Linear regression

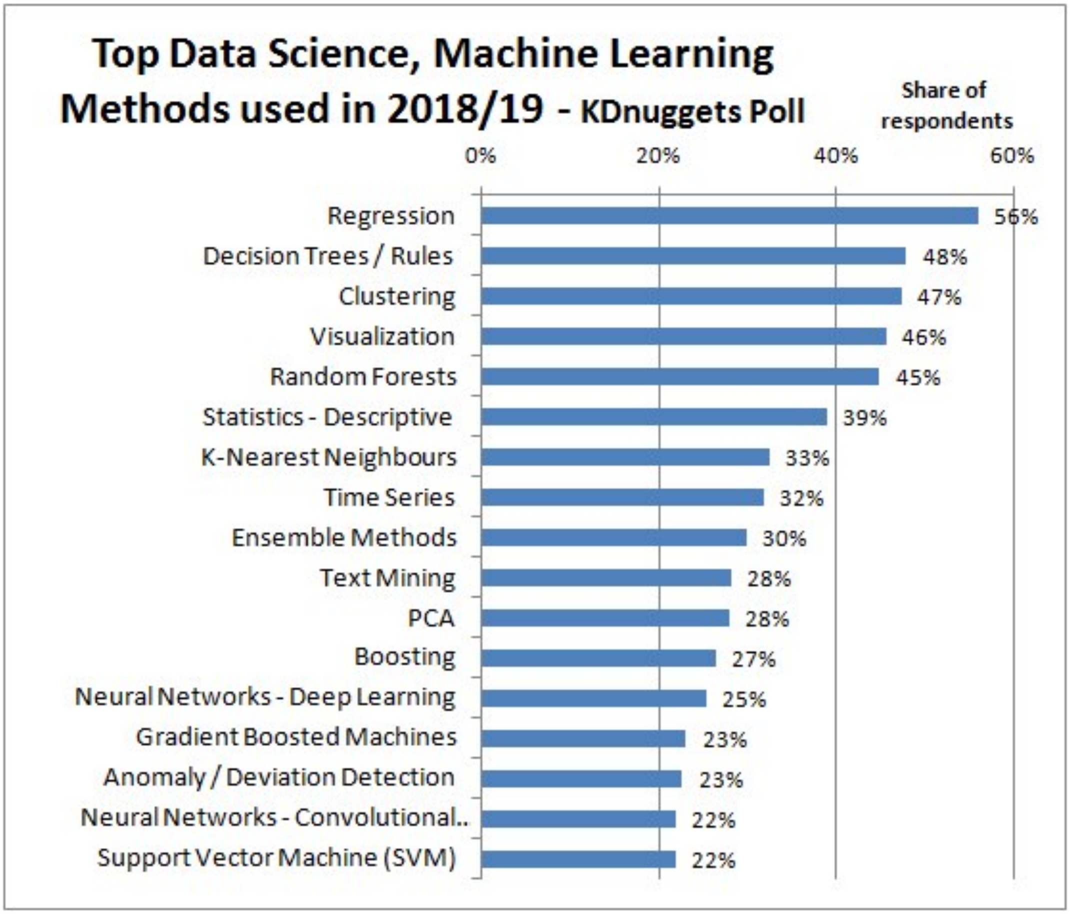
Plan

1. Recap: Statistical Inference
2. Motivations about Linear Regression
3. Linear regression (visual approach with `seaborn`)
4. Linear regression (with `statsmodels`)
5. Conditions for Inference
6. Multivariate Linear Regression

Recap: Statistical Inference

- Sampling distribution of the *mean* μ
  - $\hat{\mu}$ = plausible distribution of values for μ
  - [95% confidence intervals]
- Hypothesis testing
  - `p-value` : "probability that what you observed is just due to pure chance"
  - significance level $\alpha = 0.05$
- Central Limit Theorem extended
  - z-test for normal $\mathcal{N}$ distributions
  - t-tests for student $\mathcal{T}_\nu$ distribution
- Bayesian inference
  - $posterior \sim prior * likelihood$
  - Maximum Likelihood Estimate (MLE)
  - Maximim A Posteriori Estimate (MAP)

1. Motivation



- Most important applied statistical tool
- Interpretable
- Standard practice for `causal inference`

[Source KDnuggets (https://www.kdnuggets.com/2019/04/top-data-science-machine-learning-methods-2018-2019.html)](https://www.kdnuggets.com/2019/04/top-data-science-machine-learning-methods-2018-2019.html) (800 participants)

Recall our business problem 👇

> *How to increase customer satisfaction while maintaining a healthy order volume?*

`Linear Regression` will help us analyse:

1. What features impact `review_score` the most?
2. How to control the **confounding factors**?

2. Simple Linear Regression (visual approach with `seaborn`)

The mpg (miles per gallon) dataset

🥋 Let's take an example!

