```
([347, 1, 265, 286, 328, 25, 127, 307, 73,    more ,382], [358, 45, 225, 19, 210, 251,
```

```
begin
    using MLJ   ,StatsPlots
import RDatasets   : dataset
import DataFrames   : DataFrame, select
import  MLJLinearModels
auto = dataset("ISLR", "Auto")
y, X = unpack(auto, ==(:MPG), col->true)
train, test = partition(eachindex(y), 0.5, shuffle=true, rng=444);
end
```

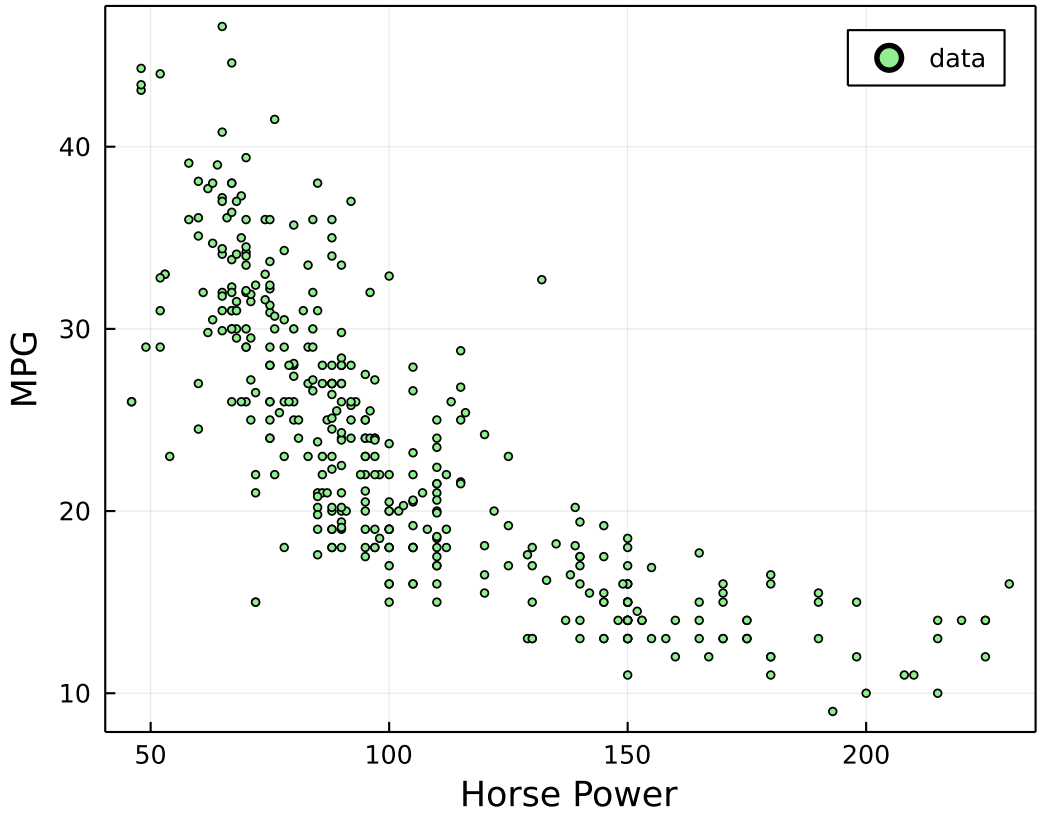
1. 多项式回归

```
md"""
## 1. 多项式回归
"""
```

```
LR = MLJLinearModels.LinearRegressor
```

```
LR = @load LinearRegressor pkg=MLJLinearModels
```

```
import MLJLinearModels ✓ ? erbosity=0`.
```



