Logistic Regression

Plan

1. Primers & Problem Statement
2. Logistic Regressions
3. Interpretation
4. Performance Evaluation
5. Multicollinearity issues in Linear Models (Lin/Log)

[Lecture notebook (https://github.com/lewagon/data-lecture-starters/blob/main/starters/04-Decision-Science_04-Logistic-Regression.ipynb)](https://github.com/lewagon/data-lecture-starters/blob/main/starters/04-Decision-Science_04-Logistic-Regression.ipynb)

1. Primers

$\log_2(8) = 3$

$\log_2(\frac{1}{2}) = -1$

$\ln(5) = \log_e(5) = 1.6$

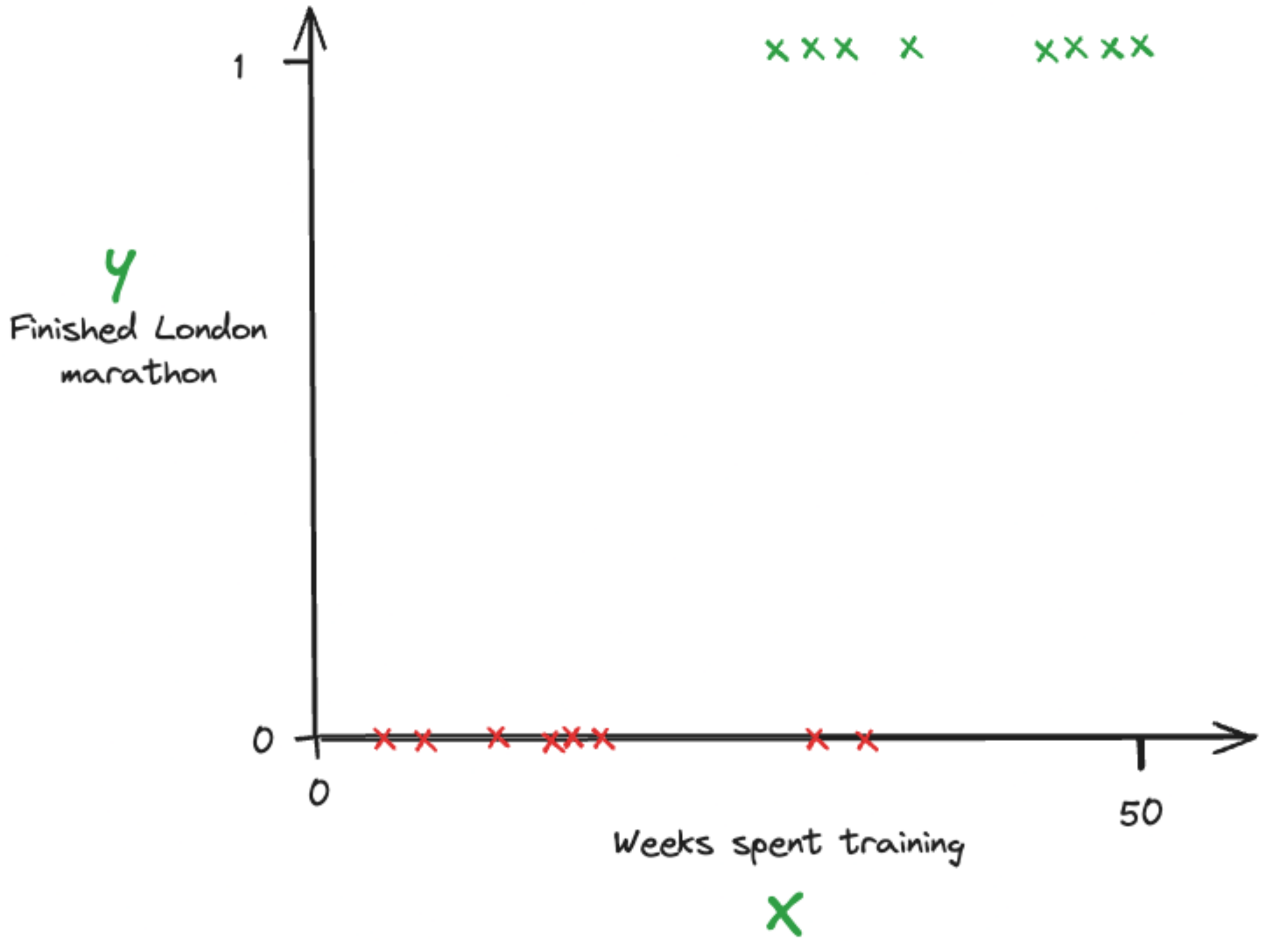
$e^0 = 2^0 = 1$

A probability of 0.9 (a.k.a 90%) can be represented as:
$P(\text{Event}) = 0.9$

This means that out of 100 occurrences, the event is expected to happen 90 times.

The problem: How can we predict a binary outcome?

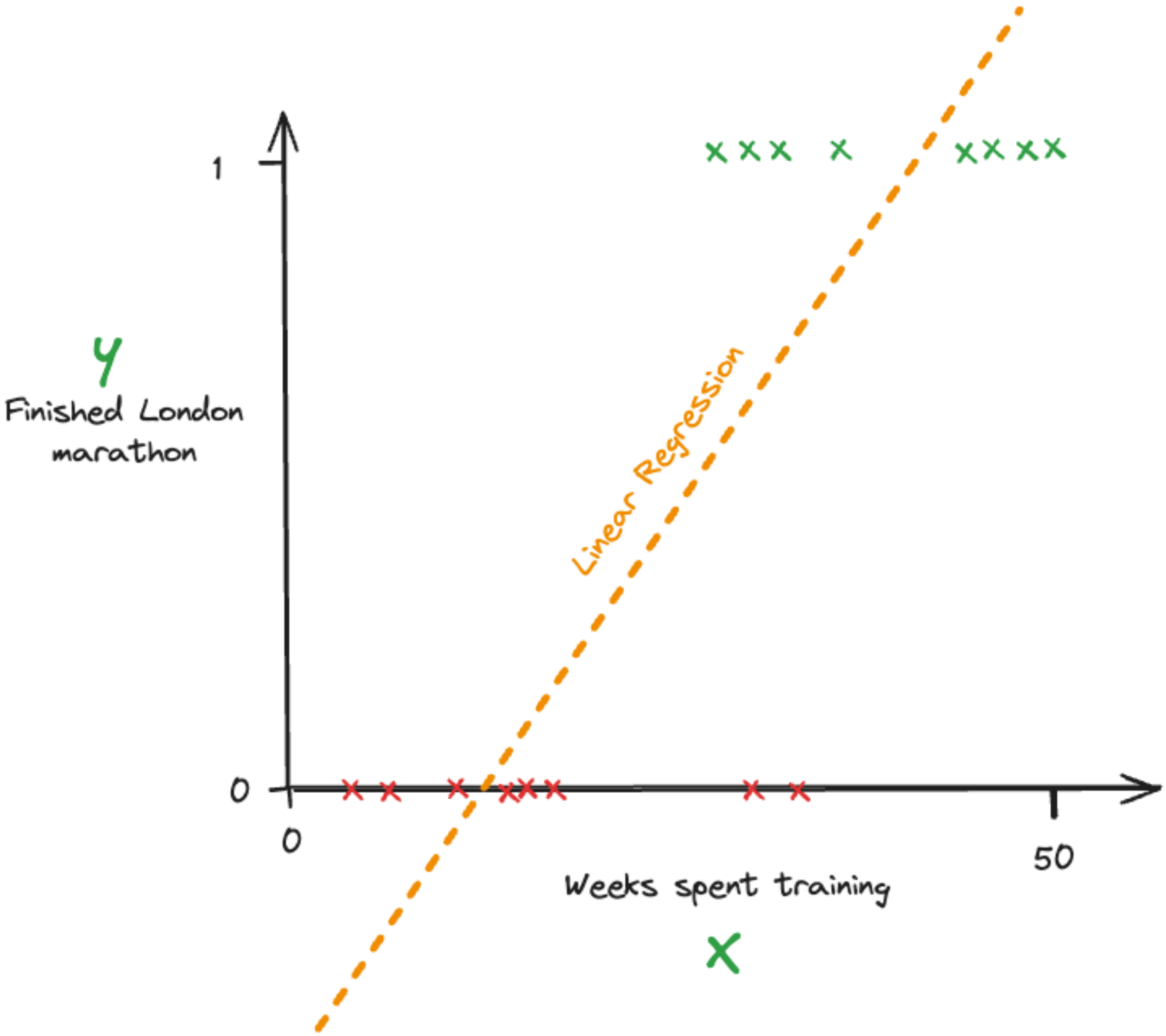
- Win / Loss ?
- Success / Failure
- dim_is_one_star or not?