🚗 The **`mpg`** dataset

👉 Contains ~400 models of car statistics from 1970 to 1982

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

mpg = sns.load_dataset("mpg").dropna()
mpg.head()
```

Out[ ]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	na
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevro chev mal
1	15.0	8	350.0	165.0	3693	11.5	70	usa	bu skyl 3
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymo satel
3	16.0	8	304.0	150.0	3433	12.0	70	usa	a rebel
4	17.0	8	302.0	140.0	3449	10.5	70	usa	f tor

Full description of the dataset is [here (https://data.world/dataman-udit/cars-data)](https://data.world/dataman-udit/cars-data) 🏎

The columns we will focus on are:

- `mpg` : miles per gallon
- `cylinders`
- `displacement` : volume of all the pistons (in cc)
- `horsepower`
- `weight` : pounds (lbs)
- `acceleration` : zero to sixty miles per hour (in seconds)

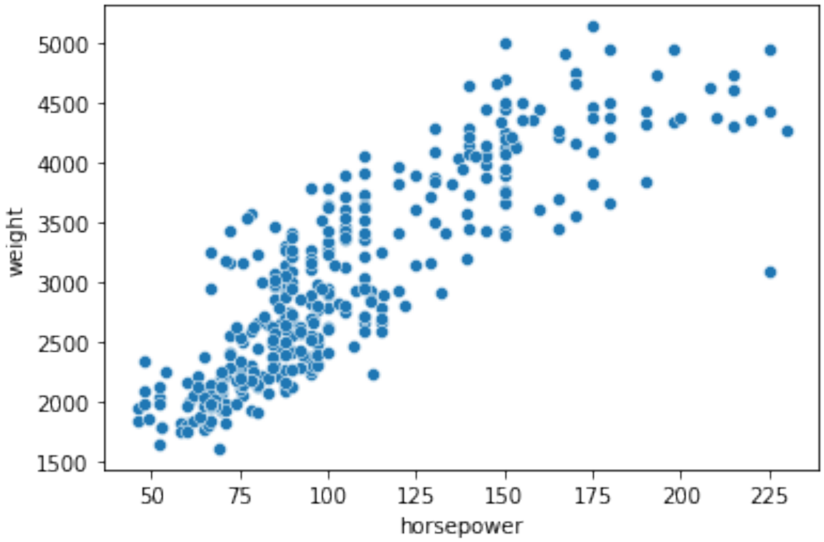
In [ ]:
```
mpg.describe().apply(lambda x: round(x))
```

Out[ ]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
count	392	392	392	392	392	392	392
mean	23	5	194	104	2978	16	76
std	8	2	105	38	849	3	4
min	9	3	68	46	1613	8	70
25%	17	4	105	75	2225	14	73
50%	23	4	151	94	2804	16	76
75%	29	8	276	126	3615	17	79
max	47	8	455	230	5140	25	82

Regress `weight` on `horsepower` ?

In [ ]:
```
sns.scatterplot(x='horsepower', y='weight', data=mpg);
```



❓ Find a regression line
$\hat{y}$
that is the **closest** to the *weights*

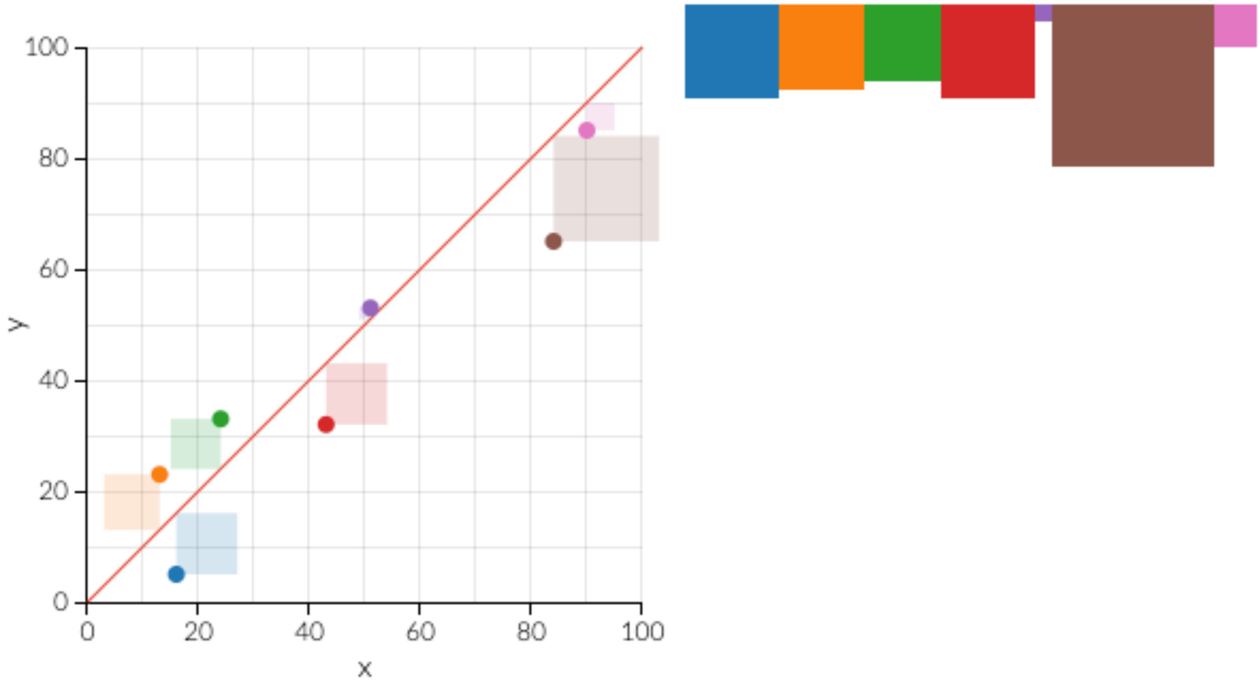
🤓 Find
$\beta = (\beta_0, \beta_1)$
that minimizes the **norm**
$\|weights - (\beta_0 + \beta_1 horsepowers)\|$

Ordinary Least Square (OLS) regression

- uses the "natural" L_2 **`Euclidian Distance`**
- solves the β that minimizes **`Sum of Squared Residuals (SSR)`**

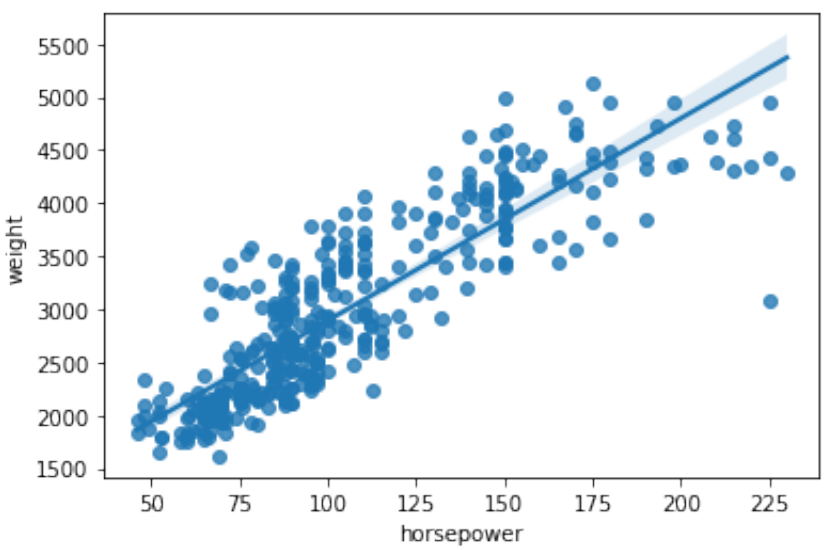
$$\underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{392} (weight_i - (\beta_0 + \beta_1 horsepowers))^2$$

👉 [Dynamic vizualisation (http://setosa.io/ev/ordinary-least-squares-regression/)](http://setosa.io/ev/ordinary-least-squares-regression/)



⚠️ ***OLS is very sensitive to outliers!***

In [ ]:
```
sns.regplot(x='horsepower', y='weight', data=mpg);
```



Interpretation

❌ "Higher horsepower causes higher weight"

✅ "Powerful cars seem heavier"

- By how much? Measured by the **`slope`** of the line = β_1

✅ "Horsepower seems to explain a good deal of the weights' variations"

- How much? Measured by the **`correlation coefficient`** $\rho \in [-1, 1]$

In [ ]:
```
round(mpg.corr(numeric_only=True), 2)
```

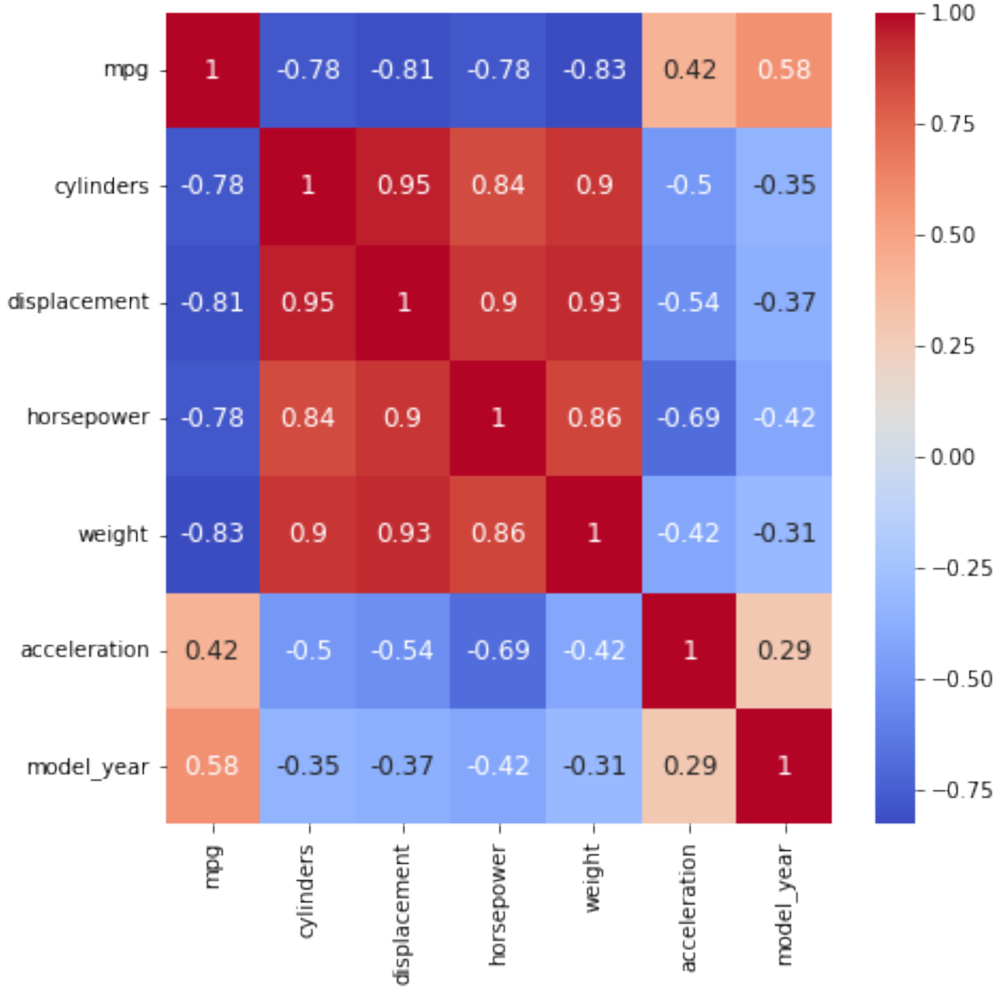
Out[ ]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
mpg	1.00	-0.78	-0.81	-0.78	-0.83	0.42	0.58
cylinders	-0.78	1.00	0.95	0.84	0.90	-0.50	-0.35
displacement	-0.81	0.95	1.00	0.90	0.93	-0.54	-0.37
horsepower	-0.78	0.84	0.90	1.00	0.86	-0.69	-0.42
weight	-0.83	0.90	0.93	0.86	1.00	-0.42	-0.31
acceleration	0.42	-0.50	-0.54	-0.69	-0.42	1.00	0.29
model_year	0.58	-0.35	-0.37	-0.42	-0.31	0.29	1.00

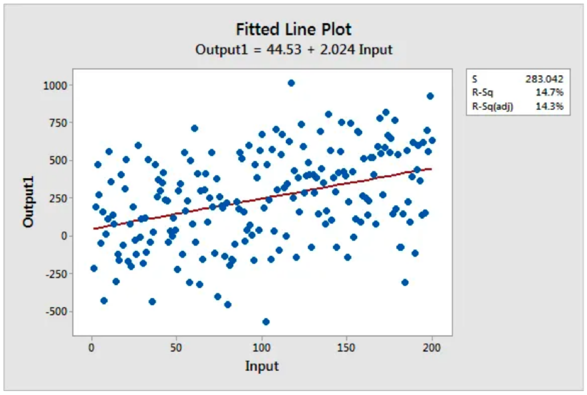
In [ ]:
```
## R-squared (r2) is often preferred, from [0 to 1]
print('R-Squared = ', (mpg.corr(numeric_only=True)['weight']['horsepow
er'])**2)
```

