Импортируем нужные библиотеки

```
using Econometrics, RDatasets, GLM, GLMNet, Statistics, Plots, DataFrames,
StatsPlots
```

Первый взгляд на датасет

Для построения регрессионых моделей воспользуемся датасетом **Auto** из `RDatasets`

auto =

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Year	Or
1	18.0	8.0	307.0	130.0	3504.0	12.0	70.0	1.0
2	15.0	8.0	350.0	165.0	3693.0	11.5	70.0	1.0
3	18.0	8.0	318.0	150.0	3436.0	11.0	70.0	1.0
4	16.0	8.0	304.0	150.0	3433.0	12.0	70.0	1.0
5	17.0	8.0	302.0	140.0	3449.0	10.5	70.0	1.0
6	15.0	8.0	429.0	198.0	4341.0	10.0	70.0	1.0
7	14.0	8.0	454.0	220.0	4354.0	9.0	70.0	1.0
8	14.0	8.0	440.0	215.0	4312.0	8.5	70.0	1.0
9	14.0	8.0	455.0	225.0	4425.0	10.0	70.0	1.0
10	15.0	8.0	390.0	190.0	3850.0	8.5	70.0	1.0

more

```
auto = dataset("ISLR", "Auto")
```

Выведем некоторую статистическую информацию по колонкам

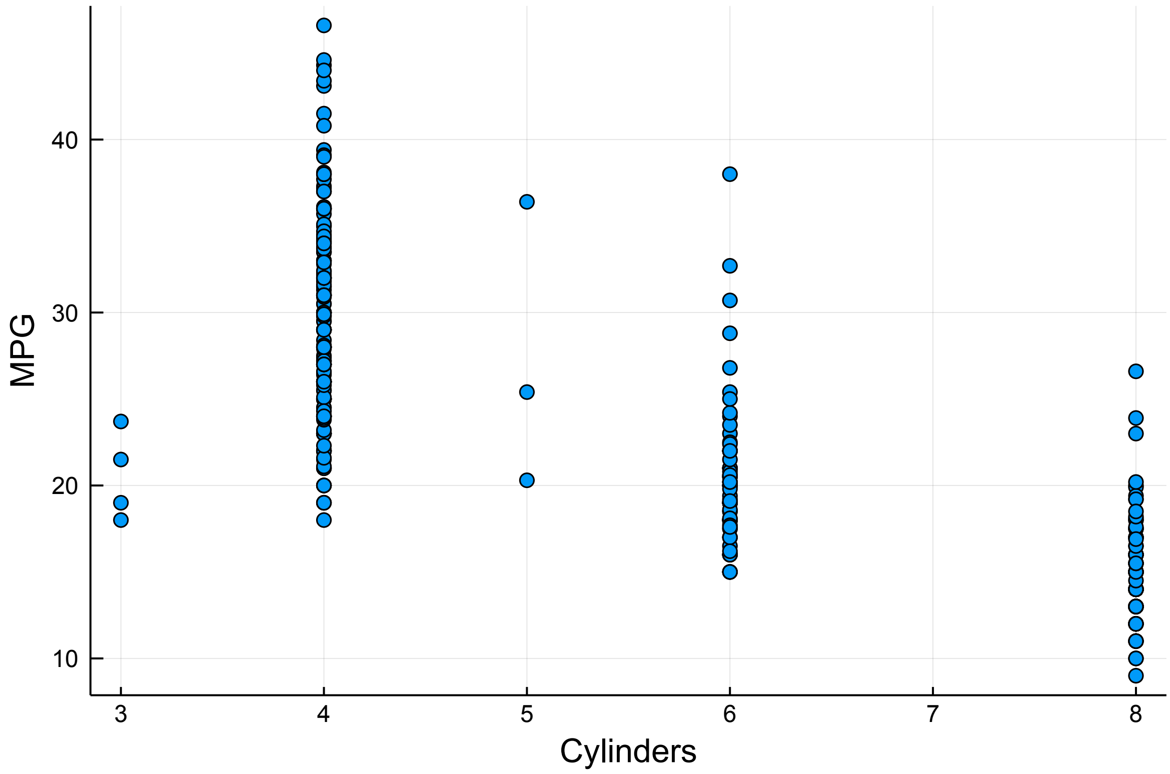
	variable	mean	min	median	max
1	:MPG	23.4459	9.0	22.75	46.6
2	:Cylinders	5.47194	3.0	4.0	8.0
3	:Displacement	194.412	68.0	151.0	455.0
4	:Horsepower	104.469	46.0	93.5	230.0
5	:Weight	2977.58	1613.0	2803.5	5140.0
6	:Acceleration	15.5413	8.0	15.5	24.8
7	:Year	75.9796	70.0	76.0	82.0

	variable	mean	min	median	max
8	:Origin	1.57653	1.0	1.0	3.0
9	:Name	nothing	"amc ambassador brougham"	nothing	"vw rabbit custo