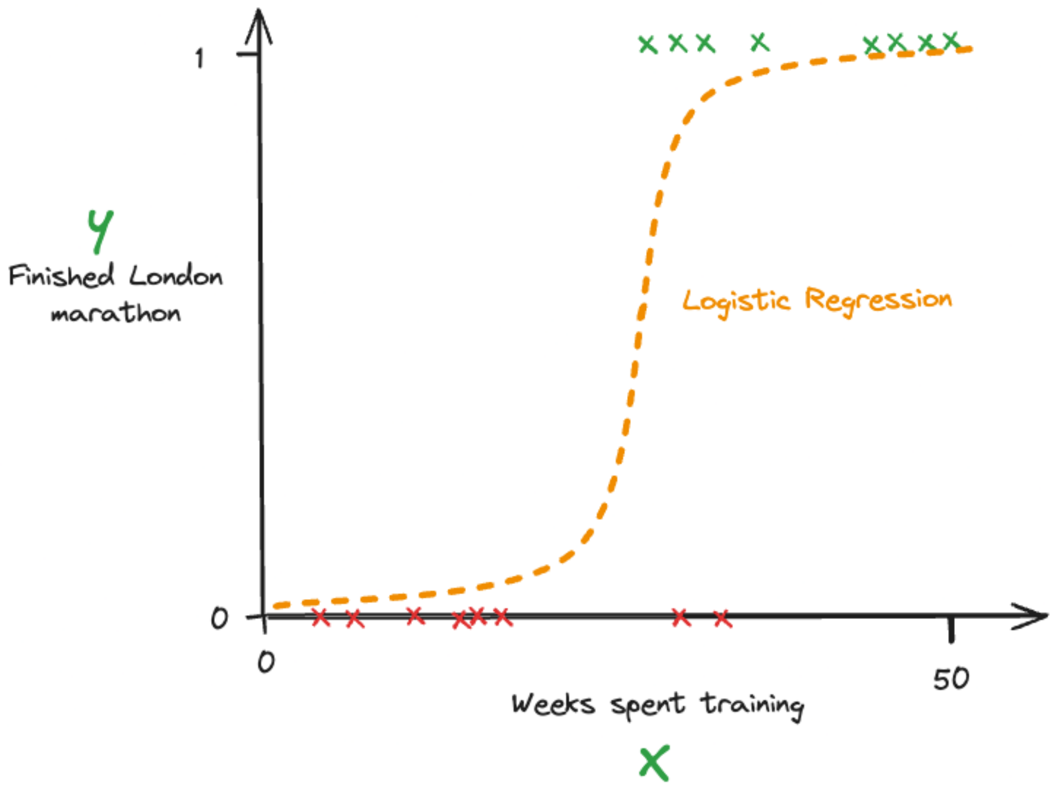
Instead of fitting the best `straight line`

$linreg(x) = \beta_0 + \beta_1 X$

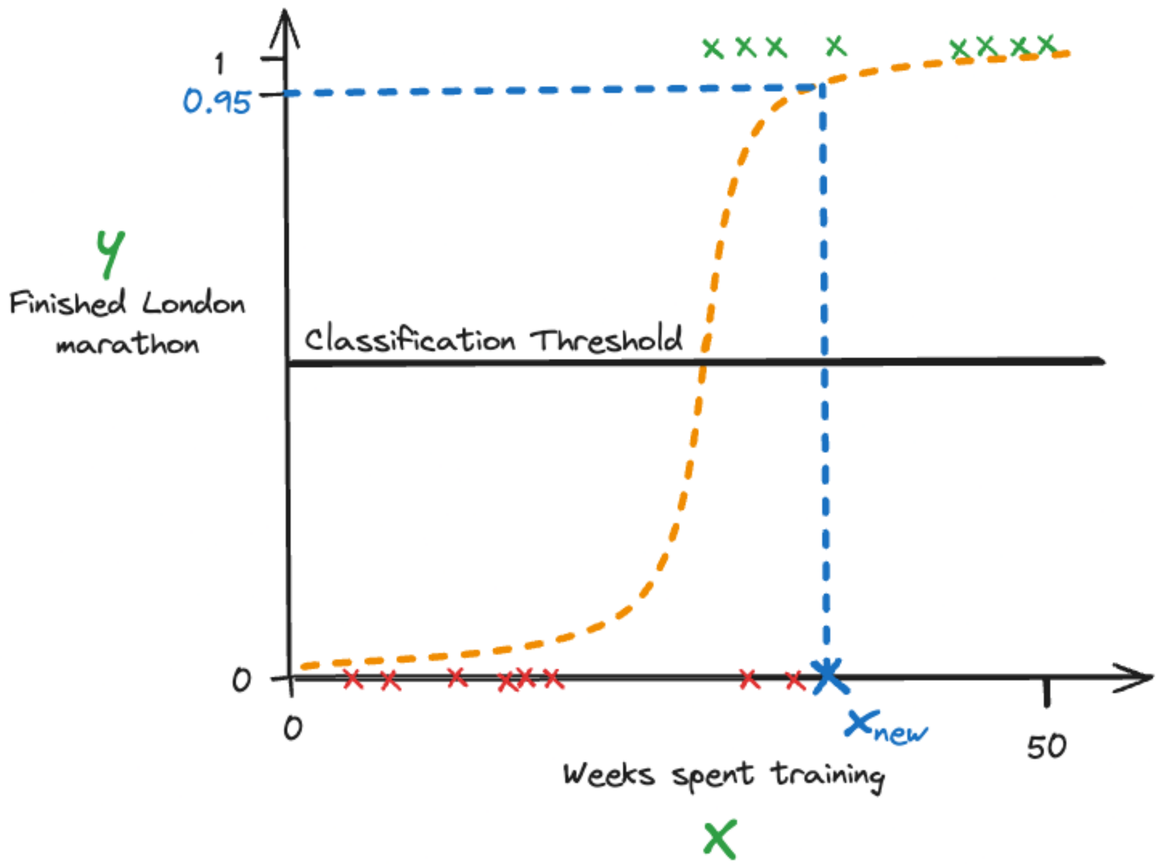
...



