Week3: Assessing fit (Polynomial Regression)

Author: Pascal, Feb 2021

In this notebook you will compare different regression models in order to assess which model fits best. We will be using polynomial regression as a mean to examine this topic. In particular we will:

- Write a function to take a Vector and a degree and return an DataFrame where each column is the Vector to a polynomial value up to the total degree e.g. degree = 3 then column 1 is the Vector column 2 is the Vector squared and column 3 is the Vector cubed
- *Use Plots to visualize polynomial regressions*
- *Use Plots to visualize the same polynomial degree on different subsets of the data*
- Use a validation set to select a polynomial degree
- Assess the final fit using test data

We will continue to use the House data from previous notebooks.

```
md"""
# Week3: Assessing fit (Polynomial Regression)

*Author: Pascal,  Feb 2021*

In this notebook you will compare different regression models in order to assess which model fits best. We will be using polynomial regression as a mean to examine this topic. In particular we will:
  - Write a function to take a Vector and a degree and return an DataFrame where each column is the Vector to a polynomial value up to the total degree e.g. degree 3 then column 1 is the Vector column 2 is the Vector squared and column 3 is the Vector cubed
  - *Use Plots to visualize polynomial regressions*
  - *Use Plots to visualize the same polynomial degree on different subsets of the data*
  - Use a validation set to select a polynomial degree
  - Assess the final fit using test data

We will continue to use the House data from previous notebooks.
"""
```

```
begin
    using Pkg
    Pkg.activate("MLJ_env", shared=true)
```

```
begin
    using MLJ
    using CSV
    using DataFrames
```

```
using PlutoUI
using Test
using Printf
using Random
using Plots  # using PyPlot
```

Polynomial dataframe function

polynomial_df (generic function with 1 method)

```
function polynomial_df(feature; degree=3)
    @assert degree ≥ 1 "Expect degree to be ≥ 1"

    hsh = Dict{Symbol, Vector{Float64}}(:power_1 => feature)
    for deg ∈ 2:degree
        hsh[Symbol("power_$(deg)")] = feature .^ deg
    end

    return DataFrame(hsh)
```

v = Float64[10.0, 4.0, 9.0]

df =

	power_1	power_2	power_3	power_4
1	10.0	100.0	1000.0	10000.0
2	4.0	16.0	64.0	256.0
3	9.0	81.0	729.0	6561.0

Visualizing polynomial regression

```
sales = CSV.File("../../ML_UW_Spec/C02/data/kc_house_test_data.csv";
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_l
1	114101516	"20140528T000000"	310000.0	3	1.0	1430	19901
2	9297300055	"20150124T000000"	650000.0	4	3.0	2950	5000
3	1202000200	"20141103T000000"	233000.0	3	2.0	1710	4697
4	8562750320	"20141110T000000"	580500.0	3	2.5	2320	3980
5	7589200193	"20141110T000000"	535000.0	3	1.0	1090	3000

train_test_split (generic function with 1 method)

```
function train_test_split(df; split=0.8, seed=42)
    Random.seed!(seed)
    (nr, nc) = size(df)
    nrp = round(Int, nr * split)
    row_ixes = shuffle(1:nr)
    df_train = view(df[row_ixes, :], 1:nrp, 1:nc)
    df_test = view(df[row_ixes, :], nrp+1:nr, 1:nc)
    (df_train, df_test)
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_l
1	2856101479	"20140701T000000"	276000.0	1	0.75	370	1801
2	9266700190	"20150511T000000"	245000.0	1	1.0	390	2000
3	6303400395	"20150130T000000"	325000.0	1	0.75	410	8636
4	7549801385	"20140612T000000"	280000.0	1	0.75	420	6720
5	745000005	"20140825T000000"	145000.0	1	0.75	480	9750

```
begin
    sort!(sales, [:sqft_living, :price], rev=[false, false]);
    first(sales, 5)
```

First degree polynomial

Let's start with a degree 1 polynomial, using `:sqft_living` to predict `:price` nad plot what it looks like.

```
md"""
##### First degree polynomial

Let's start with a degree 1 polynomial, using `:sqft_living` to predict `:price` na
plot what it looks like.
"""
```