```
R-Squared =  0.7474254996898221
```

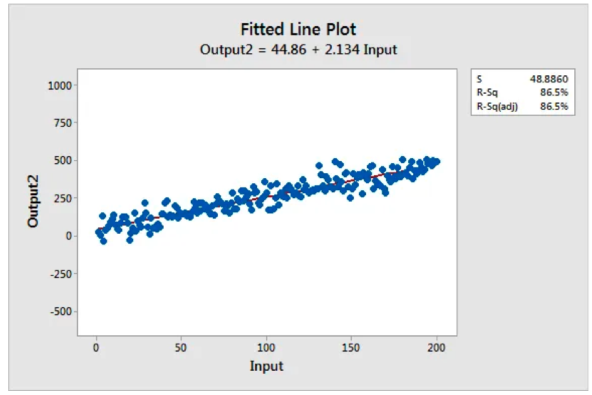
In [ ]:
```
plt.figure(figsize = (7,7))
sns.heatmap(round(mpg.corr(numeric_only=True), 2),
            cmap="coolwarm", annot=True, annot_kws={"size":12});
```



R^2
(explanation of the variance)



- **R2** low (15%)



- **R2** high (86%)

- % of the variance of `weights` that is explainable by the variance of `horsepower`
- Ranges from 0 (explains nothing) to 1 (perfect relationship)

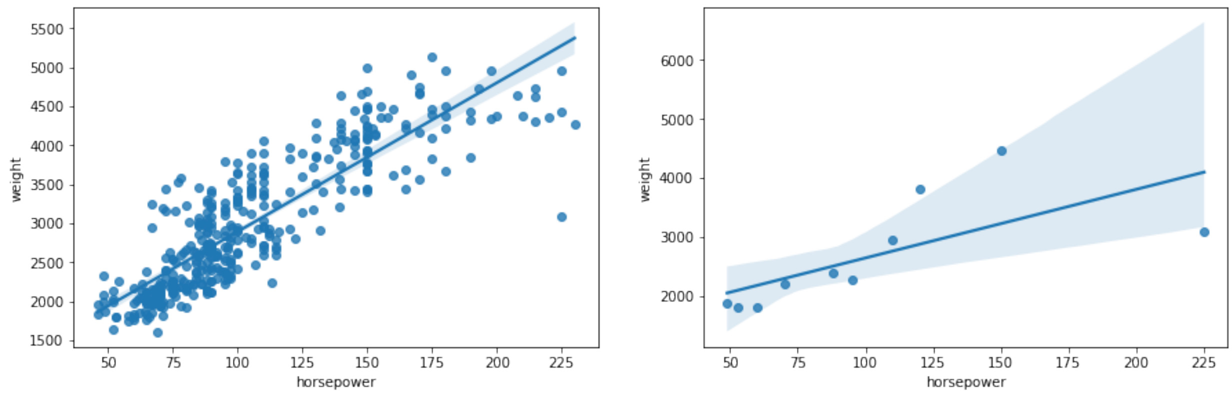
📚 [Interpret R-squared - Statistics by Jim (https://statisticsbyjim.com/regression/interpret-r-squared-regression/)](https://statisticsbyjim.com/regression/interpret-r-squared-regression/)

❓ *Am I confident that this relationship **generalizes** well to all car models in the world* ❓

Measured by 👇

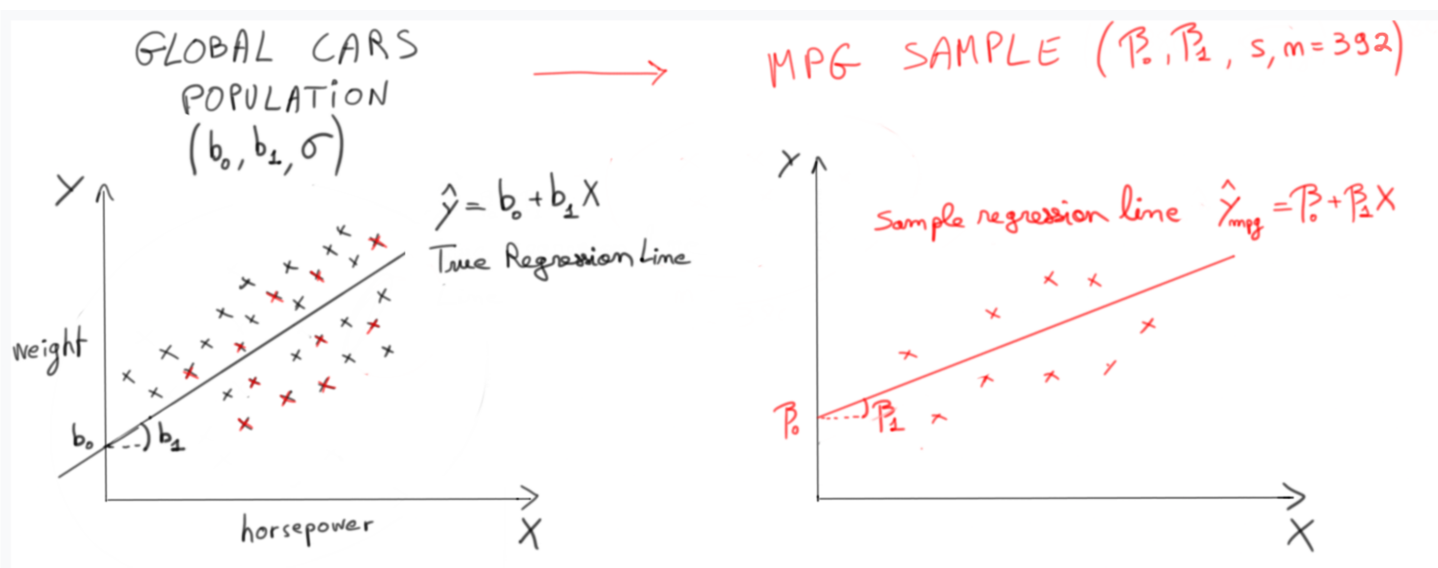
- confidence intervals
- p-values associated with hypothesis testing

In [ ]:
```
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.regplot(x='horsepower', y='weight', data=mpg, ci=95)
plt.subplot(2,2,2)
sns.regplot(x='horsepower', y='weight', data=mpg.sample(10, random_sta
te=6), ci=95);
```



Imagine for a moment the mpg dataset was made of only 10 cars, as per sampled above 👆

- **What if you had found a negative correlation** with horsepower ❗ ❓
- How much would you trust the regression coefficients ❗ ❓



3. Simple Linear Regression (with `statsmodels`)

Statsmodels

👉 [statsmodels.org (http://www.statsmodels.org/stable/index.html)](http://www.statsmodels.org/stable/index.html)