We will try to fit the best **`sigmoid function`**
$\hat{y}$



Then, we set a `classification threshold` (usually at 50% by default)
🔴 observations predicted with
$\hat{y}$
are classified as 1's
🔵 observations predicted with
$\hat{y}$
are classified as 0's



2. Logistic Regressions

The Barcelona Women's Football Team is going on tour

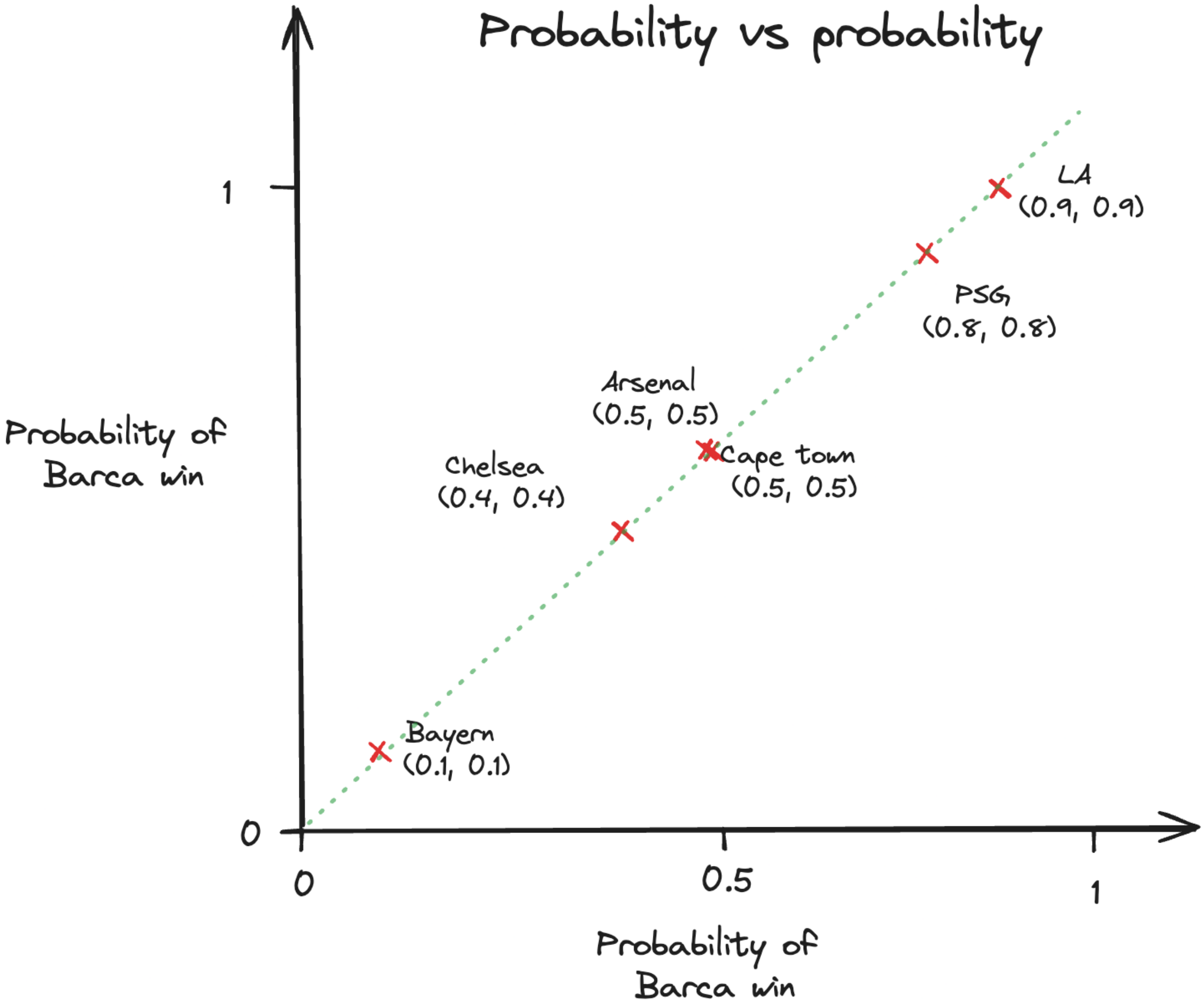
- Every game is a win-lose scenario (they play penalties to avoid draws!)
- We are given the opportunity to bet on the team **before** the tour happens

Probability, Odds and Log-Odds 💪

Imagine we go to some gambling website...

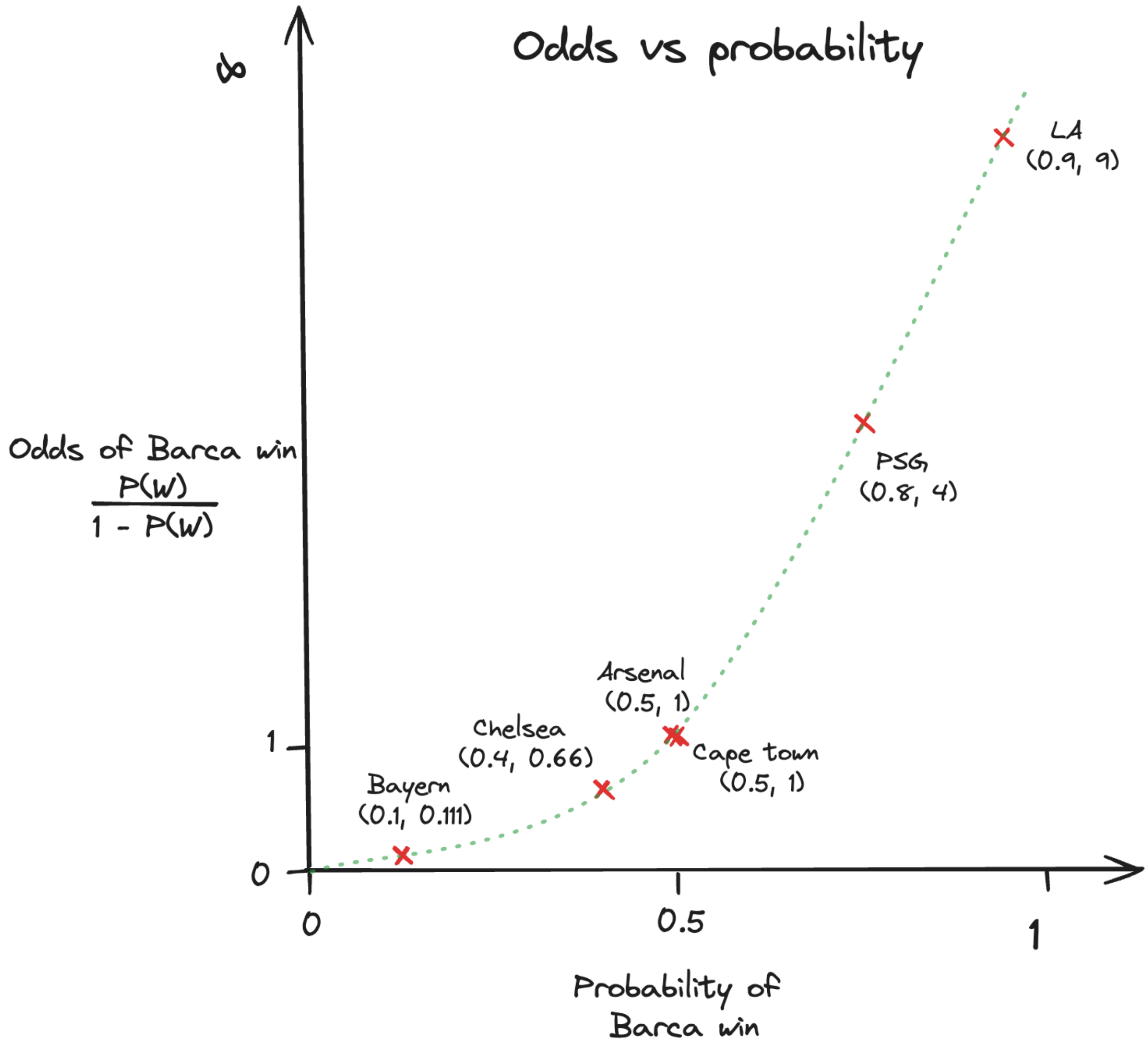
They show us what they think the probability of Barca winning will be:

Opponent	Probability of Barca winning: $P(x)$
Arsenal	0.5
Cape Town	0.5
Chelsea	0.4
Bayern Munich	0.1
PSG	0.8
Los Angeles	0.9



They also show us the odds!

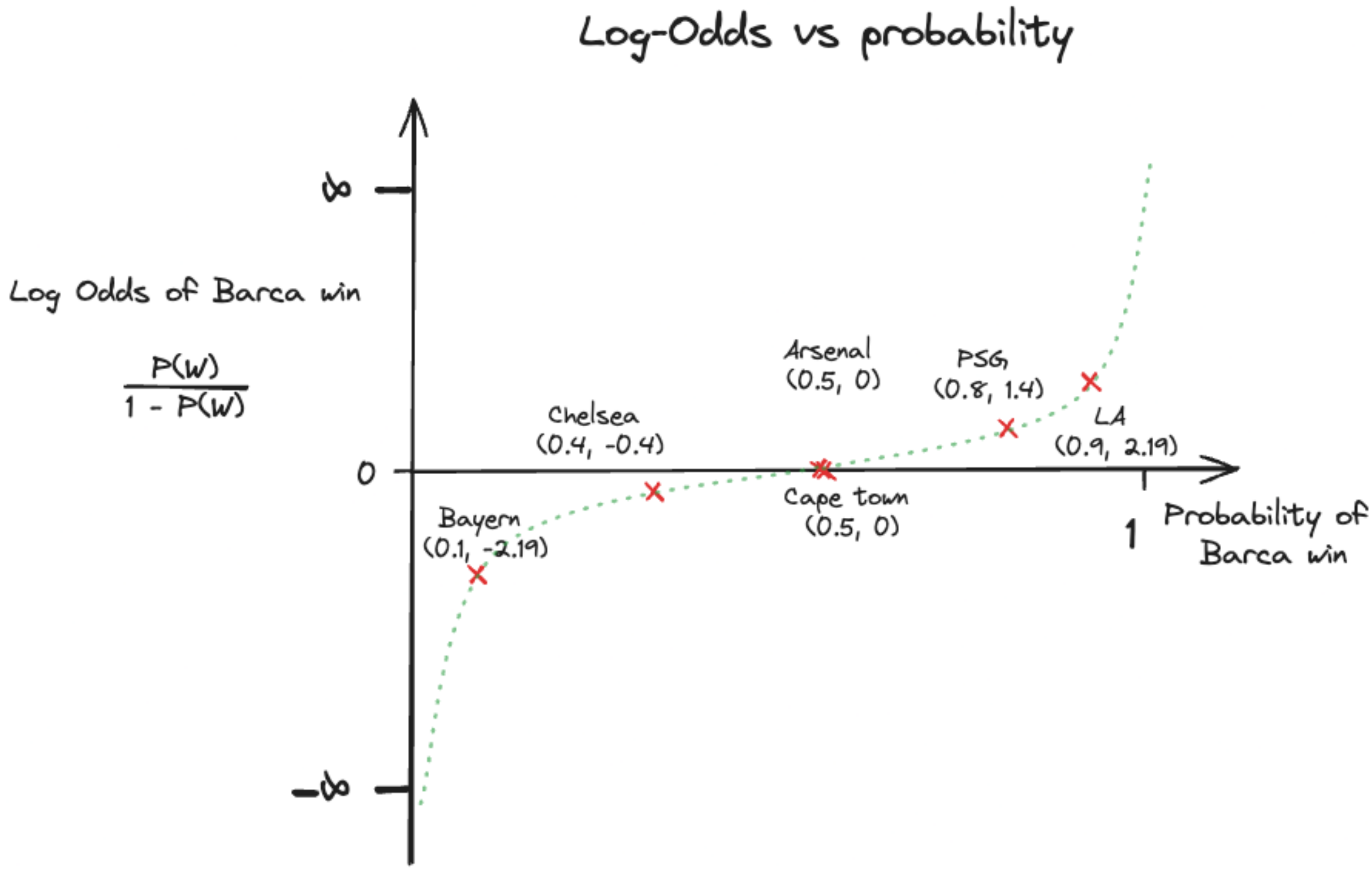
Opponent	Probability of Barca winning: $P(x)$	Odds of winning: $\frac{P(x)}{1 - P(x)}$
Arsenal	0.5	$\frac{0.5}{0.5} = 1$ ("1:1 odds")
Cape Town	0.5	$\frac{0.5}{0.5} = 1$
Chelsea	0.4	$\frac{0.4}{0.6} = \frac{2}{3} = 0.666$
Bayern Munich	0.1	$\frac{0.1}{0.9} = \frac{1}{9} = 0.111$
PSG	0.8	$\frac{0.8}{0.2} = \frac{4}{1} = 4$
Los Angeles	0.9	$\frac{0.9}{0.1} = \frac{9}{1} = 9$