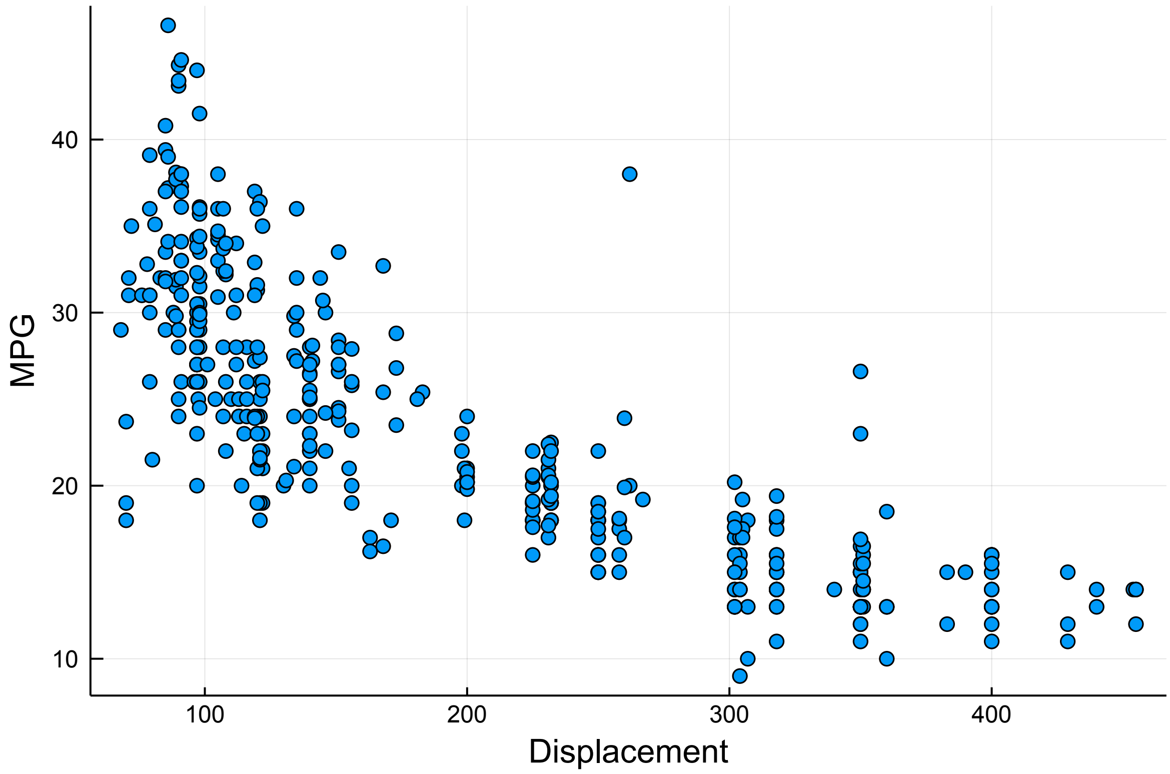
```
describe(auto)
```

MPG - это так называемые литры на 100 км. пути

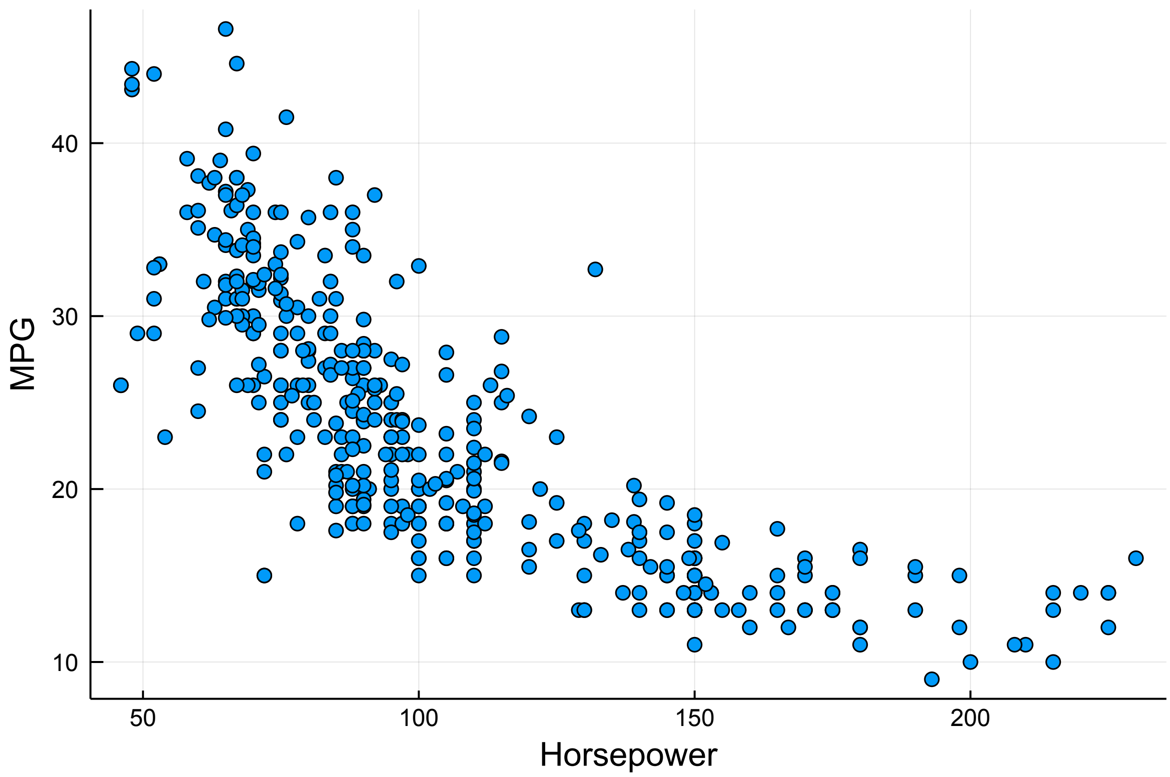
Сперва давайте посмотрим на графиках на зависимость между **MPG** и пятью первыми переменными



