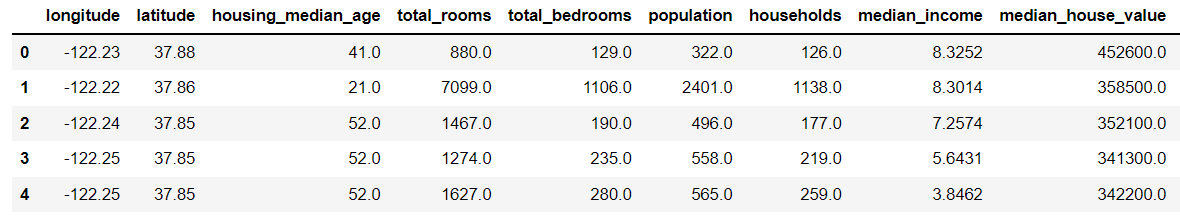
**Regression**

* Mostly concentrate on linear regression (line-based regressions)
* Supervised or unsupervised?
* Example: picture
  + Datapoints, prediction line
  + What is the difference compared to classification?

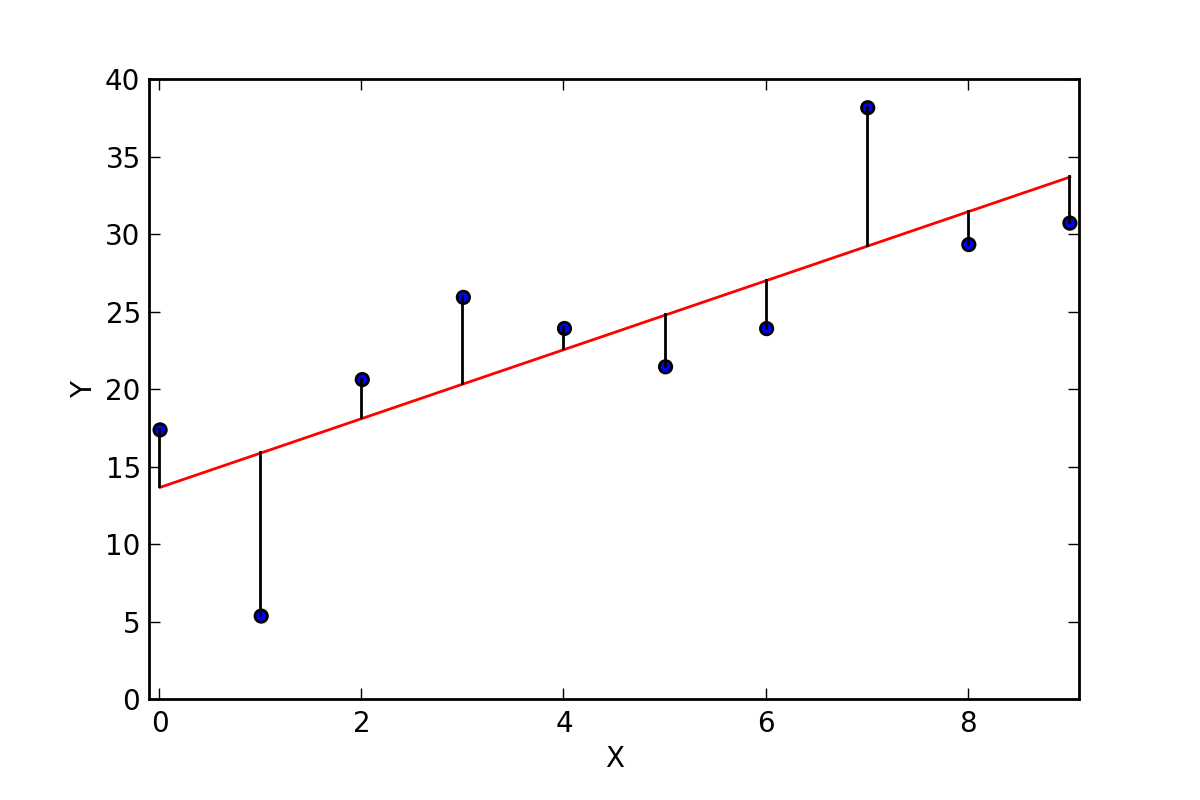
**Formally**

* Dependent variable (y)
* Independent variables (xi)
* Error term
* Looking for βi coefficients 🡪 predicted value: "y hat"  
  (looking for a linear line which fits the data)
* Β0: constant or intercept
* Example: prediction of income from housing data
*   
  (Jupyter – 4)