```
pip install statsmodels
```

- Simple Linear ML models + Statistical Inference
- Very easy to use
- ~ Replace **[R (https://www.r-project.org/)](https://www.r-project.org/)** in Python

Two ways to use statsmodels

`"Standard" API` :

```python
import statsmodels.api as sm
Y = mpg['weight']
X = mpg['horsepower']
model = sm.OLS(Y, X).fit() # Finds the best beta
model.predict(X) # The Y_pred (regression-line)
```

`"Formula" API`
(more intuitive):

```python
import statsmodels.formula.api as smf
model = smf.ols(formula = 'weight ~ horsepower', data=data).fit()
```

Formula uses the [patsy (https://patsy.readthedocs.io/en/latest/formulas.html)](https://patsy.readthedocs.io/en/latest/formulas.html) syntax derived from `R`

In [ ]:
```python
# Instantiate a model
model = smf.ols(formula='weight ~ horsepower', data=mpg)

# Train the model to find the best line
model = model.fit()
model
```

Out[ ]:
```
<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x1
77bc4760>
```

Interpretation

In [ ]:
```
print(model.params)
```

```
Intercept     984.500327
horsepower     19.078162
dtype: float64
```

👉 Horsepower
β_1
: "For each increase of 1 horsepower, a car's weight increases on average by 19 lbs (pounds)"

👉 Intercept
β_0
: "a car with 0 horsepower would weigh 984 lbs"

In [ ]:
```
model.rsquared
```

Out[ ]: `0.7474254996898198`

👉 74% of the variance of `weight` is explained by the variance of `horsepower`

In [ ]:
```
model.summary()
```

Out[ ]:

OLS Regression Results

Dep. Variable:	weight	**R-squared:**	0.747
Model:	OLS	**Adj. R-squared:**	0.747
Method:	Least Squares	**F-statistic:**	1154.
Date:	Wed, 14 Sep 2022	**Prob (F-statistic):**	1.36e-118
Time:	16:17:41	**Log-Likelihood:**	-2929.9
No. Observations:	392	**AIC:**	5864.
Df Residuals:	390	**BIC:**	5872.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	984.5003	62.514	15.748	0.000	861.593	1107.408
horsepower	19.0782	0.562	33.972	0.000	17.974	20.182

Omnibus:	11.785	**Durbin-Watson:**	0.933
Prob(Omnibus):	0.003	**Jarque-Bera (JB):**	21.895
Skew:	0.109	**Prob(JB):**	1.76e-05
Kurtosis:	4.137	**Cond. No.**	322.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

`Inferential Analysis`: can I trust my coefficients β ?

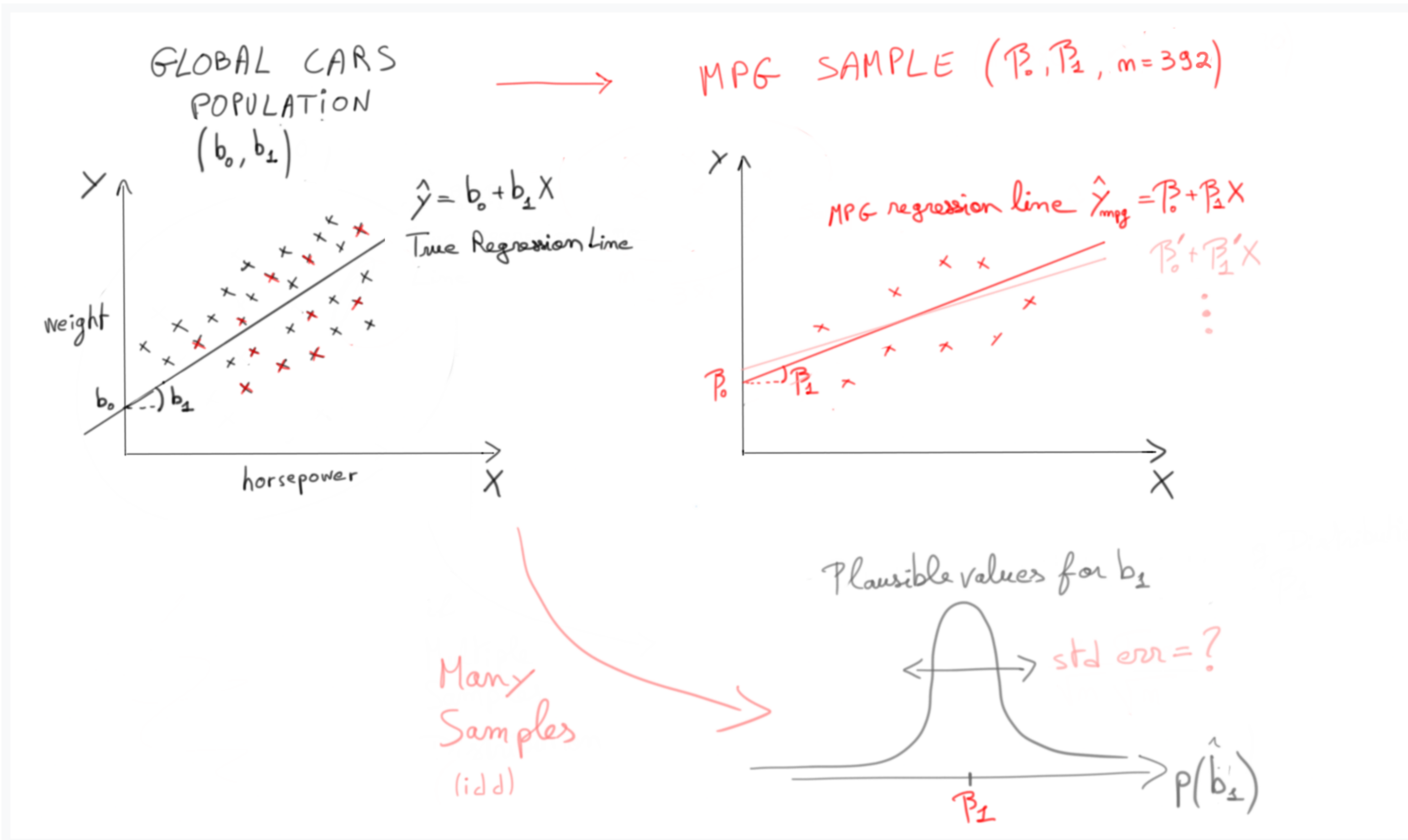
Deterministic Probabilistic

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	984.5003	62.514	15.748	0.000	861.593	1107.408
horsepower	19.0782	0.562	33.972	0.000	17.974	20.182

📖 Under certain conditions (randomness of sampling, etc.):

- The **Distribution of plausible values** for the real b_1 can be **estimated** via the sample `mpg`
- $\hat{b_1}$ = 19.07 [17.9 - 20.1] with 95% confidence interval
- $p(\hat{b_1}$

`std err` on the slope b_1 ?



$$\text{std err}(b_1) = \text{std err}\left(\frac{weights}{horsepower}\right) = \frac{1}{\sqrt{n-2}}\frac{s(residuals)}{s(horsepower)} = 0.562$$

$$\text{std err}(b_1) = \frac{1}{\sqrt{n-2}}\frac{s(residuals)}{s(horsepower)} = \sqrt{\frac{1}{n-2}\frac{\sum(y_i-\hat{y}_i)^2}{\sum(x_i-\bar{x}}} = 0.562)^2$$