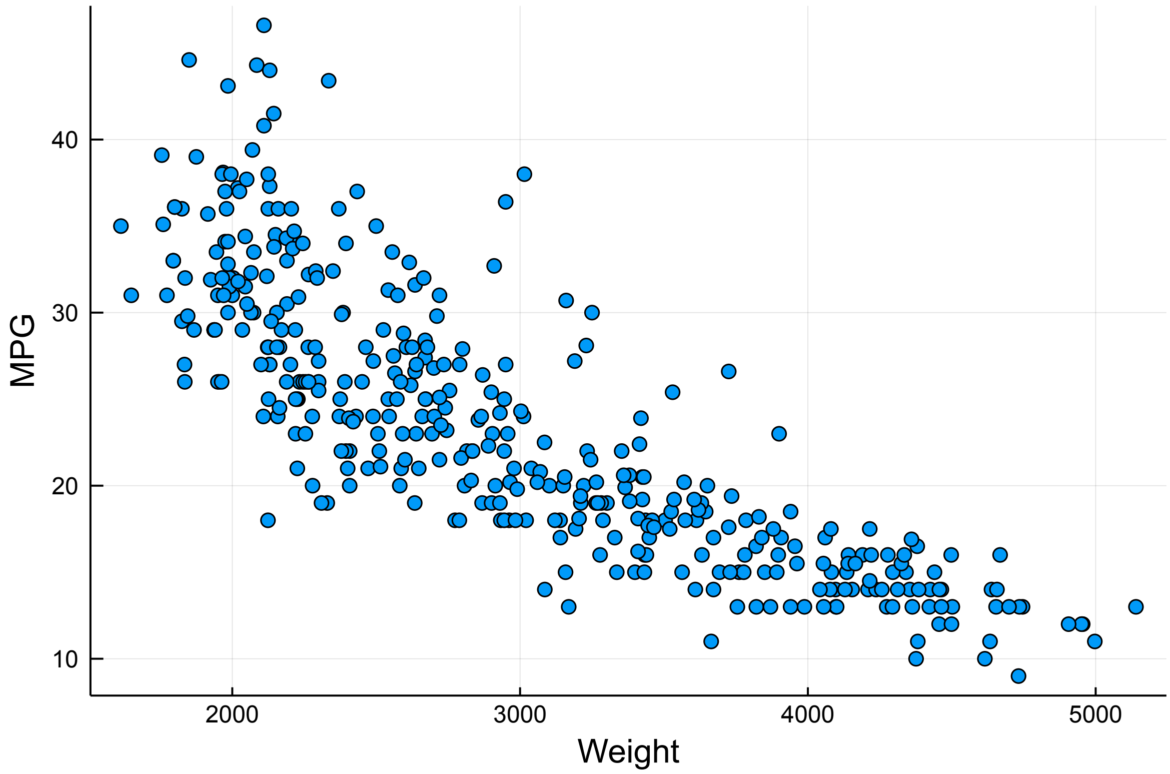

```
scatter(auto.Cylinders, auto.MPG, xaxis="Cylinders", yaxis="MPG", legend = false)
```



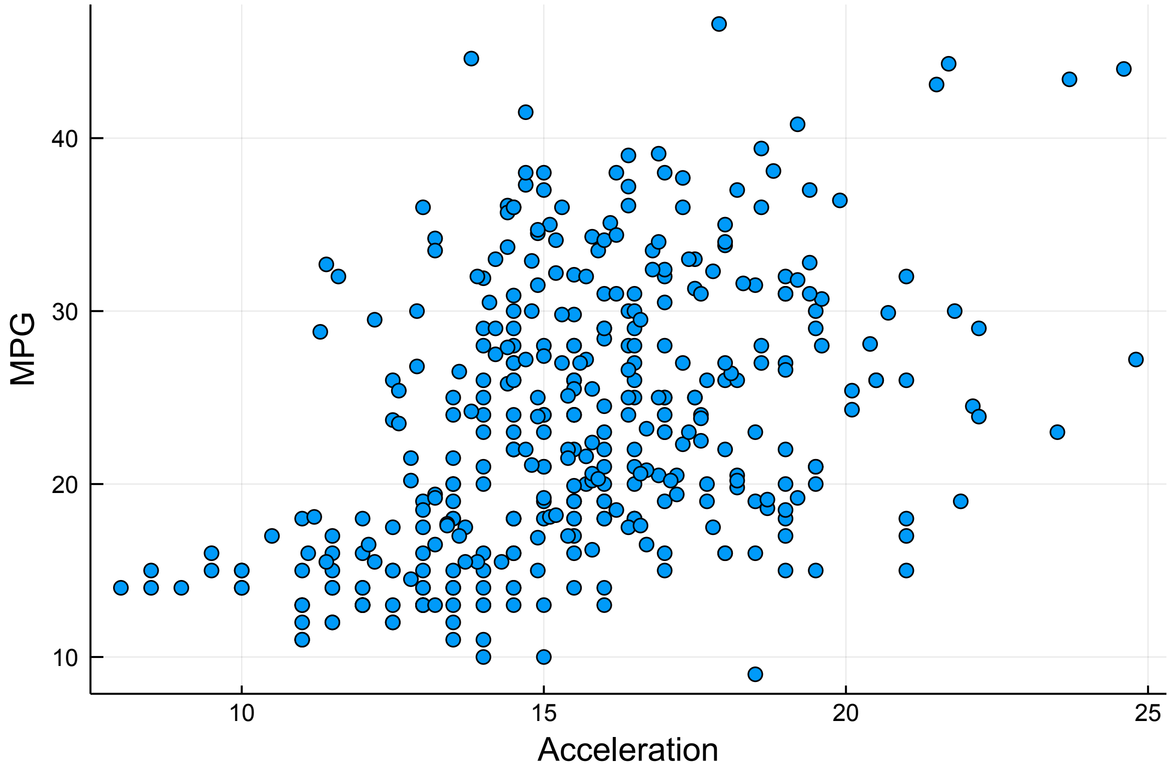

```
scatter(auto.Displacement, auto.MPG, xaxis="Displacement", yaxis="MPG", legend =
false)
```




```
scatter(auto.Horsepower, auto.MPG, xaxis="Horsepower", yaxis="MPG", legend = false)
```