And the **log** of the odds 🤔

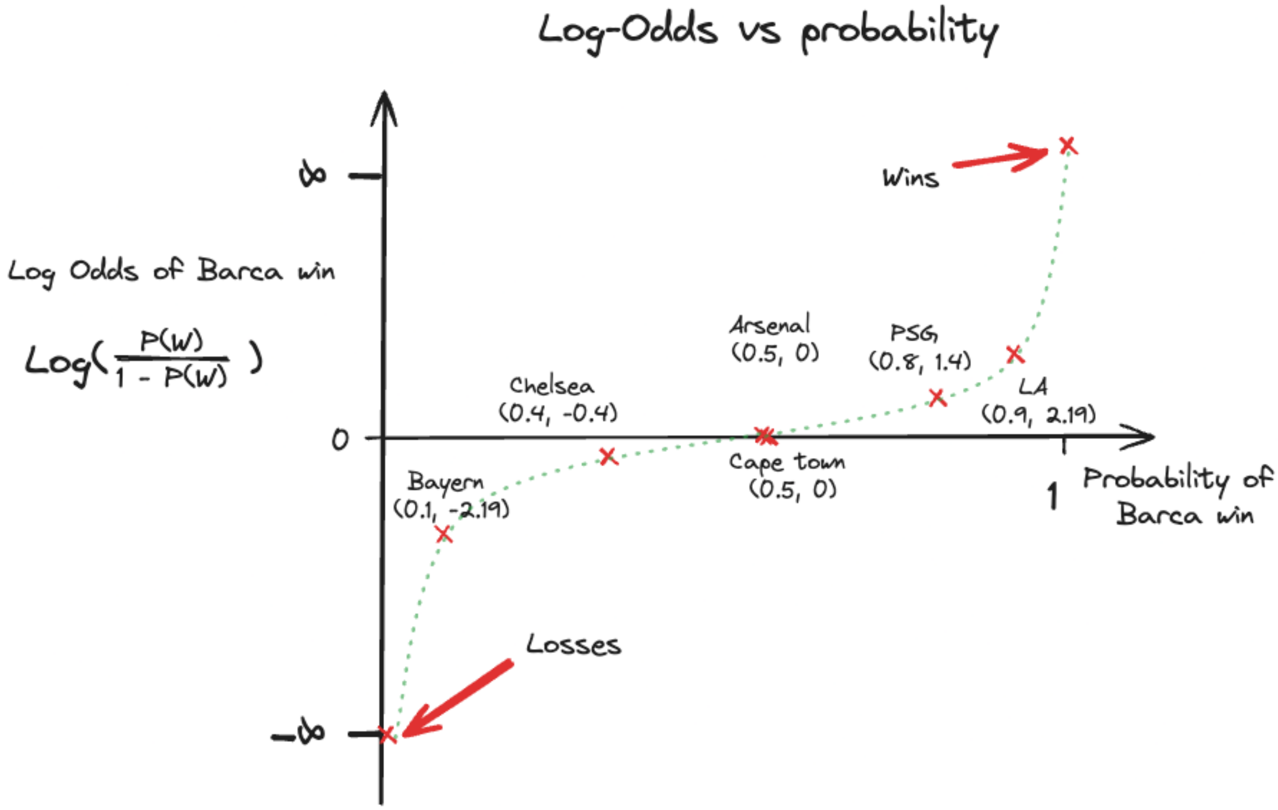
Opponent	Probability of Barca winning: $P(x)$	Odds of winning: $\frac{P(x)}{1 - P(x)}$	Log odds of winning: $\log\left(\frac{P(x)}{1 - P(x)}\right)$
Arsenal	0.5	$\frac{0.5}{0.5} = 1$ ("1:1 odds")	$\log(1) = 0$
Cape Town	0.5	$\frac{0.5}{0.5} = 1$	$\log(1) = 0$
Chelsea	0.4	$\frac{0.4}{0.6} = \frac{2}{3} = 0.666$	$\log(0.666) = -0.4$
Bayern Munich	0.1	$\frac{0.1}{0.9} = \frac{1}{9} = 0.111$	$\log(0.111) = -2.19$
PSG	0.8	$\frac{0.8}{0.2} = \frac{4}{1} = 4$	$\log(4) = 1.4$
Los Angeles	0.9	$\frac{0.9}{0.1} = \frac{9}{1} = 9$	$\log(9) = 2.19$

👉The `Logit` function maps a probability
$p \in [0, 1]$
to its log-odds
$\in [-\infty, +\infty]$

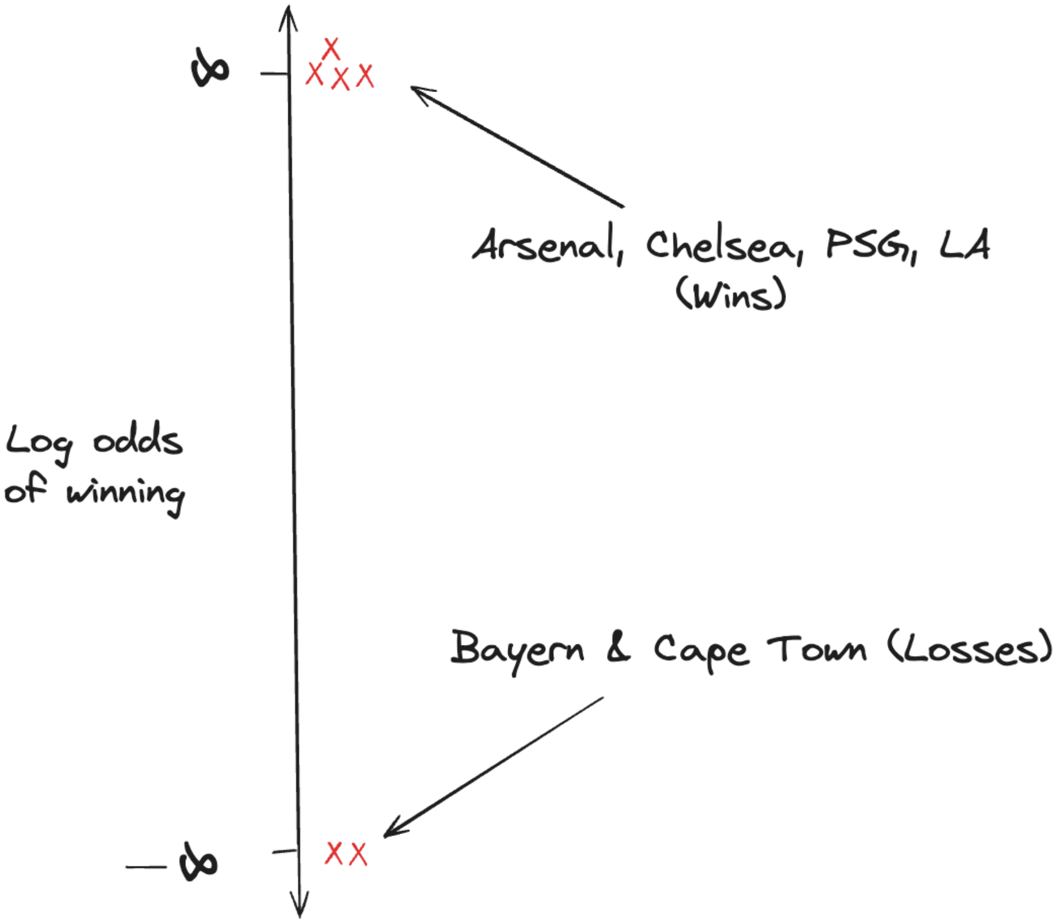


The games are played and the results are in!