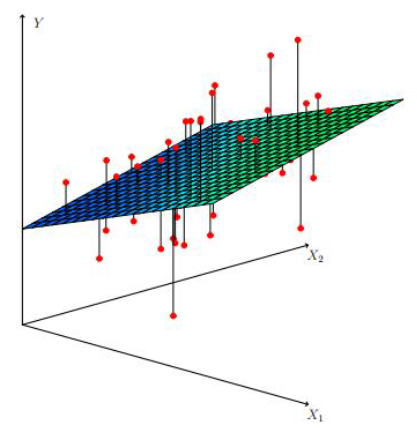
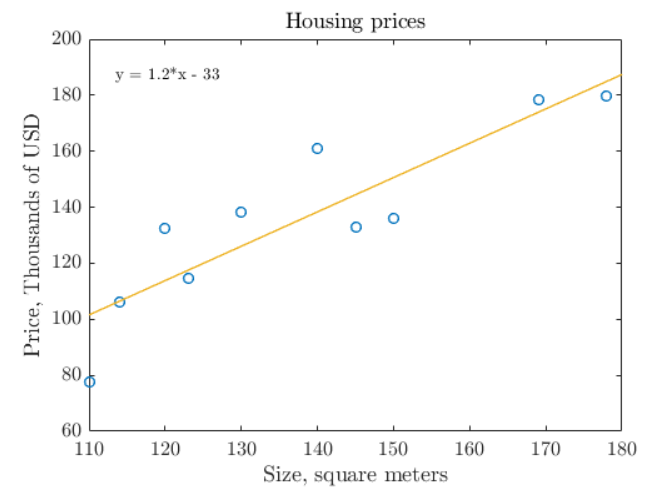
**Nomenclature**

* 1 target and 1 input variable
* 1 target and more input variables
* Also possible to have more target variables (not just the income but the age of the house)

**How do we learn the model?**

* (the coefficients)
* find it out by minimizing a **loss**
* What was loss?
* 
* A loss e.g. is the distance like above (absolute error) -> taking the mean: MAE
* But the more usual loss fn:
  + MSE: mean squared error
  + SE: squared error

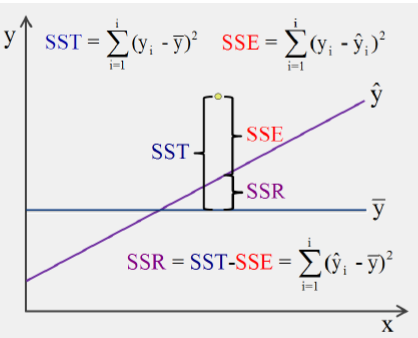
(picture below with squares -> size of the square = square of the distance)

* Formulas and more loss functions: <https://www.section.io/engineering-education/understanding-loss-functions-in-machine-learning/>
* Looking for the line for which the loss value is the smallest
* Back to MSE: why is it good to square the distance?   
  bigger punishment for the farther datapoints (e.g. dist 1 -> 1^2, dist 10 -> 10^2)
* In higher dimension:  
  