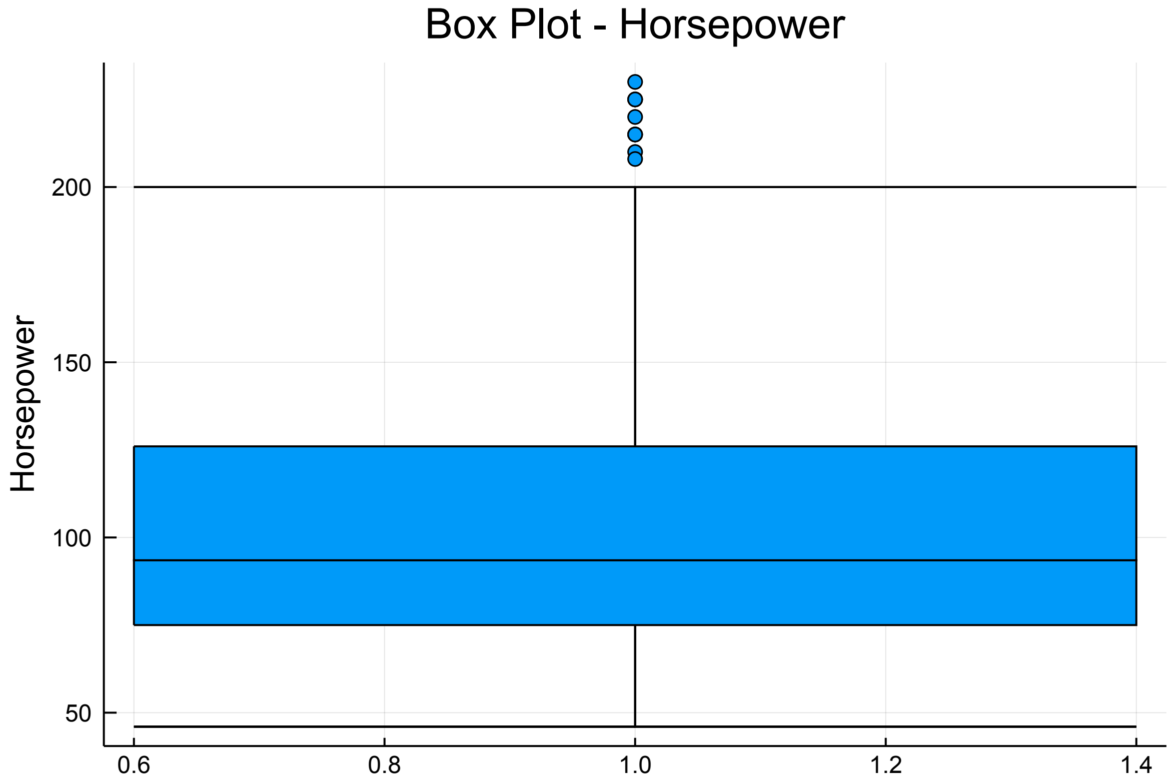

```
scatter(auto.Weight, auto.MPG, xaxis="Weight", yaxis="MPG", legend = false)
```




```
scatter(auto.Acceleration, auto.MPG, xaxis="Acceleration", yaxis="MPG", legend =
false)
```

Важно также посмотреть на *outliers* - так называемые выбросы. Подробности и тонкости работы с данными представлены в другом туториале, тут же мы посмотрим только на один важный момент :)

Можно построить **box plot** и после удалить выбросы, которые видны на графике




```
boxplot(auto.Horsepower, title = "Box Plot - Horsepower", ylabel = "Horsepower",
legend = false)
```

Модели в `GLM`

Для начала воспользуемся пакетом `GLM`. Начнем с простой парной регрессии:

$$\widehat{MPG} = \widehat{\beta_0} + \widehat{\beta_1} \cdot Horsepower$$

```
linear_model =
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Array{Float64,1}},GLM.DensePredCh

MPG ~ 1 + Horsepower

Coefficients:
```

	Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	Upper 95%
(Intercept)	39.9359	0.717499	55.66	<1e-99	38.5252	41.3465
Horsepower	-0.157845	0.0064455	-24.49	<1e-80	-0.170517	-0.145172