In [ ]:
```
# Let's check the formula ourself!
n = 392
residuals = model.predict(mpg['horsepower']) - mpg['weight']
residuals.std() / mpg.horsepower.std() / (n-2)**0.5
```

Out[ ]: `0.5615843732511717`

$p(\hat{b}_1$

💡 Why divide by n-2? We divide by the number of degrees of freedom, which is n-2 in this case. Want to know more about it? Check out this [video (https://www.youtube.com/watch?v=4otEcA3gjLk)](https://www.youtube.com/watch?v=4otEcA3gjLk)

➡️ The **t-statistic**, **p-value** and **95% confidence interval** correspond to the Null Hypothesis:

> H_0
> : In reality, `horsepower` is **not** correlated with `weights` (
> $b_1 = 0$
>)

If
H_0
were true, the observed
β_1
would have a t-score of:

$$t = \frac{\beta_1 - b_1}{\text{std err}(b_1)}$$

`p-value` = "probability that what you observed is just due to pure chance"

= Proba of observing a sample slope of
$\beta_1 = 19.0728$
or bigger...

- **assuming that H_0 is true**
- i.e. *assuming that the real slope* β_1 *was actually 0*
- Since n = 392, we can use a Gaussian Distribution

= Proba of observing
$\beta_1 > 19.0728$
if it was sampled from a distribution
$\mathcal{N}$ (

= Proba of observing t > 33.0927 from a standard distribution
$\mathcal{N}$ (

≈ 0

`p-value` $\approx$ 0 $\leq$ < 0.05

- It is almost **impossible that the feature wouldn't be correlated** with the target variable
- The relationship between `weight` and `horsepower` is **statistically significant**

`F-statistic` = overall statistical significance of the regression

OLS Regression Results

Dep. Variable:	weight	**R-squared:**	0.846
Model:	OLS	**Adj. R-squared:**	0.845
Method:	Least Squares	**F-statistic:**	1067.
Date:	Thu, 09 Jul 2020	**Prob (F-statistic):**	1.19e-158

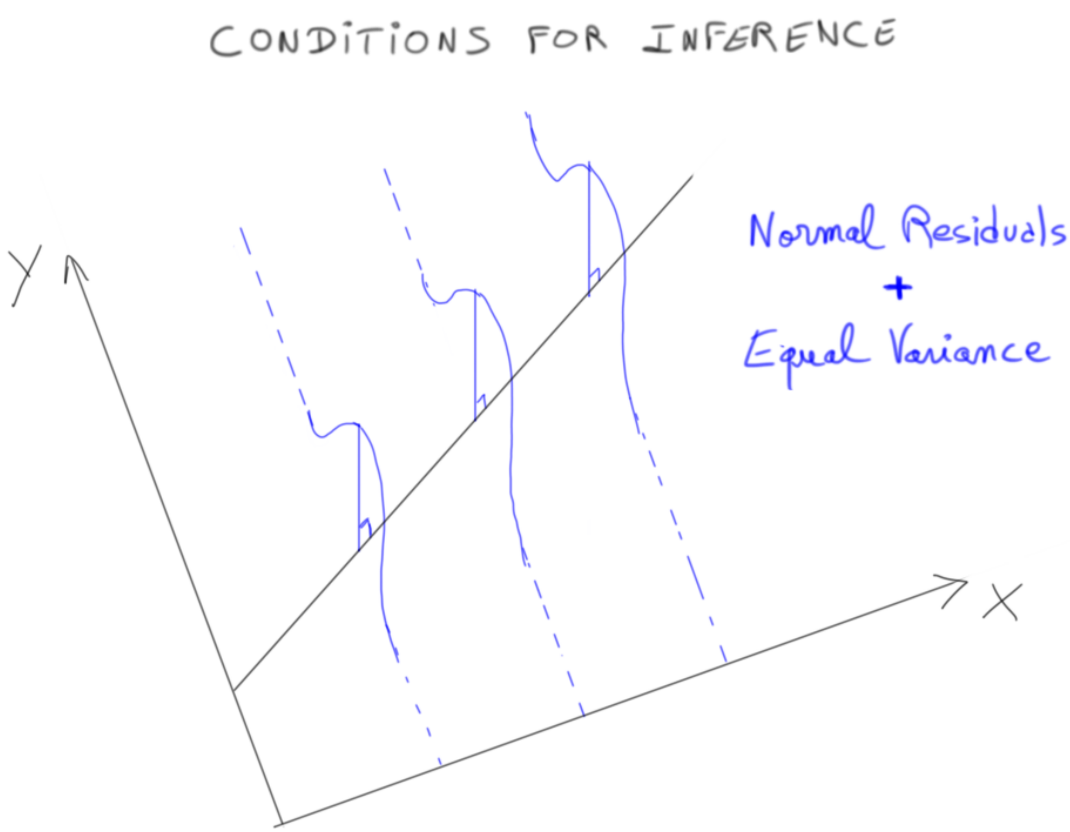
- The F-Statistic represents the combined `p-value` **of all your coefficients**
- It measures the null hypothesis H_0 **: all coefs are null**
- $F \subset [1, \infty]$
- $F \sim 1 \implies H_0$ cannot be ruled out
- $F >> 1 \implies$ at least one coef `p-value` < 0.05
- $F >> 1 \implies$ the regression is statistically significant

4. Checking the assumptions for inferential analysis

✔ Random sampling

✔ Independent sampling (sample with replacement, or n < 10% global pop.)

⚠️ **Residuals normally distributed and of equal variance**



Are residuals normally distributed ?

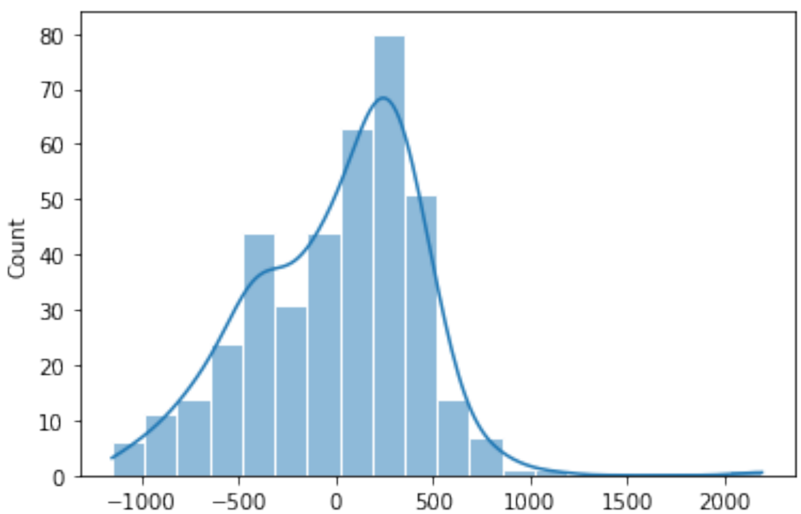
In [ ]:
```
predicted_weights = model.predict(mpg['horsepower'])
predicted_weights
```

Out[ ]:
```
0      3464.661329
1      4132.396983
2      3846.224560
3      3846.224560
4      3655.442944
          ...
393    2625.222220
394    1976.564728
395    2587.065897
396    2491.675089
397    2548.909574
Length: 392, dtype: float64
```

In [ ]:
```
residuals = predicted_weights - mpg['weight']
residuals
# also avaiable via model.resid
```

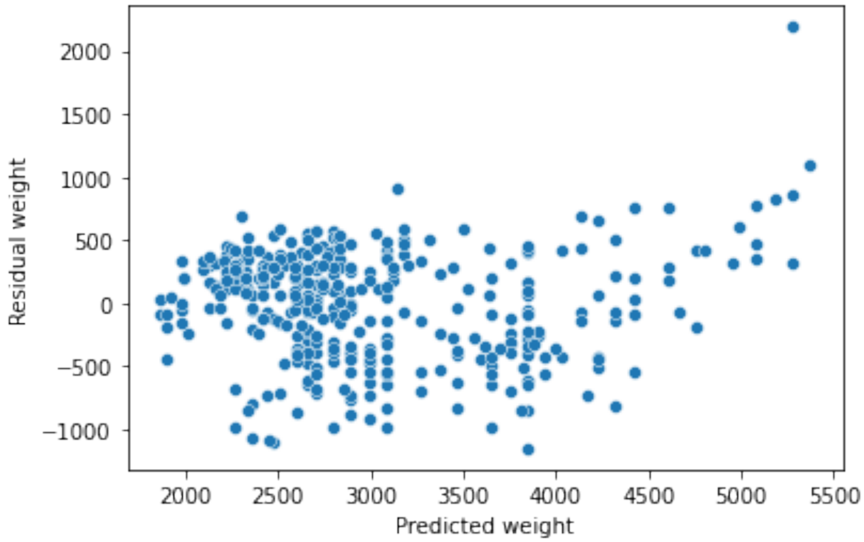
Out[ ]:
```
0      -39.338671
1      439.396983
2      410.224560
3      413.224560
4      206.442944
          ...
393   -164.777780