* 

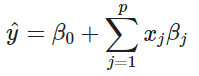
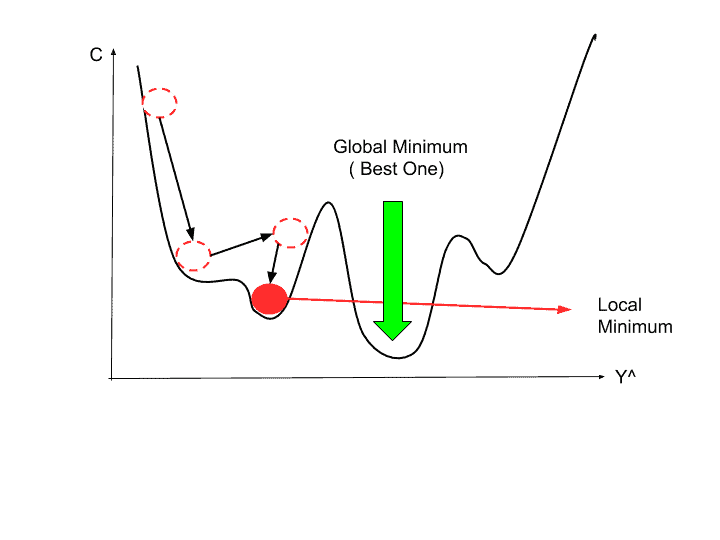
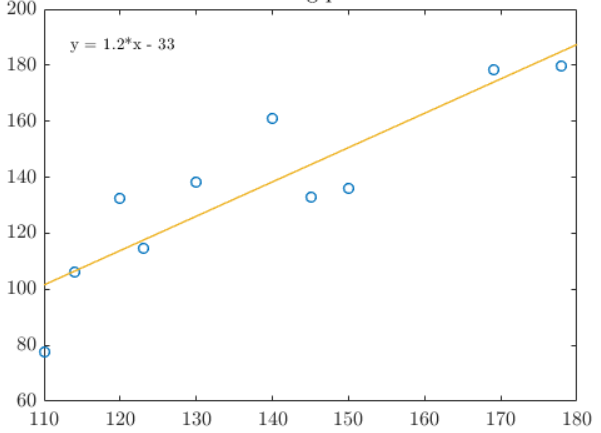
**Sums of squares**

* Residual: (distance of real datapoint and prediction)
* We sum the squares of these residuals
* Total=Explained+Residual:
  + explained variance: the mean ()
  + residual: as above. We want it to be as small as possible.

**Alternative measures - Regression sum of squares**

* 

**Analytic vs. non-analytic way of finding the best model parameters** (go back a bit)

* So according to the loss fn. I know “how bad am I”.
* But we still need to find the best parameter which give us the best result according to the chosen loss fn.
* But how to find the fn. (params) which has the min loss?
* 2 ways of finding the best model parameters according to the loss fn you chose:
  + Analytic way:  
    calculation based on a formula, e.g. 
    - deterministic
    - finds the global optimum
  + Non-analytic: most typically gradient descent (GD)
    - Approximation, not deterministic
    - Might stuck in local optimum
    - Method:
    -   
      (iterative process with learning rate – step size – parameter, no improvement with that step size from the loc.min.)
    - In case of the lin reg:
    - Basically you step through the opt line  
      
* Why would we choose GD, when analytic method seems much better?
  + GD is faster (computationally less expens)
  + works well even with a large number of features
* Takeaway: 2 approaches in ML:
  + Analytic: deterministic: brute-force calculation
  + Non-analytic: based on approximation, give less good result, but can be calculated in more cases

**How good is it? - "Robust regression"**

* y-outlier:
  + 1.pic: good. Check squared error.
  + 2.pic: outlier along the y axis
    - e.g. x is age of house, y is price of it
    - but it’s a special house because a celebrity lived there
    - but the outlier doesn’t help us to find the usual tendency (connection between age and price)
    - but our squared error makes it worse: makes the line go much closer to the outlier.
    - So in such cases when we have lots of outliers, squared error is not our friend
    - 🡪 solution:
      * Within a treshold, count errors squared
      * outside the treshold, count only the distance (without squaring)
* x-outlier:  
  basically the same  
  (can be interpreted as a y-outlier)