Opponent	Probability of winning: $P(x)$	Odds of winning: $\frac{P(x)}{1 - P(x)}$	Log odds of winning: $\log\left(\frac{P(x)}{1 - P(x)}\right)$
Arsenal	1	$\frac{1}{0}$	for practical/ visualization purposes we treat this as ∞
Cape Town	0	$\frac{0}{1}$	we'll treat this as $-\infty$
Chelsea	1	$\frac{1}{0}$	∞
Bayern Munich	0	$\frac{0}{1}$	$-\infty$
PSG	1	$\frac{1}{0}$	∞
Los Angeles	1	$\frac{1}{0}$	∞



Log-Odds vs probability
∞
Log Odds of Barca win
Log(P(W) / 1 - P(W))
0
-∞
Wins
Losses
Arsenal (0.5, 0)
PSG (0.8, 1.4)
Chelsea (0.4, -0.4)
LA (0.9, 2.19)
Cape town (0.5, 0)
Bayern (0.1, -2.19)
1
Probability of Barca win



We hear news that another game will be played!

We've lost some money and we want to build a model so we can predict what the outcome will be. But **can we**?

Not really! All we know is that Barca has played 6 games: they won 4 and lost 2.

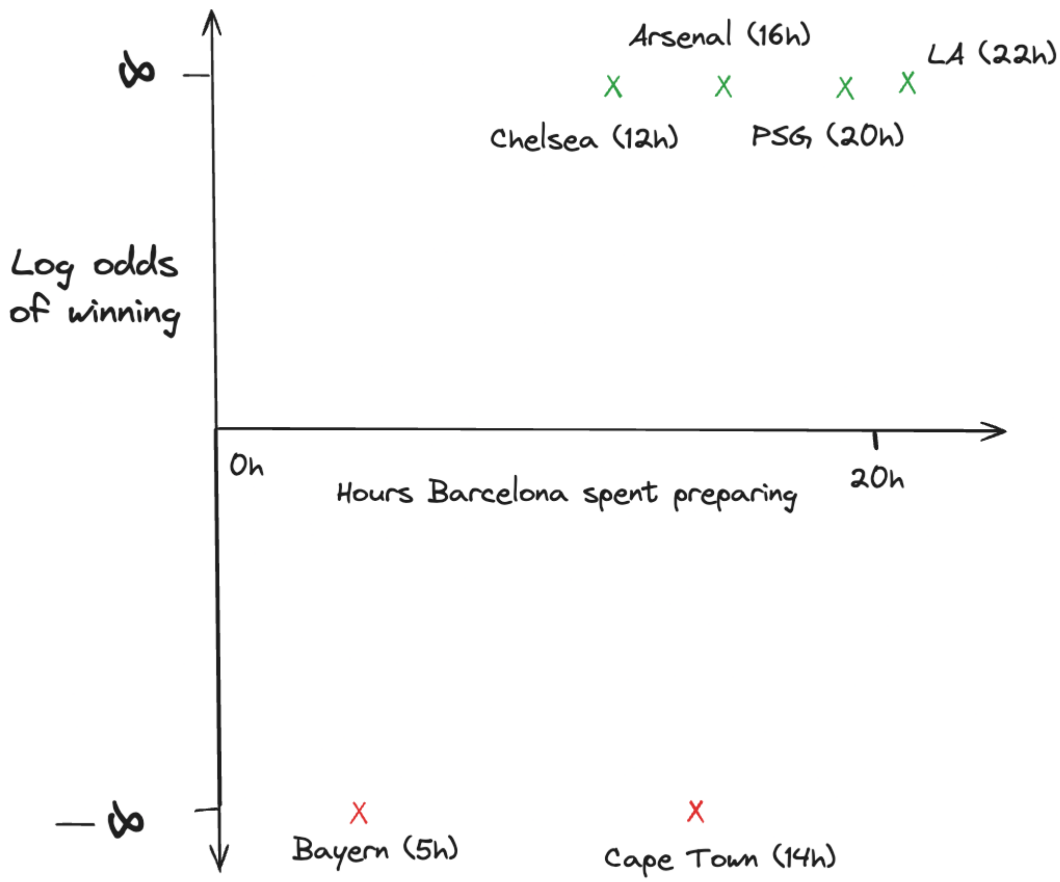
We have no explanatory variables (X) to determine our outcome (y)

So the naive method we have to work with is that Barca win 4/6 times so
$P(\text{Barcelona winning}) = 0.666$

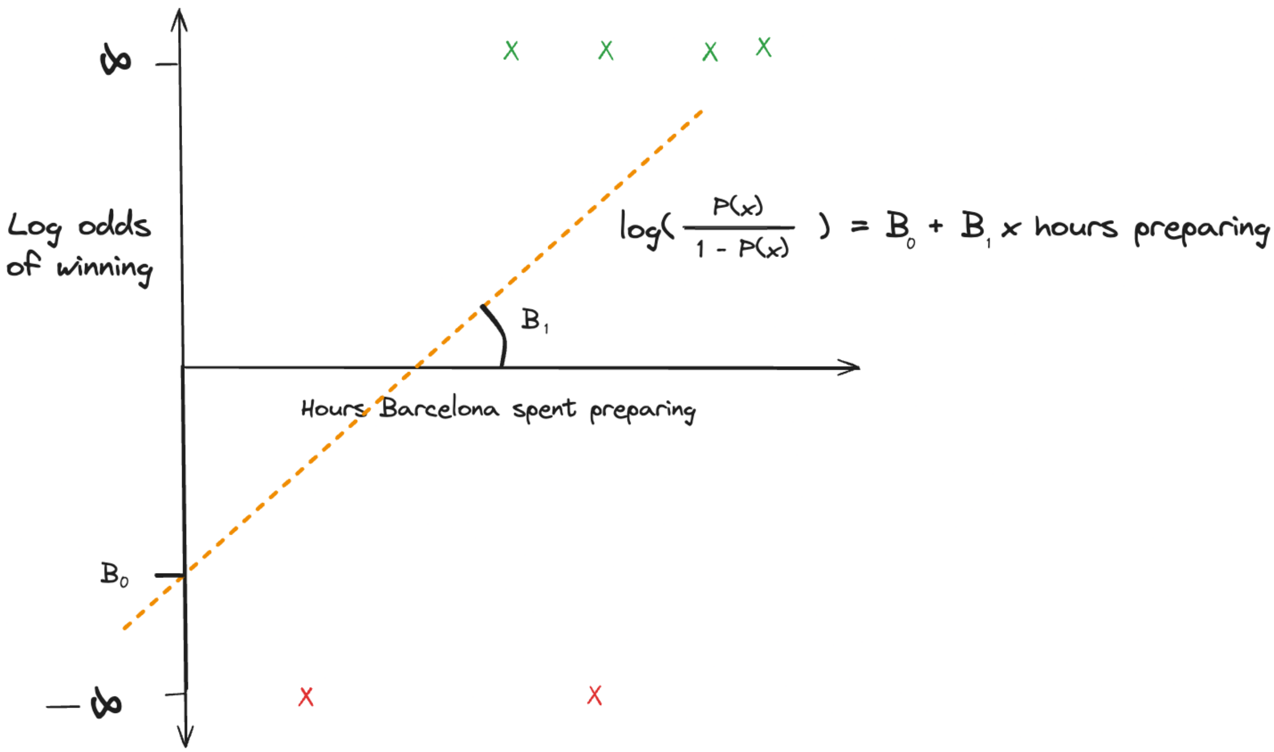
A friend comes to us with some insider knowledge 👀