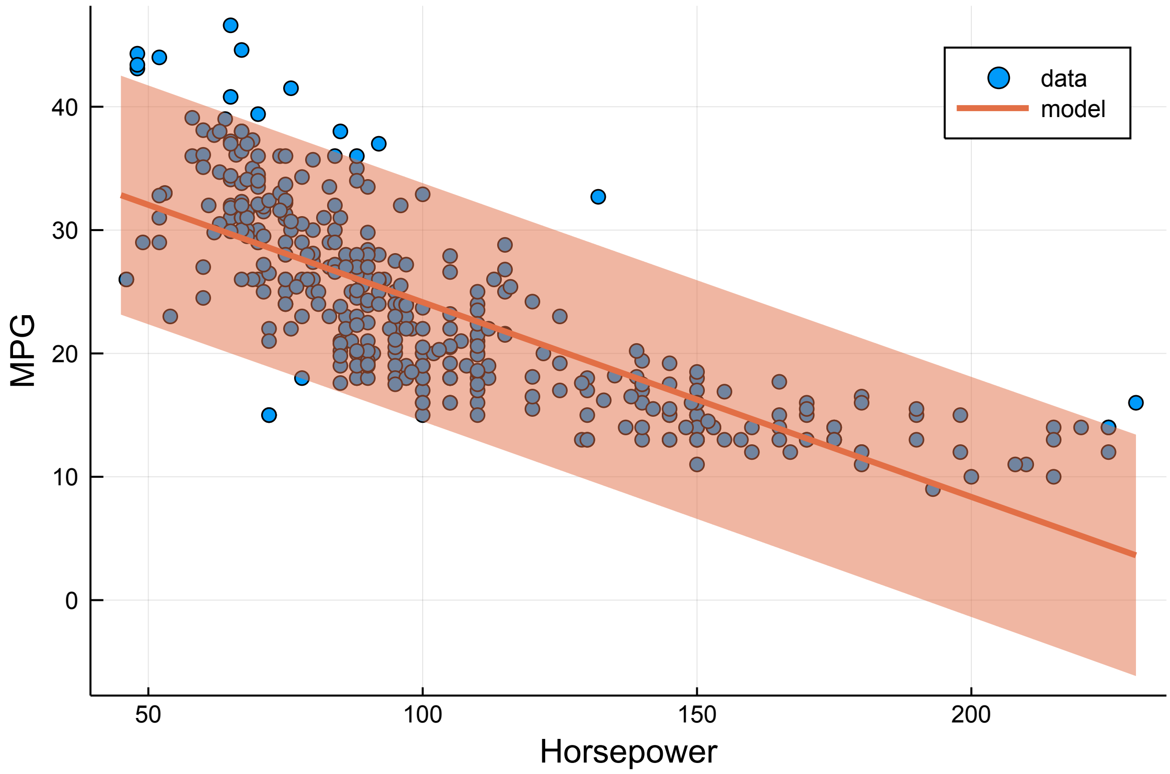
```
linear_model = lm(@formula(MPG ~ Horsepower), auto)
```

Выведем коэффициент детерминации R^2

```
0.6059482578894348
```

```
GLM.r2(linear_model)
```

Horsepower объясняет примерно 60,6% дисперсии разброса MPG. Вполне неплохо, построим график с данными, регрессией и доверительным интервалом для предсказания с уровнем доверия 95%




```
begin
    pred = DataFrame(Horsepower = 45:0.1:230);
    pr = GLM.predict(linear_model, pred, interval = :prediction, level = 0.95);
    plot(xlabel="Horsepower", ylabel="MPG", legend=:best)
    plot!(auto.Horsepower, auto.MPG, label="data", seriestype=:scatter)
    plot!(pred.Horsepower, pr.prediction, label="model", linewidth=3,
        ribbon = (pr.prediction .- pr.lower, pr.upper .- pr.prediction))
end
```

По графику видим, что связь возможно гиперболическая. Рассмотрим модель

$$\widehat{MPG} = \widehat{\beta_0} + \widehat{\beta_1} \cdot \frac{1}{Horsepower}$$

```
hyperbolic_model =
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Array{Float64,1}},GLM.DensePredCh

MPG ~ 1 + :(1 / Horsepower)

Coefficients:
```

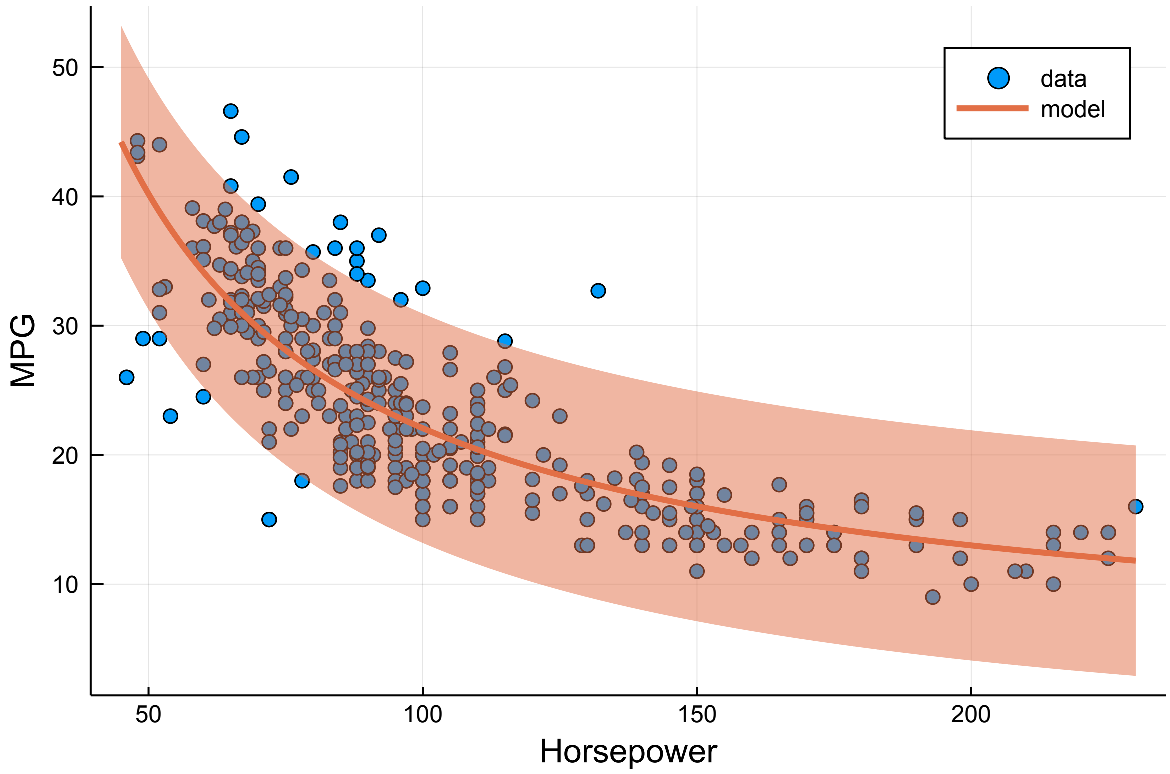
	Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	Upper 95%
(Intercept)	3.93552	0.734109	5.36	<1e-6	2.49221	5.37882
1 / Horsepower	1812.99	64.8509	27.96	<1e-94	1685.49	1940.49

```
hyperbolic_model = lm(@formula(MPG ~ 1 / Horsepower), auto)
```

```
0.6671084785591674
```

```
GLM.r2(hyperbolic_model)
```