	power_1	price
1	370.0	276000.0
2	390.0	245000.0
3	410.0	325000.0

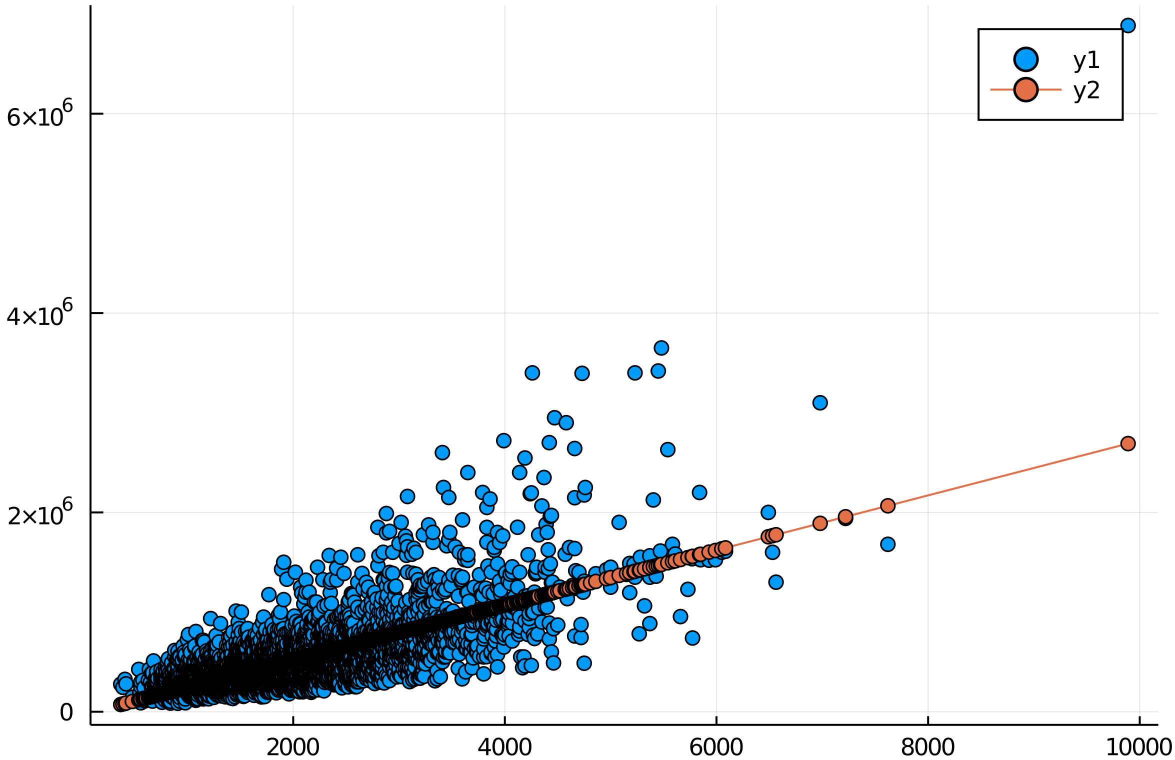
```julia
begin
    poly_df_1 = polynomial_df(sales.sqft_living; degree=1)
    poly_df_1[!, :price] = sales.price

    first(poly_df_1, 3)
```

```
MLJLinearModels.LinearRegressor
```

```
power_1   :  275.0
Intercept: -28600.0
```

```julia
begin
    mdl1 = MLJLinearModels.LinearRegressor()

    X_1 = select(poly_df_1, :power_1)
    y_1 = poly_df_1.price

    mach1 = machine(mdl1, X_1, y_1)
    fit!(mach1)
    fp1 = fitted_params(mach1)

    with_terminal() do
        for (name, c) in fp1.coefs
            println("$(rpad(name, 10)):  $(round(c, sigdigits=3))")
        end

        println("Intercept: $(round(fp1.intercept, sigdigits=3))")
    end
```

```
(DataFrame,  Array{Float64,1})
```




```julia
begin
    scatter(X_1.power_1, y_1, marker=".")
    plot!(X_1.power_1, predict(mach1, X_1), marker="-")
```

We can see, not surprisingly, that the predicted values all fall on a line, specifically the one with slope 275 and intercept -28600.