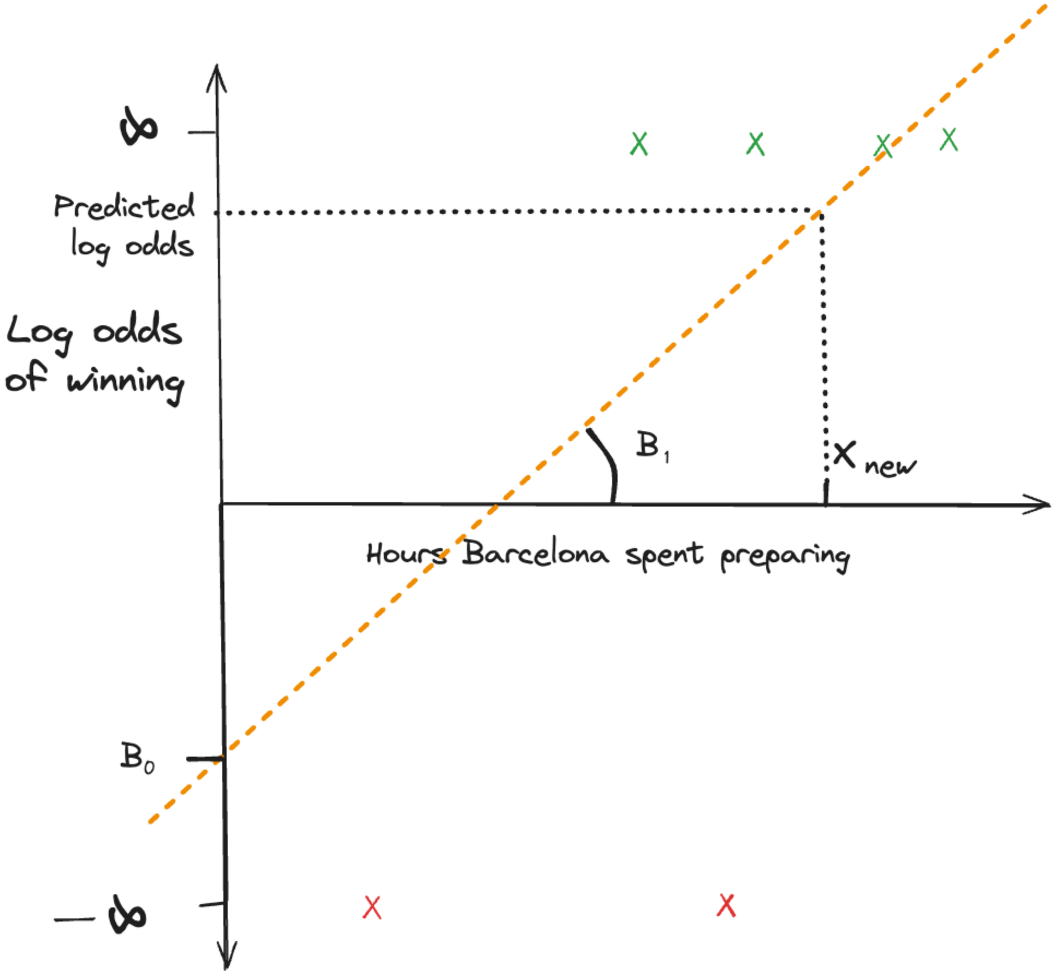
	y		x
Opponent	Probability of winning: $P(x)$	Log odds of winning: $\log\left(\frac{P(x)}{1 - P(x)}\right)$	Hours Barcelona spent preparing for match
Arsenal	1	∞	16
Cape Town	0	$-\infty$	14
Chelsea	1	∞	12
Bayern Munich	0	$-\infty$	5
PSG	1	∞	20
Los Angeles	1	∞	22

Let's visualize things again



Create a first try at "best fit" line


∞
Log odds
of winning
$\log\left(\frac{P(x)}{1 - P(x)}\right) = B_0 + B_1 \times \text{hours preparing}$
B_1
Hours Barcelona spent preparing
B_0
$-\infty$



$$\log\left(\frac{P(x)}{1 - P(x)}\right) = B_0 + B_1 \times \text{hours preparing}$$

Dummy values

Example new data point

$$\log\left(\frac{P(x)}{1 - P(x)}\right) = 5 + 2 \times 15 \text{ hours preparing}$$

$$= 35$$

But I thought we were interested in probabilities! What happened to that nice S-shaped curve? 🤔

