.53812	7.6
1	17.4	6488.0210	1	24.95719	121.47353	11.2
2	16.0	4066.5870	0	24.94297	121.50342	11.6
3	30.9	6396.2830	1	24.94375	121.47883	12.2
4	16.5	4082.0150	0	24.94155	121.50381	12.8

- The content of the csv files is a made accessible in a table-like object (`DataFrame`)

A Look at the Data

Let's have a better look at the data

In [4]:
```
data.head()
```

Out[4]:

	house age	dist to MRT	#stores	latitude	longitude	price per area
0	14.8	393.2606	6	24.96172	121.53812	7.6
1	17.4	6488.0210	1	24.95719	121.47353	11.2
2	16.0	4066.5870	0	24.94297	121.50342	11.6
3	30.9	6396.2830	1	24.94375	121.47883	12.2
4	16.5	4082.0150	0	24.94155	121.50381	12.8

- The first four columns contain quantities that easy to estimate
- ...But that's not true for the last one!

Obtaining prices required actual houses to be sold and bought

- Our goal is to use the data to learn a model
- ...That can estimate the price based on the easily available information

Input, Output, Examples, Targets

Formally, we say that

- All columns except the price represent the input x of our model
- The price represetns the output y of our model
- Each row in the table represents one data point, i.e. an example $(\hat{x}_i, \hat{y}_i)$
  - $\hat{x}_i$ is the input value for the i-th example
  - $\hat{y}_i$ is the true output value (or target) for the i-th example

Our goal is to learn a model f such that

- When we feed the input $\hat{x}_i$ of each example to it
- ...The output value $y_i = f(\hat{x}_i)$ is as close as possible to $\hat{y}_i$

This kind of setup is known in ML as supervised learning

Supervised Learning and Regression

Supervised Learning is among the most common forms of ML

Our model is a function $f(x, ; w)$ with input x and parameters w

- If the output is numeric, we speak of regression
- ...And we can define the approximation error over the exampple using, e.g.:

$$MSE(w) = \sum_{i=1}^{m} (f(x_i, ; w) - y_i)^2$$

- "MSE" stands for Mean Squared Error and it's a common error metric

Training in a (MSE) regression problem consists in solving

$$\text{argmin}_w \, MSE(w)$$

- I.e. choosing the parameters w to minimize approximation error

Supervised Learning...And Linear Regression

We speak instead of Linear Regression

...When f is defined as a linear combination of basis functions

$$f(x; w) = \sum_{i=1}^{n} w_j \phi_j(x)$$

In our case:

- Each basis function will correspond to a specific input column
- I.e. "house age", "distance to MRT", "#stores", "latitude", "longitude"

This is a very common setup in Linear Regression

Linear regression is one of the simplest supervised learning approaches

- ...But it is still a very good example!
- ...And will allow us to discuss some of the kay challenges in ML

Separating Input and Ouput

First, we separate our input and output

In [5]:
```
cols = data.columns
X = data[cols[:-1]] # all columns except the last one
display(X.head())
```

	house age	dist to MRT	#stores	latitude	longitude
0	14.8	393.2606	6	24.96172	121.53812
1	17.4	6488.0210	1	24.95719	121.47353
2	16.0	4066.5870	0	24.94297	121.50342
3	30.9	6396.2830	1	24.94375	121.47883
4	16.5	4082.0150	0	24.94155	121.50381

We will focus on predicting the logarithm of the price per area

In [6]:
```
y = np.log(data[cols[-1]]) # just the last column
```

- In practice, it's like predicting the order of magnitude

Training and Test Set

The model we learn should work well on all relevant data

Formally, the model should generalize well

- How do we check whether this is the case?
- A typical approach: partitioning our dataset

The basic idea is to separe our data in two groups

- The first group will actually be used for training
  - This will be called the training set
- The second group will be used only for model evaluation
  - This will be called the test set (or holdout set)

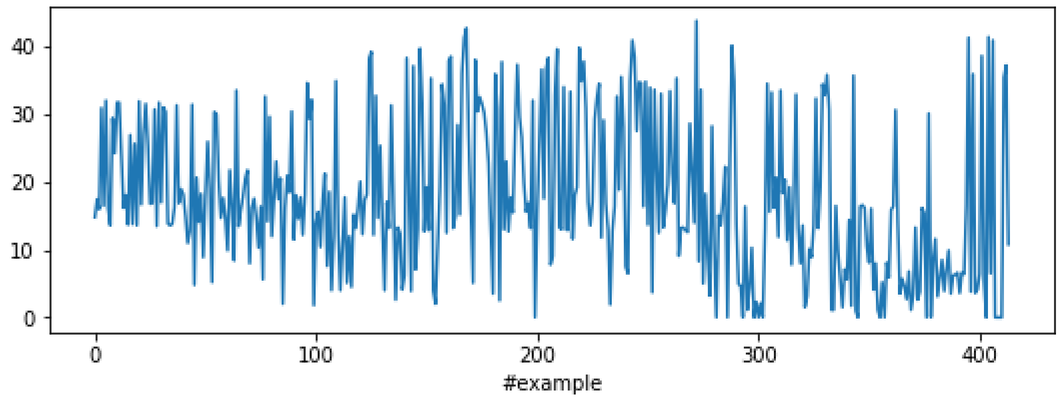
With this trick, we can assess our model performance on unseen data