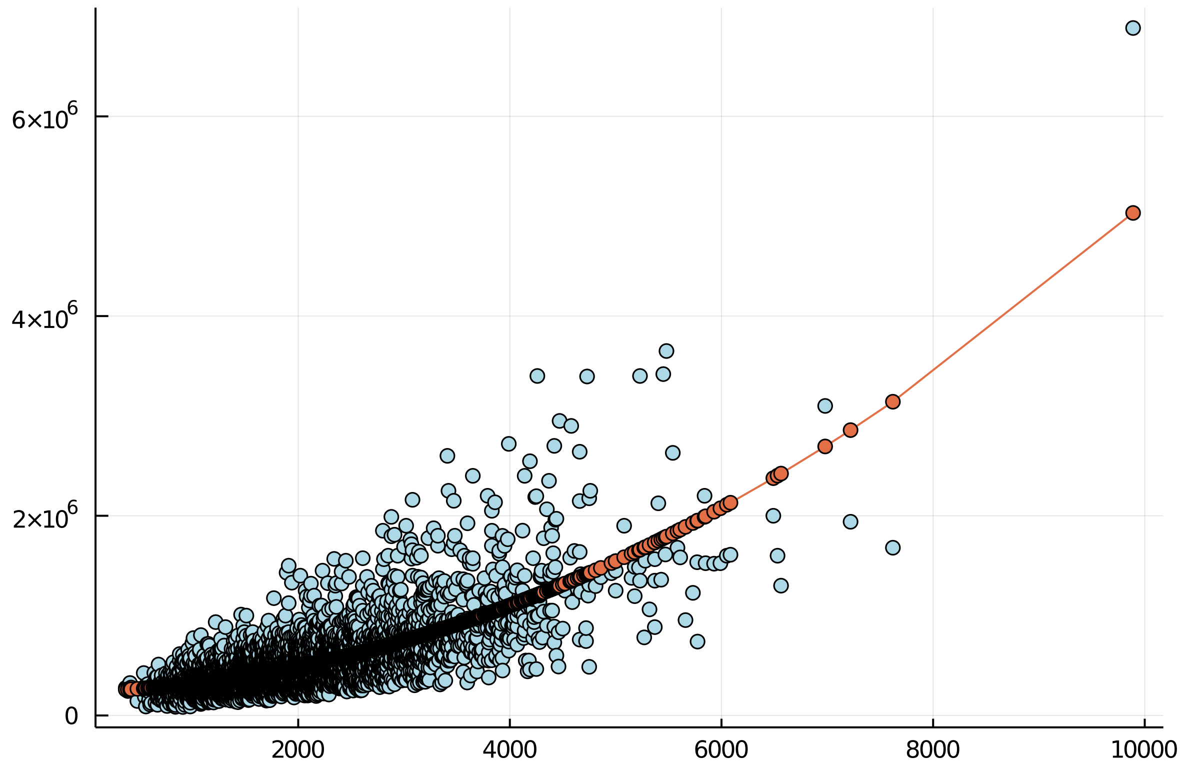
Second degree polynomial

	power_1	power_2	price
1	370.0	136900.0	276000.0
2	390.0	152100.0	245000.0
3	410.0	168100.0	325000.0

```
begin
    poly_df_2 = polynomial_df(sales.sqft_living; degree=2)
    feature_df_2 = names(poly_df_2)
    poly_df_2[!, :price] = sales.price

    first(poly_df_2, 3)
```

```
power_1   :  32.4
power_2   :  0.0457
Intercept: 240000.0
```

```
begin
    mdl2 = MLJLinearModels.LinearRegressor()

    X_2 = select(poly_df_2, feature_df_2)
    y_2 = poly_df_2.price

    mach2 = machine(mdl2, X_2, y_2)
    fit!(mach2)
    fp2 = fitted_params(mach2)

    with_terminal() do
        for (name, c) in fp2.coefs
            println("$(rpad(name, 10)):  $(round(c, sigdigits=3))")
        end

        println("Intercept: $(round(fp2.intercept, sigdigits=3))")
    end
```

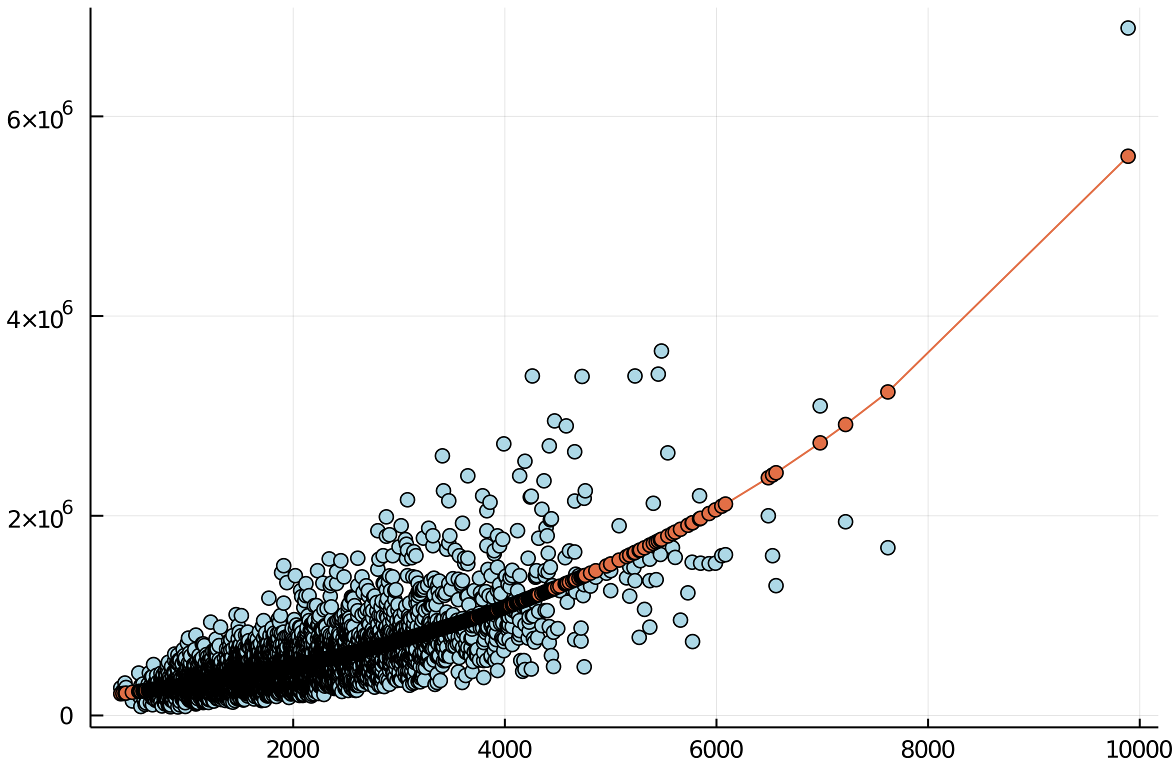



```
## Visualization
begin
    scatter(X_2.power_1, y_2, legend=false, color=[:lightblue], marker=".")
    plot!(X_2.power_1, predict(mach2, X_2), marker="-")
```