**Demonstration**

* importing
* generating some data
* 1. figure:
  + as you can see, a sinusoid function was used for the data generation
  + fn: it starts with large amptitude and gets smaller and smaller
* 2.figure:
  + Let’s leave out some of the datapoints
  + Concentrate on a subset of them
  + Still a descending line would be fitted
* 3.figure:
  + Let’s add an outlier
  + All the other points are the same as on pic2
* Apply linear regression to the 2.dataset:  
  Check the implementation
  + Model training and predicing as usual
  + Plot the actal datapoints and the prediction
  + Result: linreg performs well
* Apply for the 3.dataset with the outlier:
  + Result (plot): linreg doesn’t perform well
  + The model uses least square regression
  + The outlier is far, so the penalty is even bigger (squared error) -> the line changes this much because of only one point
* 🡪 what can we do to get back our good line?

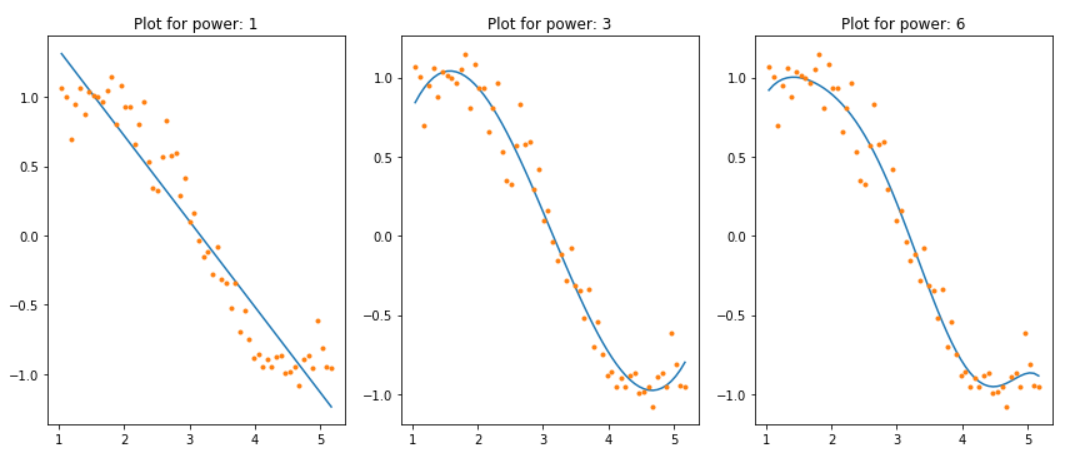
**Robust regression** (picture)

* + Use absolute error instead of squared.  
    But the abs fn has a sharp edge, which we don’t like (not differentiable).
  + And also, among the non-outlier datapoints, we prefer squared error.
  + 🡪 solution: Huber regrression
  + Implementation: HuberRegressor
  + Result: original prediction line even with the outlier in the data
  + 🡪 loss fn is important!