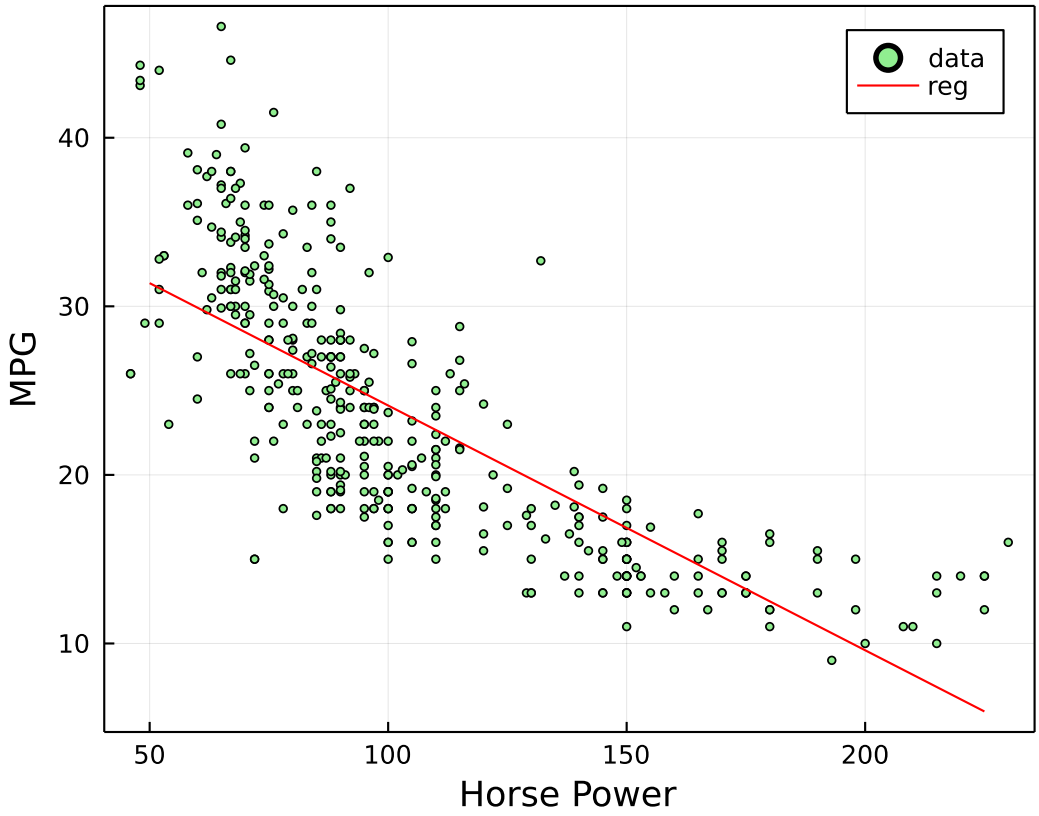

```
begin
    p1=scatter(X.Horsepower, y,ms=2, color=:lightgreen,label="data",
    xlabel="Horse Power",ylabel="MPG",size=(500,400),frame=:box)
end
```

23.493990895007986

```
begin
    lm = LR()
    mlm = machine(lm, select(X, :Horsepower), y)
    fit!(mlm, rows=train)
    rms(MLJ.predict(mlm, rows=test), y[test])^2

end
```

```
Solver: MLJLinearModels.Analytical _intercept = true, …), …).
  iterative: Bool false
  max_inner: Int64 200
```




```
begin
    xx = (Horsepower=range(50, 225, length=100) |> collect, )
    yy = MLJ.predict(mlm, xx)
    p1
    p2=plot!(xx.Horsepower, yy,label="reg",color=:red)

end
```

	hp1	hp2	hp3
1	130.0	16900.0	2.197e6
2	165.0	27225.0	4.49212e6
3	150.0	22500.0	3.375e6
4	150.0	22500.0	3.375e6
5	140.0	19600.0	2.744e6
6	198.0	39204.0	7.76239e6
7	220.0	48400.0	1.0648e7
8	215.0	46225.0	9.93838e6
9	225.0	50625.0	1.13906e7
10	190.0	36100.0	6.859e6
	more		
392	82.0	6724.0	551368.0

```
begin
    hp = X.Horsepower
    Xhp = DataFrame(hp1=hp, hp2=hp.^2, hp3=hp.^3);
end
```

```
LinMod = Pipeline(
    FeatureSelector(features=[:hp1]),
    LR()
);
```

```
Machine trained 1 time; caches data
  model: DeterministicPipeline(feature_selector = FeatureSelector(features = [:hp1,
  args:
    1:  Source @902 ⏎ `ScientificTypesBase.Table{AbstractVector{ScientificTypesBase.(
    2:  Source @923 ⏎ `AbstractVector{ScientificTypesBase.Continuous}`
```

```
begin
    lr1 = machine(LinMod, Xhp, y) # poly of degree 1 (line)
    fit!(lr1, rows=train)

    LinMod.feature_selector.features = [:hp1, :hp2] # poly of degree 2
    lr2 = machine(LinMod, Xhp, y)
    fit!(lr2, rows=train)

    LinMod.feature_selector.features = [:hp1, :hp2, :hp3] # poly of degree 3
    lr3 = machine(LinMod, Xhp, y)
    fit!(lr3, rows=train)
end
```