Third degree polynomial

```
power_1   :  119.0
power_2   :  0.0164
power_3   :  2.74e-6
Intercept: 171000.0
```

```
begin
    poly_df_3 = polynomial_df(sales.sqft_living; degree=3)
    feature_df_3 = names(poly_df_3)
    poly_df_3[!, :price] = sales.price

    mdl3 = MLJLinearModels.LinearRegressor()

    X_3 = select(poly_df_3, feature_df_3)
    y_3 = poly_df_3.price

    mach3 = machine(mdl3, X_3, y_3)
    fit!(mach3)
    fp3 = fitted_params(mach3)

    with_terminal() do
        for (name, c) in fp3.coefs
            println("$(rpad(name, 10)):  $(round(c, sigdigits=3))")
        end

        println("Intercept: $(round(fp3.intercept, sigdigits=3))")
    end
```



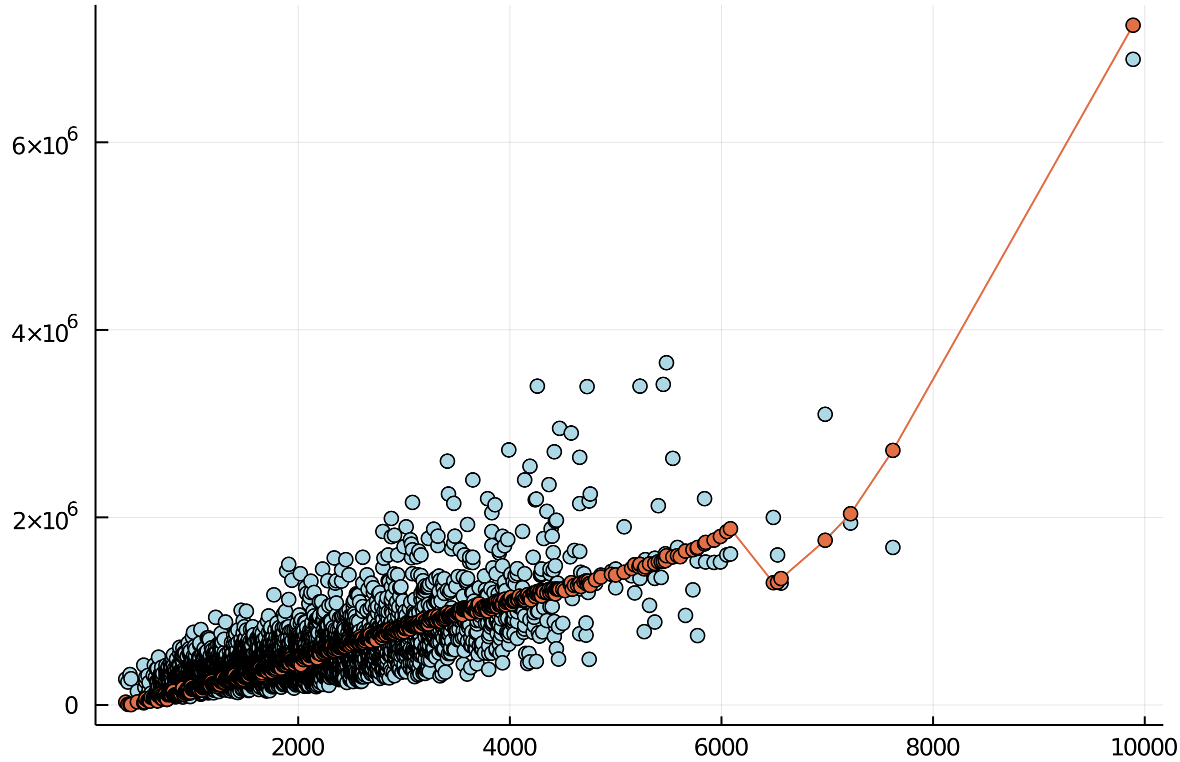

```julia
## Visualization
begin
    scatter(X_3.power_1, y_3, legend=false, color=[:lightblue], marker=".")
    plot!(X_3.power_1, predict(mach3, X_3), marker="-")
```

15th degree polynomial

```
power_1   :  0.000165
power_10  :  9.16e-16
power_11  :  -3.37e-16
power_12  :  1.27e-15
power_13  :  2.67e-15
power_14  :  1.16e-15
power_15  :  1.19e-15
power_2   :  0.205
power_3   :  -4.93e-5
power_4   :  3.63e-9
power_5   :  4.6e-14
power_6   :  9.53e-16
power_7   :  -4.53e-16
power_8   :  8.71e-16
power_9   :  -2.43e-16
Intercept: 1.03e-7
```

```julia
begin
    poly_df_15 = polynomial_df(sales.sqft_living; degree=15)
    feature_df_15 = names(poly_df_15)
    poly_df_15[!, :price] = sales.price

    mdl15 = MLJLinearModels.LinearRegressor()

    X_15 = select(poly_df_15, feature_df_15)
    y_15 = poly_df_15.price

    mach15 = machine(mdl15, X_15, y_15)
    fit!(mach15)
    fp15 = fitted_params(mach15)

```

```
with_terminal() do
    for (name, c) in fp15.coefs
        println("$(rpad(name, 10)):  $(round(c, sigdigits=3))")
    end

    println("Intercept: $(round(fp15.intercept, sigdigits=3))")
end
```




```
## Visualization
begin
    scatter(X_15.power_1, y_15, legend=false, color=[:lightblue], marker=".")
    plot!(X_15.power_1, predict(mach15, X_15), marker="-")
```

What do you think of the 15th degree polynomial?
 Do you think this is appropriate?