394   -153.435272
395    292.065897
396   -133.324911
397   -171.090426
Length: 392, dtype: float64
```

In [ ]:
```
# visual check
sns.histplot(residuals, kde=True, edgecolor='w');
```

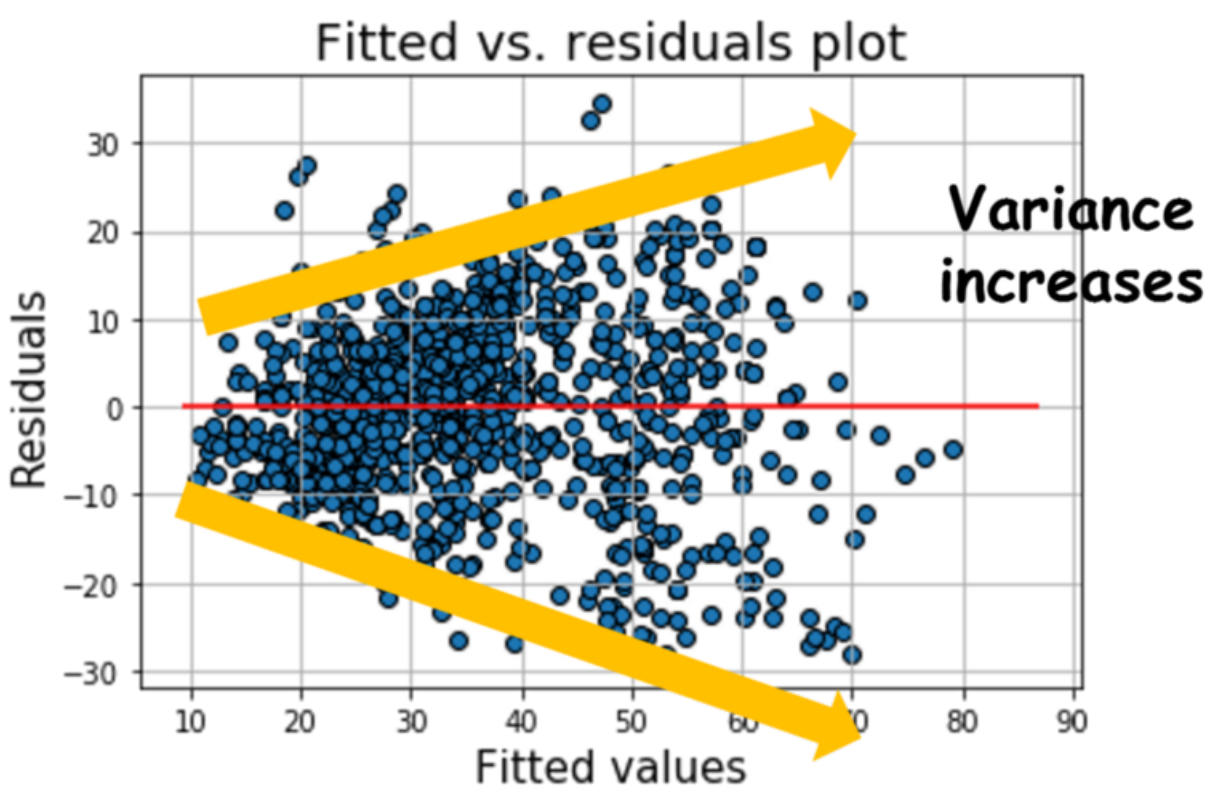


Are residuals of equal variance?

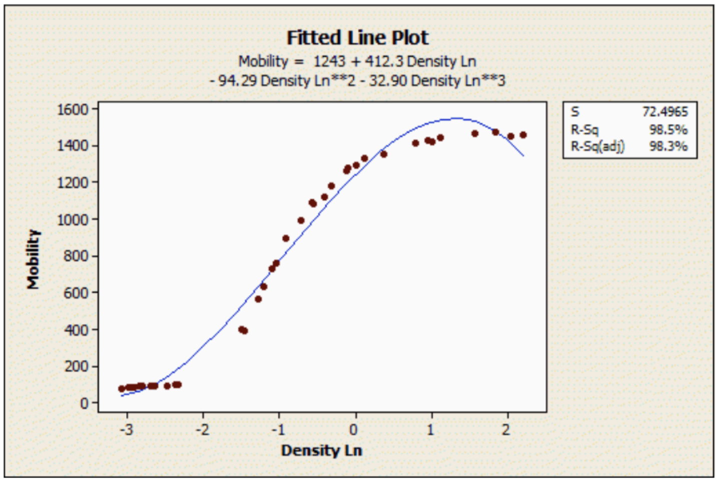
In [ ]:
```python
# Check with Residuals vs. Fitted scatterplot
sns.scatterplot(x=predicted_weights, y=residuals)
plt.xlabel('Predicted weight')
plt.ylabel('Residual weight');
```

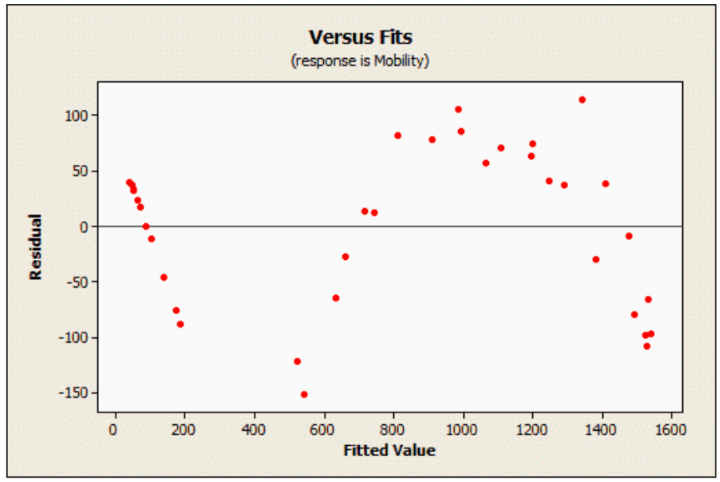


👀 Beware of **heteroscedasticity**



👀 Beware also of **autoregressive residuals**





👉 If a pattern is seen, a factor might be missing in your model!
👉 Frequent issue in Time Series (ex: inflation, weekly patterns etc.)

What if my residuals are really not random?

✅ `R-squared` remains perfecly valid (deterministic coef)

⚠️ However, inferencial coefs cannot be trusted

- `p-values` and `confidence intervals` may be smaller than they should be
- Don't be too confident that your model generalizes well

💡 Fixes?

- Try to create/add new features that explain the residual patterns?
- Try to model a transformed version of Y instead (e.g. log(Y)...)?
- Try other statistical representations than linear ones (next module...)

5. Multivariate Linear Regressions

✏️ Let's run a second OLS model where we regress `weight` on both `horsepower` and `cylinders`

$$weight \sim \beta_0 + \beta_1 horsepower + \beta_2 cylinders$$

In [ ]:
```python
# run OLS model
model2 = smf.ols(formula='weight ~ horsepower + cylinders', data=mpg).fit()
model2.rsquared
```

Out[ ]: 0.8458154043882244

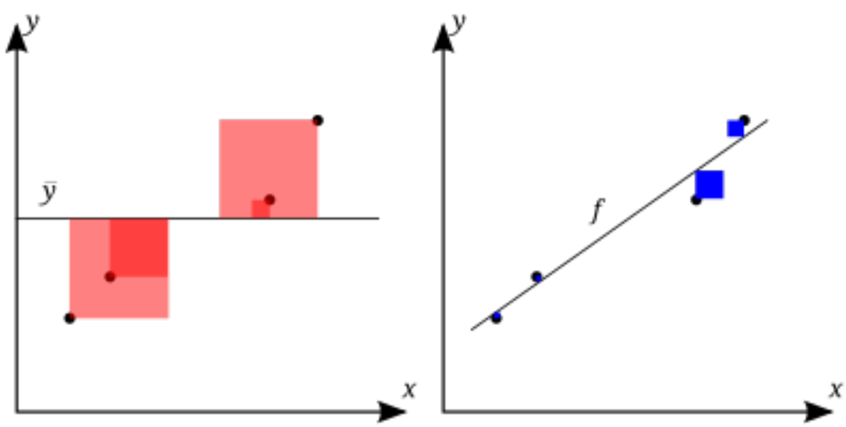
R-squared

> 84% of the variance in the cars' weights can be explained by the combined variations in `horsepower` and `cylinders`

⚠️ In order for the R^2 to be meaningful, the regression must contain an "intercept" (i.e. the matrix X of features must contain a column vector of ones)

⚠️ Contrary to simple linear regression, $R^2 \neq Corr(Y, X_i)^2$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$



❓
R^2
= by how much is my model better than a "simple mean" prediction ❓

🚀
$R^2 = 1$
best case scenario where the target is 100% explained by the features

🤨
$R^2 = 0$
simple mean

😱
$R^2 < 0$
can exist and in this worst case scenario, predicting the mean would be even better than running a Linear Model!

In [ ]:
```
model2.params
```

Out[ ]:
```
Intercept     528.876711
horsepower      8.231070
cylinders     290.356425
dtype: float64
```

> Each increase in `horsepower` increases the `weight` by 8, holding `cylinders` number constant.

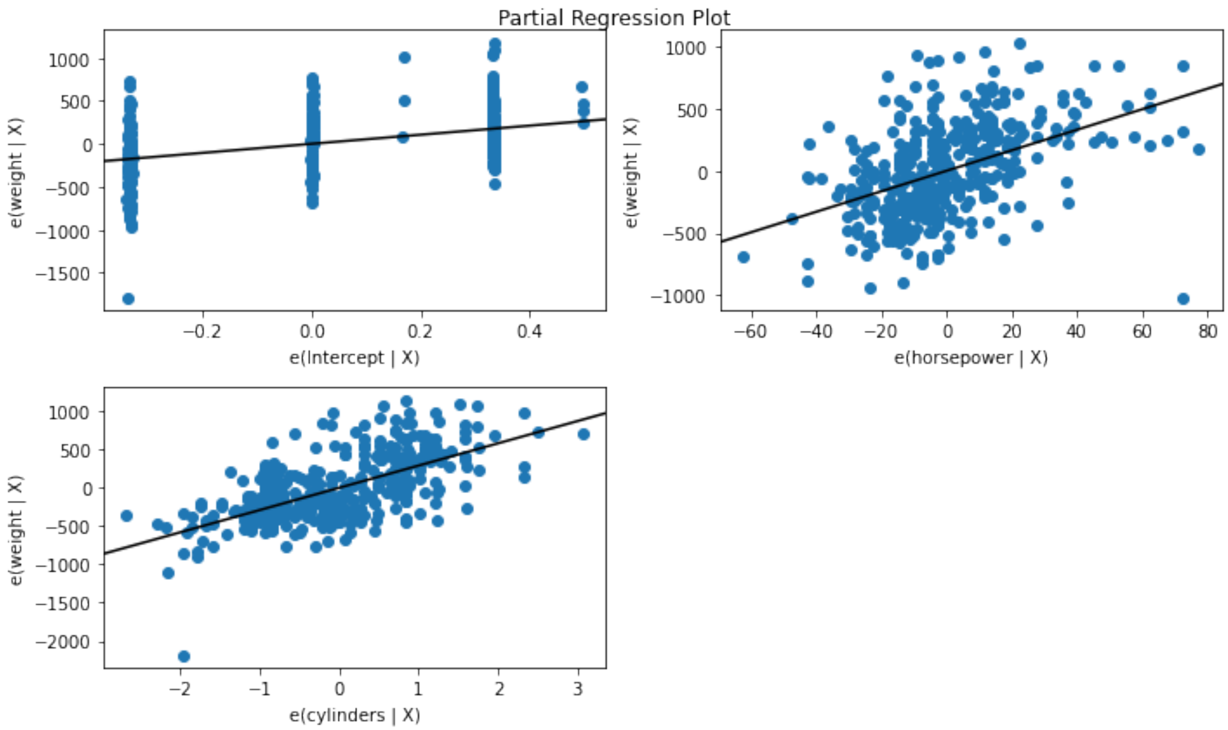
> Controlling for the `cylinders` number, each increase in `horsepower` increases the `weight` by 8 lbs

Partial regression plots

Visualize your multivariate regression coefficients!

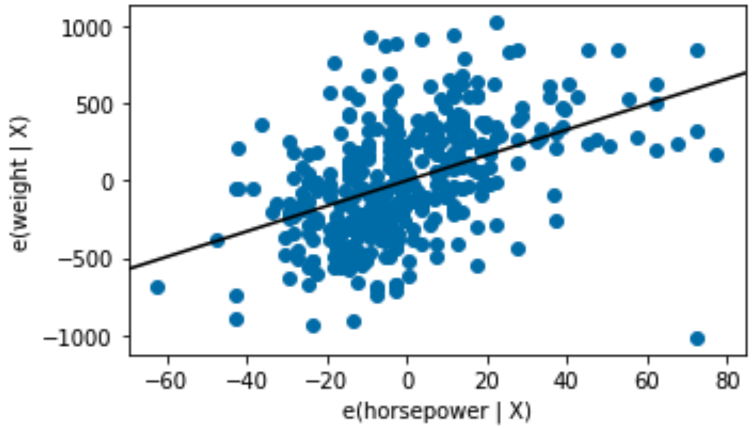
In [ ]:
```
import statsmodels.api as sm
fig = plt.figure(figsize=(10,6))
fig = sm.graphics.plot_partregress_grid(model2, fig=fig)
```