Training and Test Set

When we partition the data, we need to be careful with sorted datasets

Let's plot the values of all attributes for each example in the dataset

In [7]:
```
plt.figure(figsize=(9, 3))
data['house age'].plot(xlabel='#example');
```



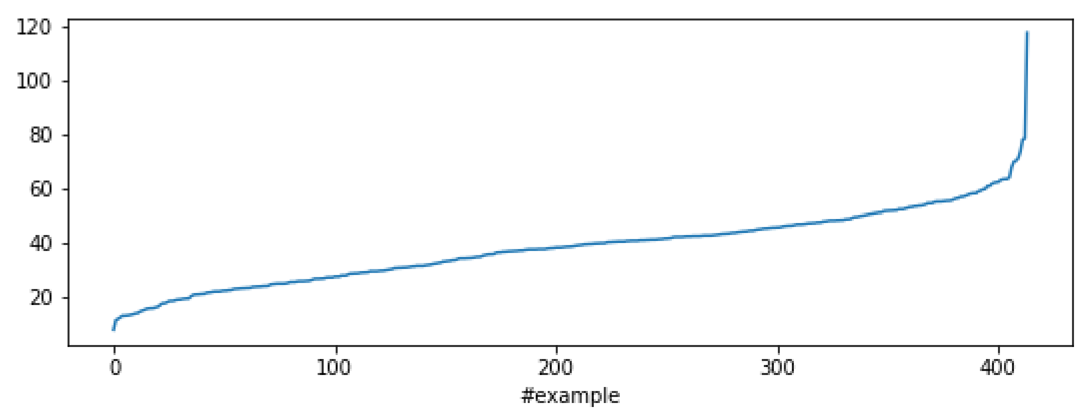
- Examples do not appear to be sorted by "house age"

Training and Test Set

When we partition the data, we need to be careful with sorted datasets

Let's plot the values of all attributes for each example in the dataset

In [8]:
```
plt.figure(figsize=(9, 3))
data['price per area'].plot(xlabel='#example');
```



- ...But they are sorted by "price per area"!

Training and Test Set

If we simply split our data in two groups

- We will train our model only on low prices
- ...And evaluate its performance only on higher prices

If we do hit, the model will generalize poorly

For most ML methods to work there is a basic requirement

The training data should be representative of the true population

- It means that the training data should be similar
- ...To the data to which we will actually apply our model

How do we achieve this with a single dataset?

Training and Test Set

The solution is to shuffle the data before partitioning

In scikit-learn, we can use a single function for both steps:

```
In [9]: from sklearn.model_selection import train_test_split

        X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.34, random_state=42)

        print(f'Size of the training set: {len(X_tr)}')
        print(f'Size of the test set: {len(X_ts)}')
```

```
Size of the training set: 273
Size of the test set: 141
```

The function `train_test_split`

- Randomly shuffles the data (optionally with a fixed seed `random_state`)
- Puts a fraction `test_size` of the data in the test set
- ...And the remaining data in the training set
- Both the input and the output data is processed in this fashion

Training and Test Set

Using separate test set is extremely important

...Because we want our model to work on new data

- We have no use for a model that learns the input data perfectly
- ...But that behaves poorly on unseen data
- In these cases, we say that the model does not generalize

By keeping a separate test set we can simulate this evaluation

For the best performance...