As expected, it looks like the model learn too much of the idiosyncrasies of the training data

If we were to change the data do you think you'd get pretty much the same curve? Let's take a look.

Changing the data and re-learning

We are going to split the sales data into four subsets of roughly equal size. Then you will estimate a 15th degree polynomial model on all four subsets of the data. Print the coefficients and plot the resulting fit (as we did above). The quiz will ask you some questions about these results.

To split the sales data into four subsets, we perform the following steps:

- First split sales into 2 subsets, 50% of the original set

- Next split the resulting subsets into 2 more subsets each.

We set `seed=42` in these steps so that different users get consistent results. we should end up with 4 subsets (`set1` , `set2` , `set3` , `set4`) of approximately equal size.

```
((1057, 21), (1057, 21), (1058, 21), (1057, 21))
```

```
begin
    (ssales_a, ssales_b) = train_test_split(sales; split=0.5, seed=42)

    (set1, set2) = train_test_split(ssales_a; split=0.5, seed=42)
    (set3, set4) = train_test_split(ssales_b; split=0.5, seed=42)

    (size(set1), size(set2), size(set3), size(set4))
```

Fit a 15th degree polynomial on `set1` , `set2` , `set3` , and `set4` using `sqft_living` to predict prices.

Print the coefficients and make a plot of the resulting model.

```
print_coeff (generic function with 1 method)
```

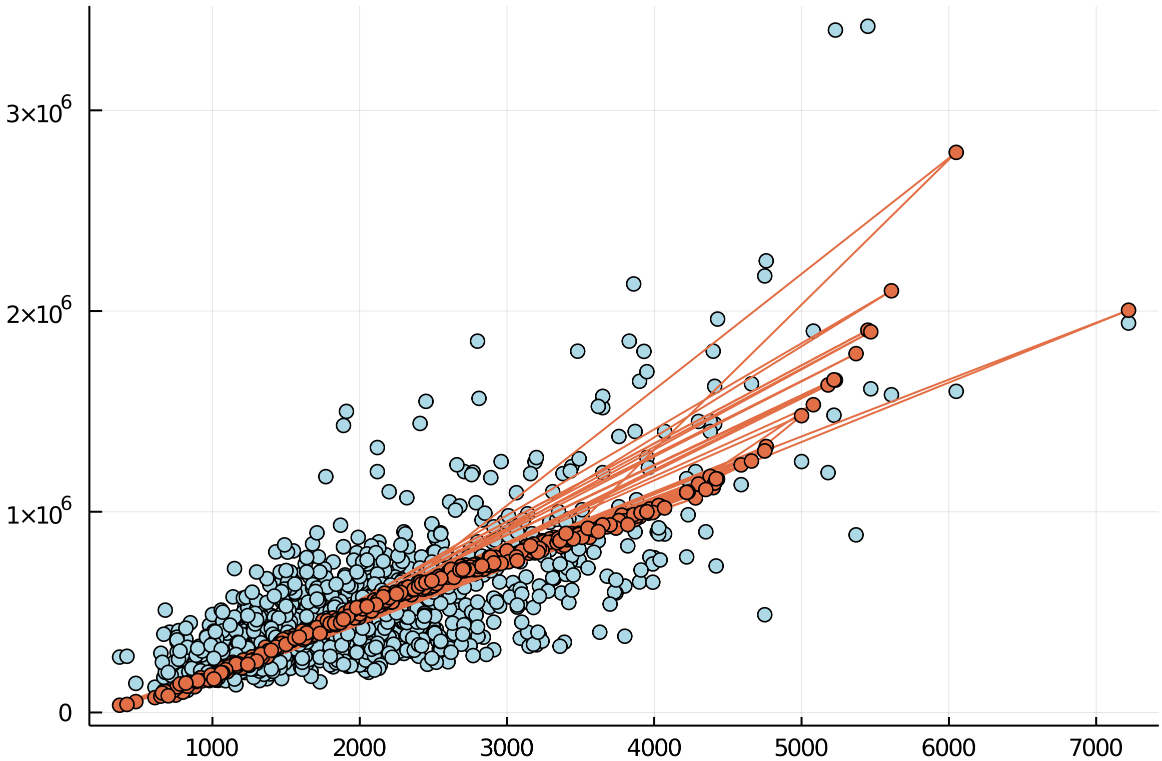
```
begin

function make_poly(tset, degree)
  poly_df = polynomial_df(tset.sqft_living; degree)
  features = names(poly_df)
  poly_df[!, :price] = tset.price
  (features, poly_df)
end

function fit_poly(tset; degree=15)
  (features, poly_df) = make_poly(tset, degree)
  mdl = MLJLinearModels.LinearRegressor()
  X_ = select(poly_df, features)
  y_ = poly_df.price

  mach = machine(mdl, X_, y_)
  fit!(mach)

  (mach, X_, y_)
end

function print_coeff(mach)
  fp = fitted_params(mach)

  with_terminal() do
    for (name, c) in fp.coefs
      println("$(rpad(name, 10)):  $(round(c, sigdigits=3))")
    end

    println("Intercept: $(round(fp.intercept, sigdigits=3))")
  end
end