Эта модель, похоже, и правда лучше объясняет **MPG**. Построим аналогичный график




```
begin
    predd = DataFrame(Horsepower = 45:0.1:230);
    prd = GLM.predict(hyperbolic_model, pred, interval = :prediction, level = 0.95);
    plot(xlabel="Horsepower", ylabel="MPG", legend=:best)
    plot!(auto.Horsepower, auto.MPG, label="data", seriestype=:scatter)
    plot!(predd.Horsepower, prd.prediction, label="model", linewidth=3,
        ribbon = (prd.prediction .- prd.lower, prd.upper .- prd.prediction))
end
```

Теперь перейдем к множественной регрессии.

Мы видим, что есть линейная связь между **MPG** и **Horsepower** и **MPG** и **Weight**

```
reg =
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Array{Float64,1}},GLM.DensePredCh

MPG ~ 1 + Weight + Horsepower

Coefficients:
```

	Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	Upper 95%
(Intercept)	45.6402	0.793196	57.54	<1e-99	44.0807	47.1997
Weight	-0.00579416	0.000502327	-11.53	<1e-25	-0.00678177	-0.00480654
Horsepower	-0.0473029	0.0110851	-4.27	<1e-4	-0.069097	-0.0255087

```
reg = lm(@formula(MPG ~ Weight + Horsepower), auto)
```

Видим достаточно знакомую и родную табличку! Но фанаты **R** и **Stata** могут справедливо заметить, что как-то маловато всего представлено в таблице. Например, не видно коэффициента детерминации. Не унывать! **Julia** - за минимализм. Да и вывести всё, что Вас интересует, невероятно просто. Достаточно лишь указать интересующий метод и применить его к нашей модели.

```
R2 = 0.7063752737298348
```

```
R2 = GLM.r2(reg)
```

coefs = Float64[45.6402, -0.00579416, -0.0473029]

```
coefs = GLM.coef(reg)
```

Что ещё можно посмотреть, применив метод к модели?

dof_residual: степени свободы остатков (если имеет смысл)

stderror: стандартные ошибки коэффициентов

vcov: ковариационная матрица оценок коэффициентов

И много всего другого! Интересующиеся могут обратиться к документации `GLM`.

А теперь давайте предсказать значение, задав заранее значения **Horsepower** и **Weight**.

new_data =

	Horsepower	Weight
1	200	4000
2	250	3800

```
new_data = DataFrame(Horsepower = [200, 250], Weight = [4000, 3800])
```