```
eval_env: 1
eval_env: 1
eval_env: 1
```



- Visualize the effect of a particular explaining variable on the dependent variable *while holding all other explaining variable constant.*
- You can see which variables have the strongest influence on **mpg**, after accounting for all other variables, by evaluating the slope of each chart in the grid.

FYI How to construct partial regression plots ?🤯



- Each point is a car in our dataset
  - Y values are the residuals of the predicted `weight` s by using all features except `horsepower` (i.e. using `cylinders`)
  - These residuals contain the remaining information about `weight` that couldn't be explained without `horsepower`
- X value is the residual of predicting `horsepower` by using all other features (i.e. using `cylinders`)
  - These residuals contain the new information that `horsepower` brings to the table, which is not already explained by the other features in the model.

📚 [A good example (https://www.youtube.com/watch?v=Xii1jVLnX60&ab_channel=MikkoR%C3%B6nkk%C3%B6)](https://www.youtube.com/watch?v=Xii1jVLnX60&ab_channel=MikkoR%C3%B6nkk%C3%B6)

Categorical features?

In [ ]:
```
mpg['origin'].unique()
```

Out[ ]:
```
array(['usa', 'japan', 'europe'], dtype=object)
```

In [ ]:
```
# Use C(variable) in the formula
model3 = smf.ols(formula='weight ~ C(origin)', data=mpg).fit()
model3.params
```

Out[ ]:
```
Intercept              2433.470588
C(origin)[T.japan]     -212.242740
C(origin)[T.usa]        939.019208
dtype: float64
```

> A car made in Japan is on average 212 lbs lighter than a European one

- When passing a categorical variable, *statsmodels* uses the first variable as the reference.
- The intercept is equal to the mean of the reference (here `origin==europe`)
- Each coefficient corresponds to the difference with the mean of the reference

In [ ]:
```
mpg.groupby('origin').agg({'weight':'mean'})
```

Out[ ]:

	weight
origin	
europe	2433.470588
japan	2221.227848
usa	3372.489796

In [ ]:
```
# Drop the intercept if you want to
model3 = smf.ols(formula='weight ~ C(origin) -1', data=mpg).fit()
model3.params
```

Out[ ]:
```
C(origin)[europe]    2433.470588
C(origin)[japan]     2221.227848
C(origin)[usa]       3372.489796
dtype: float64
```

Regression Diagnostic Cheat Sheet

Check	Description	Diagnosis
Goodness-of-fit	The model explains a good deal of the observed variance of the dependent variable	R-square
Statistical significance	Can we trust the regression coefficients of the model - do they generalize?	p-values and F-statistic
Inference conditions	Random Residuals: zero-mean, constant variance, not correlated	Residual plots

6. (Appendix) Mathematical Solution for OLS

We want to model
Y
(**dependent variable, or target**) by a linear combination of multiple
X_i
(**independent variables, or features**)

$\vec{Y}$

$\vec{Y}$

$$\begin{bmatrix} y_1 \\ y_2 \\ \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1k} \\ 1 & x_{21} & x_{22} & \dots & x_{2k} \\ \\ 1 & x_{n1} & x_{n2} & \dots & x_{nk} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \\ \beta_k \end{bmatrix} + \begin{bmatrix} u_1 \\ u_2 \\ \\ u_n \end{bmatrix}$$

Ordinary Least Squares (OLS) finds the
β
that minimizes the Euclidian norm of the residuals

Given the definition of the Euclidian norm

| |

with
u'
= transposed(
u
)

$$u'u = (Y' - \beta'$$

Minimized when derivative equals 0

$$\frac{\partial}{\partial \beta}(Y'Y - 2\beta'$$

💡 Intuitively similar to 1-D derivative formula

$$-2X'Y + 2X'X\beta$$

- $(X'X)^{-1}X'$ is called the "pseudo-inverse" of X
  - Exists only if $(X'X)$ is inversible
- Requires all features of X to be **independent** i.e **non-multicollinear**!
- i.e. X is full-rank matrix ($rank(X)$ = number of features)

The **rank** of a matrix is the **dimension of the vector space** formed by its columns

In [ ]:
```
X = np.array([
    [1., 0., 1.],
    [0., 1., 1.],
    [0., 0., 0.]
])