```

Model for set1

```
power_1    :  0.000255
```

```
power_10   :  -1.95e-15
power_11   :  -4.96e-16
power_12   :  3.09e-17
power_13   :  4.93e-16
power_14   :  -5.77e-16
power_15   :  -1.66e-18
power_2    :  0.256
power_3    :  -7.84e-5
power_4    :  6.59e-9
power_5    :  2.44e-13
power_6    :  -1.44e-15
power_7    :  -8.83e-16
power_8    :  7.44e-16
power_9    :  1.64e-16
Intercept: 1.9e-7
```

```
## set 1
begin
    (mach_set1, Xset1, yset1) = fit_poly(set1)
    print_coeff(mach_set1)
```



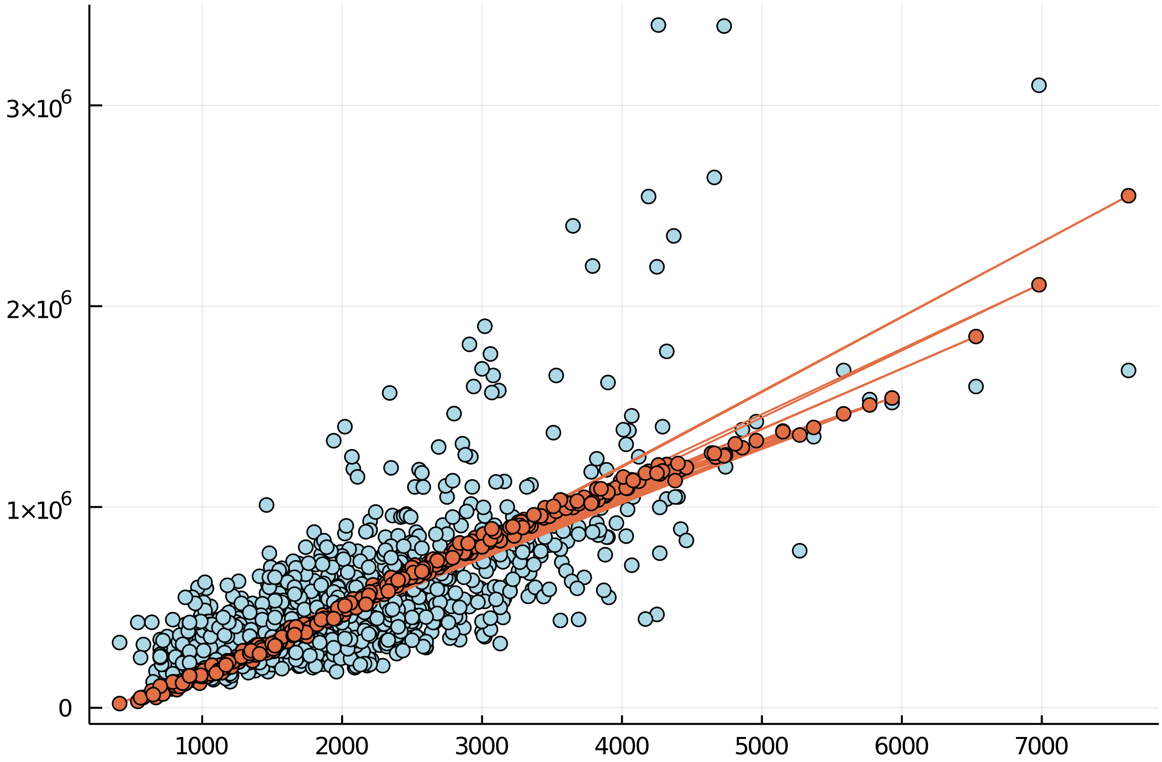

```
## Visualization
begin
    scatter(Xset1.power_1, yset1, legend=false, color=[:lightblue], marker=".")
    plot!(Xset1.power_1, predict(mach_set1, Xset1), marker="-")
```

Model for set2

```
power_1    :  0.000182
power_10   :  1.66e-15
power_11   :  -2.32e-16
power_12   :  1.31e-15
power_13   :  2.3e-16
power_14   :  1.84e-15
power_15   :  -2.23e-17
power_2    :  0.203
power_3    :  -4.69e-5
power_4    :  3.43e-9
power_5    :  -8.76e-15
```

```
power_6   :  1.78e-15
power_7   :  1.1e-15
power_8   :  -9.01e-16
power_9   :  -3.81e-16
Intercept: 1.23e-7
```

```
## set 2
begin
    (mach_set2, Xset2, yset2) = fit_poly(set2)
    print_coeff(mach_set2)
```