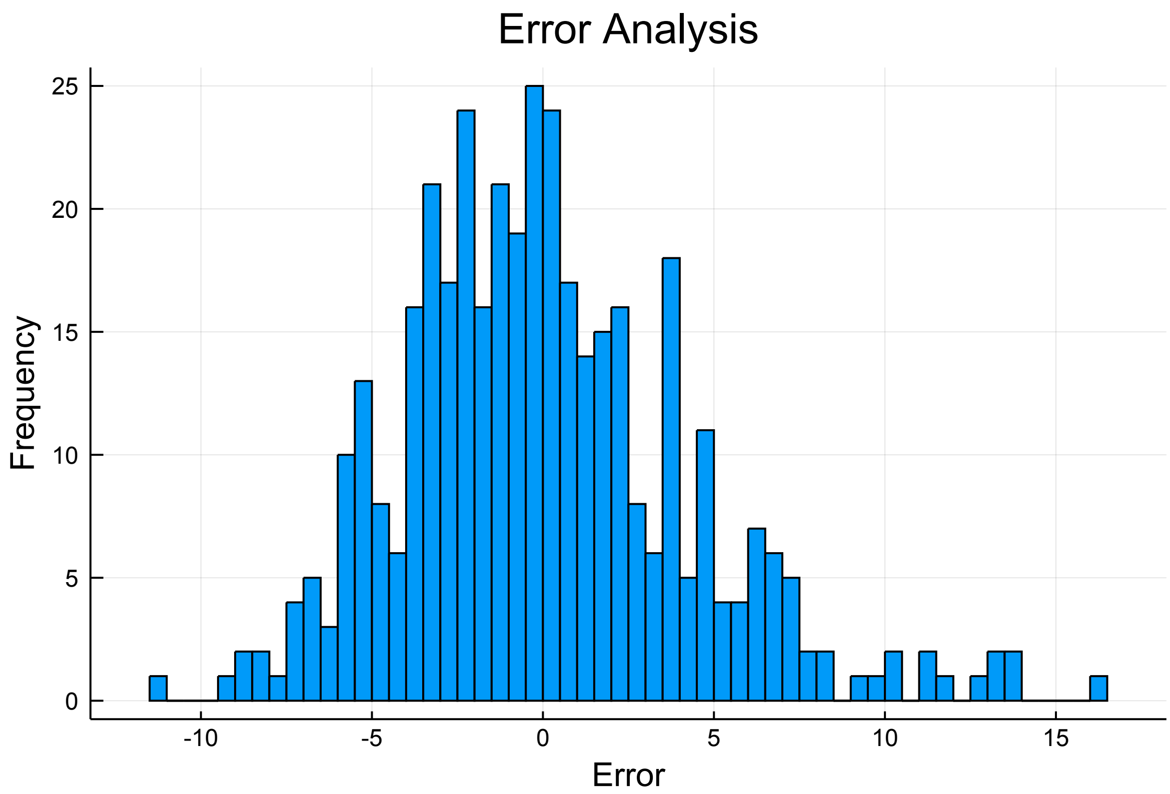
prediction_GLM =

	MPG_pred
1	13.003
2	11.7967

```
prediction_GLM = DataFrame(MPG_pred = GLM.predict(reg, new_data))
```

Можем также посмотреть на имеющиеся наблюдения и сравнить с предсказанными значениями

Посмотрим на распределение ошибок




```
begin
    performance = DataFrame(y_actual = auto.MPG, y_predicted = GLM.predict(reg));
    error = performance[:y_actual] - performance[:y_predicted];
    histogram(error, bins = 50, title = "Error Analysis", ylabel = "Frequency",
    xlabel = "Error",legend = false)
end
```

Ошибки распределены нормально, судя по гистограмме

Модели в `Econometrics`

Также рассмотрим пакет `Econometrics`, который позволяет строить линейные регрессии, но немного отличается от `GLM`. Конечно, основные различия можно увидеть при более глубоком анализе, нежели тот, что мы приводим в нашем туториале, поэтому заинтересовавшиеся могут обратиться к документации пакета.

```
model =
Continuous Response Model
Number of observations: 392
Null Loglikelihood: -1361.19
Loglikelihood: -1121.01
R-squared: 0.7064
LR Test: 480.37 ~ χ²(2) ⇒  Pr > χ² = 0.0000
Formula: MPG ~ 1 + Weight + Horsepower
Variance Covariance Estimator: OIM
```

	PE	SE	t-value	Pr > |t|	2.50%	97.50%
(Intercept)	45.6402	0.793196	57.5397	<1e-99	44.0807	47.1997
Weight	-0.00579416	0.000502327	-11.5346	<1e-25	-0.00678177	-0.00480654
Horsepower	-0.0473029	0.0110851	-4.26725	<1e-4	-0.069097	-0.0255087

```
model = fit(EconometricModel, @formula(MPG ~ Weight + Horsepower), auto)
```

Фанаты **R** и **Stata** могут ликовать! Теперь мы видимт табличку, в которой есть и коэффициент детерминации, и LR-тест, и много всего интересного для благородных Донов!

Сравним с аналогичной моделью из `GLM`

```
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Array{Float64,1}},GLM.DensePredCh

MPG ~ 1 + Weight + Horsepower

Coefficients:
```

	Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	Upper 95%
(Intercept)	45.6402	0.793196	57.54	<1e-99	44.0807	47.1997
Weight	-0.00579416	0.000502327	-11.53	<1e-25	-0.00678177	-0.00480654
Horsepower	-0.0473029	0.0110851	-4.27	<1e-4	-0.069097	-0.0255087

```
reg
```

Как мы видим, все статистические характеристики коэффициентов идентичны, только в `Econometrics` выводится еще разная полезная информация

F-тест на сравнение моделей

Рассмотрим две модели:

$$\widehat{MPG}_1 = \widehat{\beta_0} + \widehat{\beta_1} \cdot Weight + \widehat{\beta_2} \cdot Cylinders$$

$$\widehat{MPG}_2 = \widehat{\gamma_0} + \widehat{\gamma_1} \cdot Weight + \widehat{\gamma_2} \cdot Cylinders + \widehat{\gamma_3} \cdot Horsepower$$

F-тест позволит нам сравнить данные модели, и решить, является ли коэффициент γ_3 статистически значимым. Для большей ясности запишем гипотезы:

$$H_0 : \gamma_3 = 0$$

$$H_1 : \gamma_3 \neq 0$$

С помощью библиотеки `GLM` проведем F-тест

```
unrestricted_model =
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Array{Float64,1}},GLM.DensePredCh

MPG ~ 1 + Weight + Cylinders + Horsepower

Coefficients:
```

	Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	Upper 95%
(Intercept)	45.7368	0.795957	57.46	<1e-99	44.1719	47.3017
Weight	-0.0052723	0.000642353	-8.21	<1e-14	-0.00653523	-0.00400937
Cylinders	-0.388974	0.29883	-1.30	0.1938	-0.976504	0.198555
Horsepower	-0.0427277	0.0116196	-3.68	0.0003	-0.0655729	-0.0198824

```
unrestricted_model = lm(@formula(MPG ~ Weight + Cylinders + Horsepower), auto)
```

```
restricted_model =
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Array{Float64,1}},GLM.DensePredCh

MPG ~ 1 + Weight + Cylinders

Coefficients:
```

	Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	Upper 95%
(Intercept)	46.2923	0.793969	58.30	<1e-99	44.7313	47.8533
Weight	-0.00634711	0.000581133	-10.92	<1e-23	-0.00748967	-0.00520456
Cylinders	-0.721378	0.289378	-2.49	0.0131	-1.29032	-0.152437

```
restricted_model = lm(@formula(MPG ~ Weight + Cylinders), auto)
```

```
F-test: 2 models fitted on 392 observations
```

	DOF	ΔDOF	SSR	ΔSSR	R²	ΔR²	F*	p(>F)
[1]	5		6963.4376		0.7077			
[2]	4	-1	7206.1149	242.6773	0.6975	-0.0102	13.5219	0.0003

```
GLM.ftest(unrestricted_model.model, restricted_model.model)
```

Как мы видим, p-value, или p(>F), равен 0.0003, то есть он меньше любого разумного уровня значимости, отсюда $\gamma_3 \neq 0$ при любом разумном уровне значимости

Регуляризация

Так же в **Julia** можно построить модели линейной регрессии с L1-регуляризацией (то есть **LASSO**) и L2-регуляризацией (то есть **Ridge**). Для этого воспользуемся библиотекой `GLMNet`. Заодно введем квадраты пяти переменных, поэлементно возводя их в квадрат (просто потому что)

```
Float64[144.0, 132.25, 121.0, 144.0, 110.25, 100.0, 81.0, 72.25, 100.0, 72.25, 1(
```

```
begin
    auto["Cylinders_sq"] = auto["Cylinders"].^2;
    auto["Displacement_sq"] = auto["Displacement"].^2;
    auto["Horsepower_sq"] = auto["Horsepower"].^2;
    auto["Weight_sq"] = auto["Weight"].^2;
    auto["Acceleration_sq"] = auto["Acceleration"].^2;
end
```

Важное уточнение: `GLMNet` работает не с датафреймами, а с массивами, поэтому выделим нужные признаки и передевем их в массивы

```
X = 392×9 Array{Float64,2}:
 307.0  130.0  3504.0  12.0  70.0   94249.0  16900.0  1.2278e7   144.0
 350.0  165.0  3693.0  11.5  70.0  122500.0  27225.0  1.36382e7  132.25
 318.0  150.0  3436.0  11.0  70.0  101124.0  22500.0  1.18061e7  121.0
 304.0  150.0  3433.0  12.0  70.0   92416.0  22500.0  1.17855e7  144.0
 302.0  140.0  3449.0  10.5  70.0   91204.0  19600.0  1.18956e7  110.25
 429.0  198.0  4341.0  10.0  70.0  184041.0  39204.0  1.88443e7  100.0
 454.0  220.0  4354.0   9.0  70.0  206116.0  48400.0  1.89573e7   81.0
   ⋮                                          ⋮
 151.0   90.0  2950.0  17.3  82.0   22801.0   8100.0  8.7025e6   299.29
 140.0   86.0  2790.0  15.6  82.0   19600.0   7396.0  7.7841e6   243.36
  97.0   52.0  2130.0  24.6  82.0    9409.0   2704.0  4.5369e6   605.16
 135.0   84.0  2295.0  11.6  82.0   18225.0   7056.0  5.26702e6  134.56
```

```
120.0   79.0  2625.0  18.6  82.0   14400.0   6241.0  6.89062e6  345.96
119.0   82.0  2720.0  19.4  82.0   14161.0   6724.0  7.3984e6   376.36
```

```
X = hcat(Array(auto[:, 3:7]), Array(auto[:, 11:14]))
```

```
y =
Float64[18.0, 15.0, 18.0, 16.0, 17.0, 15.0, 14.0, 14.0, 14.0, 15.0, 15.0, 14.0,
```

```
y = Array(auto[:, 1])
```

Сначала инициализируем модель **LASSO**, потом **Ridge**.

Подбираем оптимальные λ с помощью кросс-валидации

```
lasso = Least Squares GLMNet Cross Validation
        92 models for 9 predictors in 10 folds
        Best λ 0.001 (mean loss 8.525, std 0.565)
```

```
lasso = glmnetcv(X, y, alpha = 1)
```

```
ridge = Least Squares GLMNet Cross Validation
        100 models for 9 predictors in 10 folds
        Best λ 0.649 (mean loss 11.332, std 0.966)
```

```
ridge = glmnetcv(X, y, alpha = 0)
```

Хочется сравнить эти модели, напишем простую функцию для рассчета коэффциента детерминации. Тут нам нужна функция mean() из пакета `Statistics`

```
r2 (generic function with 1 method)
```

```
function r2(model, X, y)
    RSS = sum((GLMNet.predict(model, X) - y).^2);
    TSS = sum((y .- Statistics.mean(y)).^2);
    (TSS - RSS)/TSS;
end
```

```
0.8669431269565578
```

```
r2(lasso, X, y)
```

```
0.8179318244679907
```

```
r2(ridge, X, y)
```

Тут модель **LASSO** показала себя лучше