np.linalg.matrix_rank(X)
```

Out[ ]: 2

☹️
$\implies$
No single solution to OLS if two features are multicollinear

☹️
$\implies$
Can't trust regression coefficients if the features are multicollinear

Computational Complexity?

- Inverting $X'X$ with a basic techinque is of $O(k^3)$ complexity
- Inverting $X'X$ with an advanced technique is of $O(k^{2.4})$ complexity
- Computing the pseudo-inverse $(X'X)^{-1}X'$ directly using SVD decomposition reduces the complexity to $O(k^2)$

👉 Not great for numerous features k

👉 Great for numerous observation n as it scales proportionally with $O(n)$

Bibliography 📚

- OLS Inference Assumption by KDNuggets (https://www.kdnuggets.com/2019/07/check-quality-regression-model-python.html) and Stats By Jim (https://statisticsbyjim.com/regression/ols-linear-regression-assumptions/)
- StatsQuest - Linear Regression (https://www.youtube.com/playlist?list=PLblh5JKOoLUIzaEkCLIUxQFjPIlapw8nU) (1h Youtube intuitive video)
- Statistics by Jim - Regression Analysis (https://statisticsbyjim.com/) (concise blog/book, intuitive)
- Coursera, Regression Models (https://www.coursera.org/learn/regression-models) (4-week free video classroom)
- G.James / D. Witten / T. Hastie / R. Tibshirani - An Introduction to Statistical Learning - Section 3 regression (https://www.statlearning.com) (BSc level maths)

🚀 Your turn!

- Challenge 01: Model `review_score` of `orders` as a linear function of multiple explaining variables
- Challenge 02: Analyze which `sellers` (and `products`) are repetitively under-performers