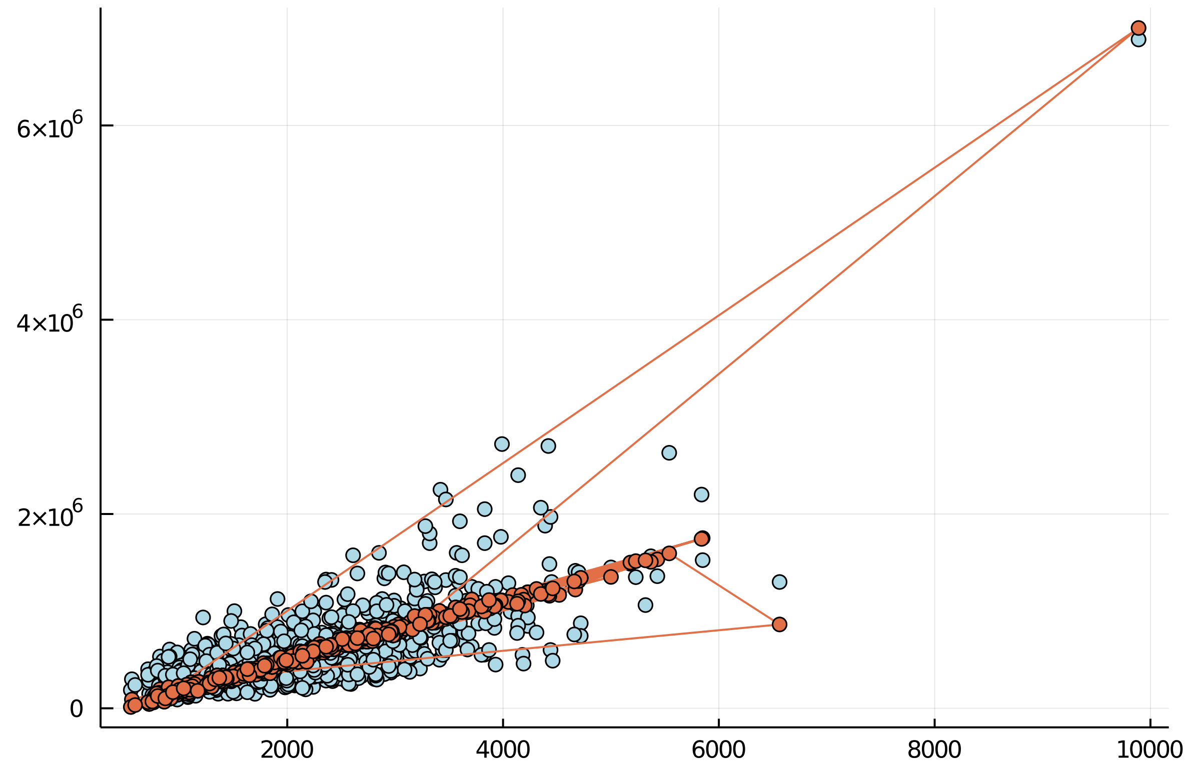

```
## Visualization
begin
    scatter(Xset2.power_1, yset2, legend=false, color=[:lightblue], marker=".")
    plot!(Xset2.power_1, predict(mach_set2, Xset2), marker="-")
```

Model for set3

UndefVarError: mach_se3 not defined

1. top-level scope @ *Local: 4*

```
## set 3
begin
    (mach_set3, Xset3, yset3) = fit_poly(set3)
    print_coeff(mach_se3)
```

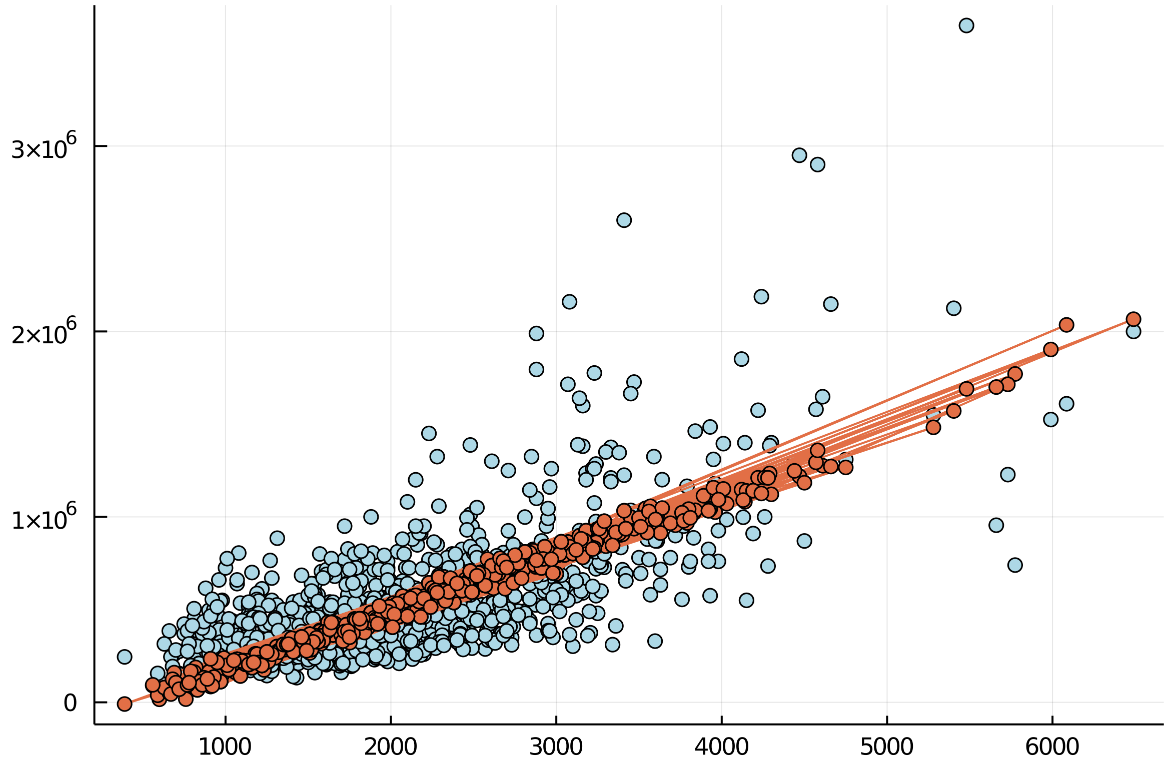



```
## Visualization
begin
    scatter(Xset3.power_1, yset3, legend=false, color=[:lightblue], marker=".")
    plot!(Xset3.power_1, predict(mach_set3, Xset3), marker="-")
```