...Training and test data should have similar probability distributions

- Informally, they should be relatively similar
- In this case, we say that the training set is representative

Fitting the Model

We can now train a linear model using the scikit-learn library

```
In [10]: from sklearn.linear_model import LinearRegression

         m = LinearRegression()
         m.fit(X_tr, y_tr)

Out[10]: LinearRegression()
```

We obtain the estimated output via the `predict` method:

```
In [11]: y_pred_tr = m.predict(X_tr)
         y_pred_ts = m.predict(X_ts)
```

- The predictions (unlike the targets) are not guaranteed to be integers
- ...But that is still fine, since it's easy to interpret them

Evaluation

Finally, we need to evaluate the prediction quality

A common approach is using metrics. Here are a few examples:

- The Mean Absolute Error is given by:

$$MAE = \frac{1}{m}\sum_{i=1}^{m} |f(x_i) - y_i|$$

- The Root Mean Squared Error is given by:

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(f(x_i) - y_i)^2}$$

Both the RMSE and MAE a simple error measures

- They are expresses in the same unit as the original variable

Evaluation

- The coefficient of determination (R^2 coefficient) is given by:

$$R^2 = 1 - \frac{\sum_{i=1}^{m} (f(x_i) - y_i)^2}{\sum_{i=1}^{m} (y_i - \tilde{y})^2}$$

where $\tilde{y}$ is the average of the y values

The coefficient of determination is a useful, but more complex metric:

- Its maximum is 1: an $R^2 = 1$ implies perfect predictions
- Having a known maximum make the metric very readable
- It can be arbitrarily low (including negative)
- It can be subject to a lot of noise if the targets y have low variance

Using the MSE directly for evaluation is usually a bad idea

...Since it is a square, and therefore not easy to parse for a human

Evaluation

Let's see the values for our example

In [12]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

print(f'MAE on the training data: {mean_absolute_error(y_tr, y_pred_tr):.3}')
print(f'MAE on the test data: {mean_absolute_error(y_ts, y_pred_ts):.3}')
print(f'RMSE on the training data: {np.sqrt(mean_squared_error(y_tr, y_pred_tr)):.3}')
print(f'RMSE on the test data: {np.sqrt(mean_squared_error(y_ts, y_pred_ts)):.3}')
print(f'R2 on the training data: {r2_score(y_tr, y_pred_tr):.3}')
print(f'R2 on the test data: {r2_score(y_ts, y_pred_ts):.3}')
```

```
MAE on the training data: 0.143
MAE on the test data: 0.177
RMSE on the training data: 0.207
RMSE on the test data: 0.253
R2 on the training data: 0.691
R2 on the test data: 0.645
```

- In general, we have better predictions on the training set than on the test set
- This is symptomatic of some overfitting
- I.e. we are learning patterns that don't translate to unseen data

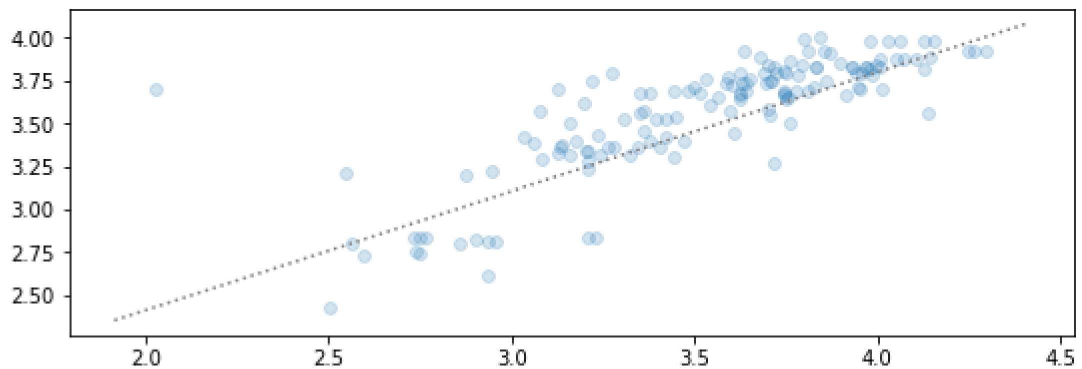
Later on, we will see some techniques to deal with this situation

Evaluation

As an (important!) alternative to metrics, we can use scatter plots:

- On the x-axis, we show the target values
- On the y-axis, we show the predictions

In [13]:
```
from matplotlib import pyplot as plt
plt.figure(figsize=(9,3))
plt.scatter(y_ts, y_pred_ts, alpha=0.2)
plt.plot(plt.xlim(), plt.ylim(), linestyle=':', color='0.5');
```



This gives us a better idea of which kind of mistakes the model is making

Conclusions and Take-Home Messages

- Basic formulation of supervised learning
  - I.e. learning a model from available examples
  - ...When the examples contain values for both the input and the output
- Basic linear regression model
  - One the simplest approaches for supervised learning
  - I.e. the output is a linear combination of the input values
  - Regression = we estimate a numeric quantity
- Train/test set split
  - Needed to evaluate our model on unseen data (generalization)
- Evaluation of regression models
  - Make sure to compare the performance on both training and test data
  - Metrics (e.g. RMSE, MAE) provide a compact evaluation
  - Scatter plot for a more fine-grained evaluation