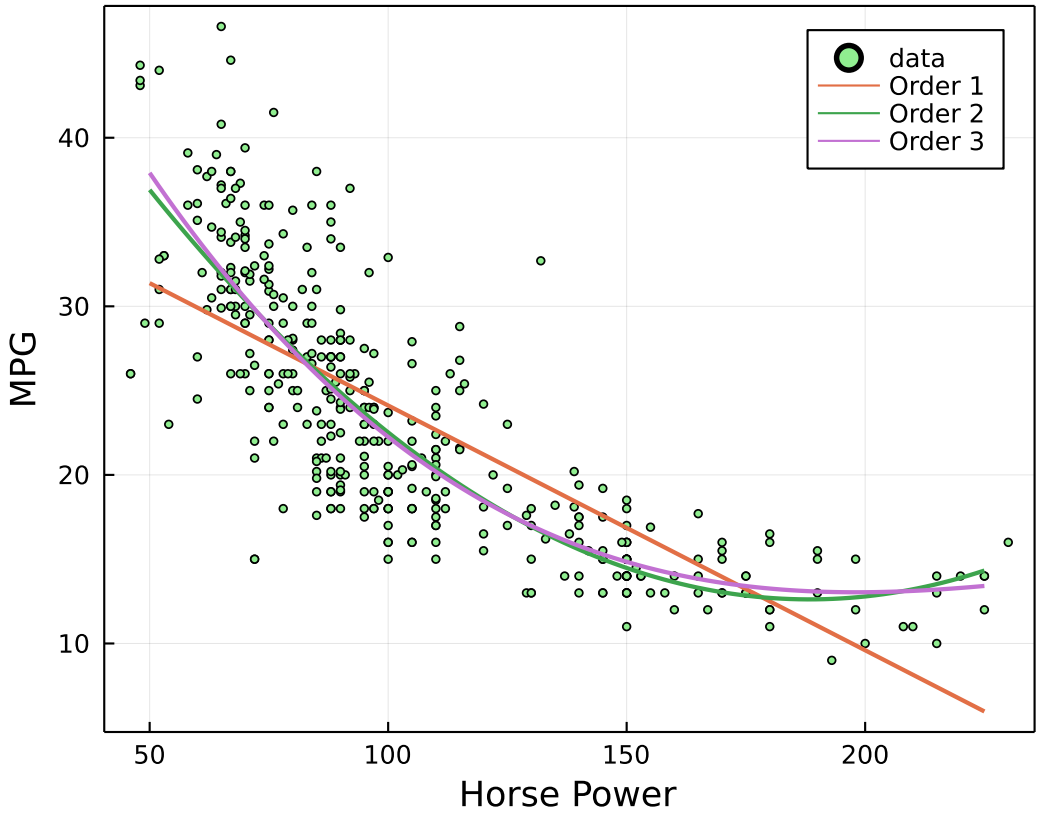
```
Solver: MLJLinearModels.Analytical _intercept = true, …), …). …), …).
  iterative: Bool false                                        ).
  max_inner: Int64 200
```

```
19.381831638657914
```

```
begin
    get_mse(lr) = rms(MLJ.predict(lr, rows=test), y[test])^2

    @show get_mse(lr1)
    @show get_mse(lr2)
    @show get_mse(lr3)
end
```

```
get_mse(lr1) = 23.493990895007986
get_mse(lr2) = 19.287175510952164
get_mse(lr3) = 19.381831638657914
```




```
begin
    hpn  = xx.Horsepower
    Xnew = DataFrame(hp1=hpn, hp2=hpn.^2, hp3=hpn.^3)
    yy1 = MLJ.predict(lr1, Xnew)
    yy2 = MLJ.predict(lr2, Xnew)
    yy3 = MLJ.predict(lr3, Xnew)
    scatter(X.Horsepower, y,ms=2, color=:lightgreen,label="data", xlabel="Horse
    Power",ylabel="MPG",size=(500,400),frame=:box)
    plot!(xx.Horsepower, [yy1,yy2,yy3], lw=2,label=["Order 1" "Order 2" "Order
    3"])

end
```