Model for set4

```
power_1   :  0.00022
power_10  :  3.43e-15
power_11  :  -4.42e-16
power_12  :  1.2e-15
power_13  :  5.75e-15
power_14  :  1.02e-15
power_15  :  1.9e-15
power_2   :  0.218
power_3   :  -5.75e-5
power_4   :  4.91e-9
power_5   :  1.83e-14
power_6   :  2.15e-15
power_7   :  -5.61e-17
power_8   :  2.89e-15
power_9   :  3.15e-16
Intercept: 1.65e-7
```

```
## set 4
begin
    (mach_set4, Xset4, yset4) = fit_poly(set4)
    print_coeff(mach_set4)
```




```
## Visualization
begin
    scatter(Xset4.power_1, yset4, legend=false, color=[:lightblue], marker=".")
    plot!(Xset4.power_1, predict(mach_set4, Xset4), marker="-")
```

Some questions you will be asked on your quiz:

Quiz Question: Is the sign (positive or negative) for power_15 the same in all four models?

- Sign is positive for model 1, 2 and 3 and negative for model 4

Quiz Question: (True/False) the plotted fitted lines look the same in all four plots

- The fitted lines are very different for each model.

Selecting a Polynomial Degree

Whenever we have a "magic" parameter like the degree of the polynomial there is one well-known way to select these parameters: validation set. (We will explore another approach in week 4).

We split the sales dataset 3-way into training set, test set, and validation set as follows:

- Split our sales data into 2 sets: `training_and_validation` and `testing`. Use 90%/10% split.
- Further split our training data into two sets: `training` and `validation`. Use 50%/50% split.

We set `seed=42` to obtain consistent results for different users.

```
((2284, 21), (1522, 21), (423, 21))
```

```
begin
    (training_validation, testing) = train_test_split(sales; split=0.9, seed=42)
    (training, validation) = train_test_split(training_validation; split=0.6, seed=42)

    (size(training), size(validation), size(testing))
```

Next you should write a loop that does the following:

- For degree ∈ 1:15
  - Build a DataFrame of polynomial data of `train_data.sqft_living` at the current degree
  - Add `train_data.price` to the polynomial DataFrame
  - Learn a polynomial regression model to sqft vs price with that degree on *training* data
  - Compute the RSS on *validation* data for that degree and you will need to make a polynmial DataFrame using *validation* data.
- Report which degree had the lowest RSS on validation data

```
get_rss (generic function with 1 method)
```

```
function get_rss(mach, X, y)
    ŷ = predict(mach, X)      # First get the predictions
    diff = y .- ŷ             # Then compute the residuals/errors
    rss = sum(diff .* diff)   # Then square and add them up
    return rss
```

```
find_best_degree (generic function with 1 method)
```

```
function find_best_degree()
    max_degree = 15
    best_rss = nothing
    best_degree, best_mach = (nothing, nothing)

    for degree ∈ 1:max_degree
        (mach_set, _Xset, _yset) = fit_poly(training; degree)
        (_features_val, poly_df_val) = make_poly(validation, degree)
        rss = get_rss(mach_set, poly_df_val, poly_df_val.price)
        # @printf("degree: %2d / rss: %2.5e / best rss: %2.5e\n", degree, rss, best_rss)
        if isnothing(best_rss) || rss < best_rss
            best_rss = rss
            best_degree = degree
            best_mach = mach_set
        end
    end
    return (best_degree, best_rss, best_mach)
```

```
best model degree:  5 / lowest rss: 1.03636e+14
```

```
begin
```

```
    (best_degree, best_rss, best_mach) = find_best_degree()
    with_terminal() do
        @printf("best model degree: %2d / lowest rss: %2.5e\n", best_degree,
best_rss)
    end
```

Quiz Question: Which degree (1, 2, …, 15) had the lowest RSS on Validation data?

Now that you have chosen the degree of your polynomial using validation data, compute the RSS of this model on *testing* data. Report the RSS on your quiz.

```
best model degree:  5 / rss: on test set 2.50135e+13
```

```
with_terminal() do
    (_features_test, poly_df_test) = make_poly(testing, best_degree)
    rss_test = get_rss(best_mach, poly_df_test, poly_df_test.price)

    @printf("best model degree: %2d / rss: on test set %2.5e\n", best_degree,
rss_test)
```