**RANSAC (Random Sample Consensus) regression**

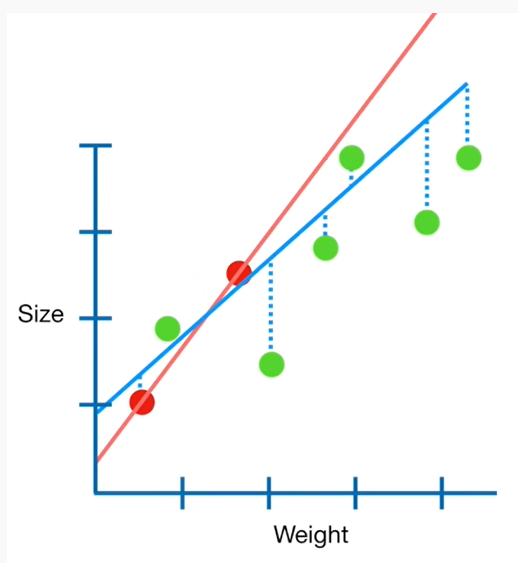
* Another approach to outliers
* Enough to remember the idea of consensus set
* Steps:
  1. Select a random number of examples to be inliers and train the model.
  2. Test all other data points against the trained model (does this point fit the pred line?)
  3. Out of all the data points tested in step 2, select the points as inliers which fall within a user-given tolerance. In scikit-learn, median absolute deviation (MAD) is used for selecting the new points as inliers.
  4. Retrain the model with all inliers data
  5. Estimate the error of the retrained model versus the inliers.
  6. Follow step 1 to step 5
  7. Terminate the algorithm execution if the model performance meets a certain user-defined threshold or if a fixed number of iterations were reached
* Again:
  + start choosing datapoints randomly
  + create a model
  + fit it on the chosen datapoints
  + try to find: how well the model fitted on them
  + Points that fit, get added to *consensus* set.
  + The model is good if the consensus set is big enough.
* Example (implementation) – check only the plot (to see the dataset):
  + We feel that the green points should be the real points of our dataset and the yellows are outliers.
  + (Actually in such a case we should suspect that we have a more complex model – below and above ~2, we have two different kind of relationship.)
  + But now let’s just say that the yellows are really outliers.
  + 🡪
    - Squared error optimized alg gives the *dark blue line* (understandable because the yellows are also considered)
    - *Light blue line:* RANSAC regressor: tries to find the consensus set (which is the majority vote)
* Takeaway: by randomly sampling (again and again), we can avoid the effect of outliers.  
  (Idea of subsumpling)
* Implementation (code):
  + This idea can be applied with any algorithm -> you can specify the alg (e.g. dec.tree)
  + Default is lin.reg.
  + inlier\_mask\_: this is just a Boolean mask (T/F), which shows which datapoints were predicted inliers by the model
  + coef, lr.coef\_,ransac.estimator\_.coef\_:
    - check lin.reg. documentation
    - intercept and coef params
    - importance of scaling:, e.g.:
      * x1 is age of house, e.g. 70 years
      * x2 is size of house, e.g. 30 m2
      * which one is more important in predicting the price of the house?
      * Cannot compare them
      * Scaling -> can compare, e.g. -1.2 vs 5.4 (same range -> can see their relative importance)

**What if we go non-linear? (Polynomial regression)**

* What can we do when our data is nonlin?  
  We shouldn’t just draw straight lines..
* We would like use higher degree polys (until now degree was 1)
* But how will we now which degrees wee need (which polynom to use)
* E.g. plot: deg=3.
* Code:
  + Choose degree
  + use pipeline: steps =
    - adding poly features, e.g. x,y,xy,x2
    - doing lin.reg.
    - we could do scaling as well, but if we don’t want to check the relative importance of our variables, then not needed (linreg works that way that it doesn’t need scaling)
  + fit the pipeline (enough to fit on x and y, because it will add the other poly features)
  + predict -> prediction: yellow polynomial
  + coefs printed
  + check in Google: y=3.03x-1.28x^2+0.13x^3
  + How is this fn calculated?  
    The alg tunes the parameters trying to find this optimal poly by using the same MAE or MSE as the lin.reg.
*   
  How to choose the right power of the polynomial?
  + The higher the degree -> the more exotic shapes can be found
  + But the higher -> the more it overfits  
    (check degrees above 8: very specific to the data)
  + 🡪 try to find the min degree which is already good
  + Because the model will use all the freedom you give. So it’s your responsibility to limit the degree properly

Extra

**Ridge regression**



* Red: training data
* Green: validation data
* Red line:
  + overfit to the training data  
    (very good on the red, but could be much better on the green)
  + has high variance
* Idea:
  + Try to find a new line that doesn’t fit the training data that well
  + Add a small amount of bias
  + In return for getting worse result on the training data, we get a smaller variance on the validation set.
* Solution:
  + Originally we minimise the sum of squared residulals (SSR)
  + Now: min(SSR+α\*slope2)
  + How to choose α?  
    Try several values and do cross-validation.

**Lasso regression**

* The idea is the same
* Just different formula