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X$$

Exponentiate

$$\frac{p}{1-p} = e^{\beta_0+\beta_1 X}$$

Flip the fractions

$$\frac{1-p}{p} = \frac{1}{e^{\beta_0+\beta_1 X}} = e^{-(\beta_0+\beta_1 X)}$$

Multiply by p

$$1 - p = pe^{-(\beta_0+\beta_1 X)}$$

Add the p across

$$1 = p + pe^{-(\beta_0+\beta_1 X)}$$

Factorize the p's

$$1 = p\boxed{(1 + e^{-(\beta_0+\beta_1 X)})}$$

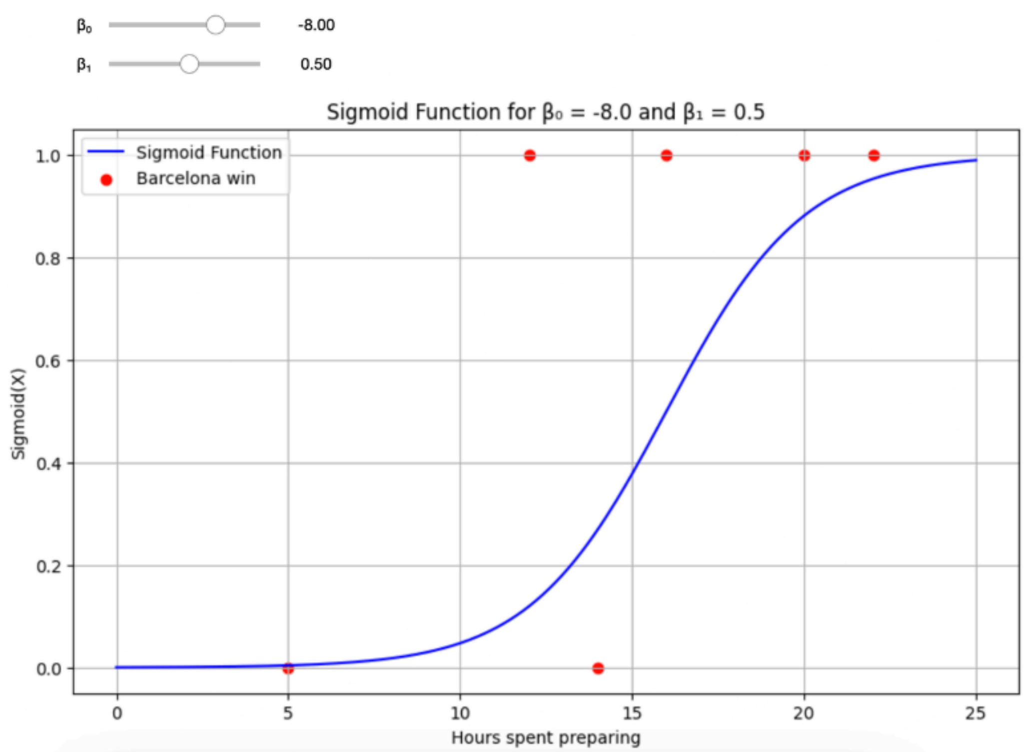
Divide by this

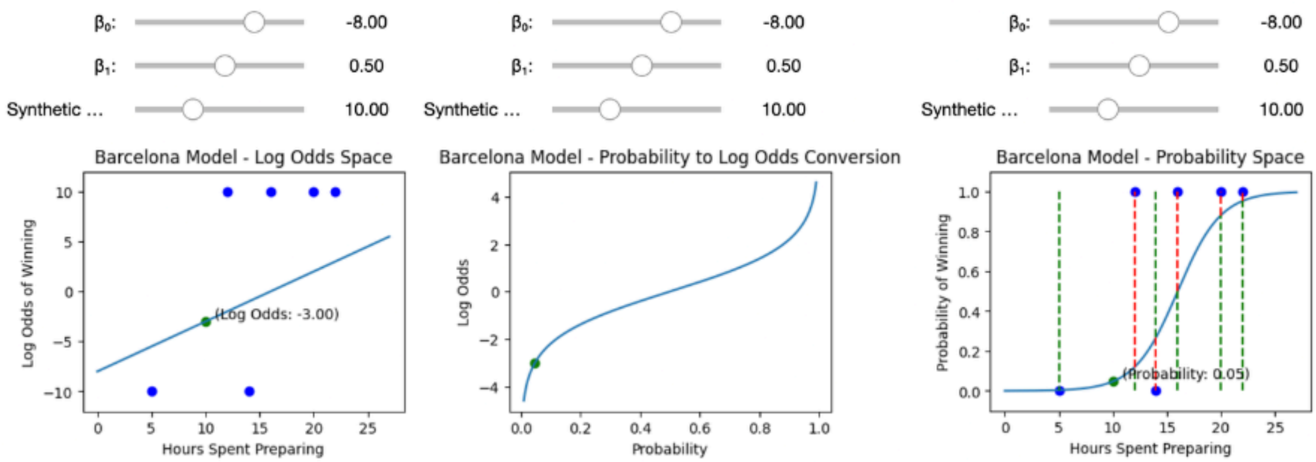
$$p = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}}$$

$$\boxed{p = \sigma(\beta_0 + \beta_1 X)}$$

This is what we refer to as the Sigmoid function!

Let's take a look






```
Model's Likelihood: 0.0364093130148228
```

But how should we decide on the best coefficients?

Our results vector is represented as:
$y = [1, 0, 1, 0, 1, 1]$

We **want** our model's predictions to be close to this:

True value	1	0	1	0	1	1
Predicted probability	0.99	0.02	0.97	0.04	0.91	0.89

How do we tweak our betas to optimize for this?

- Minimize some distance in the log odds space? ❌
- Minimize MSE in the probability space? ❌
- Minimize MAE in the probability space? ❌

👀 Consider rather the following **product**:

True value	1	0	1	0	1	1
Predicted probability	0.99	0.02	0.97	0.04	0.91	0.89
The product	0.99 ×	1 - 0.02 ×	0.97 ×	1 - 0.04 ×	0.91 ×	0.89
	0.99 ×	0.98 ×	0.97 ×	0.96 ×	0.91 ×	0.89

- It is always between 0 and 1
- The closer to 1, the better 🎉

This is *precisely* the **combined probability of observing all the independent**
y_i
outcomes, if they were drawn from Bernoulli distributions of parameters
$p = \hat{y}_i$

(👉 [A visual explanation (https://github.com/lewagon/data-images/blob/master/decision-science/likelihood-logisitc.png?raw=true)](https://github.com/lewagon/data-images/blob/master/decision-science/likelihood-logisitc.png?raw=true))

✅ This is called the **Likelihood**

$L(\beta) = 0.99 \times (1 - 0.02$

> Likelihood of observing
> y
> , given the predicted probabilities
> $\hat{y}$
>
> "We want to choose β so as to make the data as probable as possible"

Likelihood with correct predictions

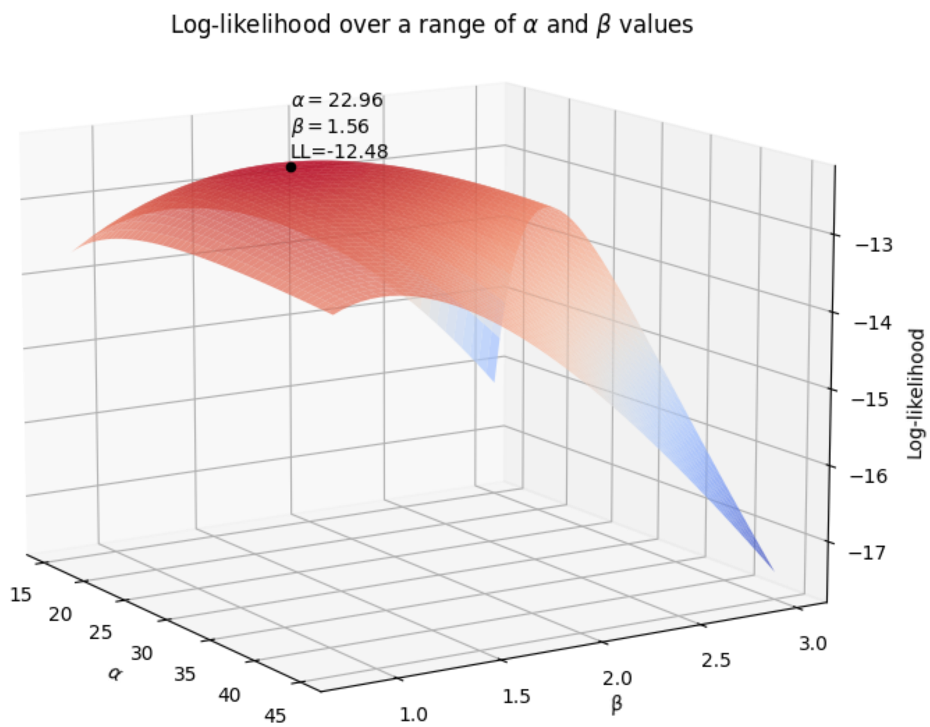
True value	1	0	1	0	1	1	
Predicted probability	0.99	0.02	0.97	0.04	0.91	0.89	
Prediction after threshold	1	0	1	0	1	1	
Likelihood	0.99 ×	1 - 0.02 ×	0.97 ×	1 - 0.04 ×	0.91 ×	0.89	
	0.99 ×	0.98 ×	0.97 ×	0.96 ×	0.91 ×	0.89	= 0.73

Likelihood with wrong predictions

True value	1	0	1	0	1	1	
Predicted probability	0.99	0.02	0.37	0.74	0.91	0.89	
Prediction after threshold	1	0	0	1	1	1	
Likelihood	0.99 ×	1 - 0.02 ×	0.37 ×	1 - 0.74 ×	0.91 ×	0.89	
	0.99 ×	0.98 ×	0.37 ×	0.26 ×	0.91 ×	0.89	= 0.08

Advanced: Under the hood, our models actually try to maximize log-likelihood

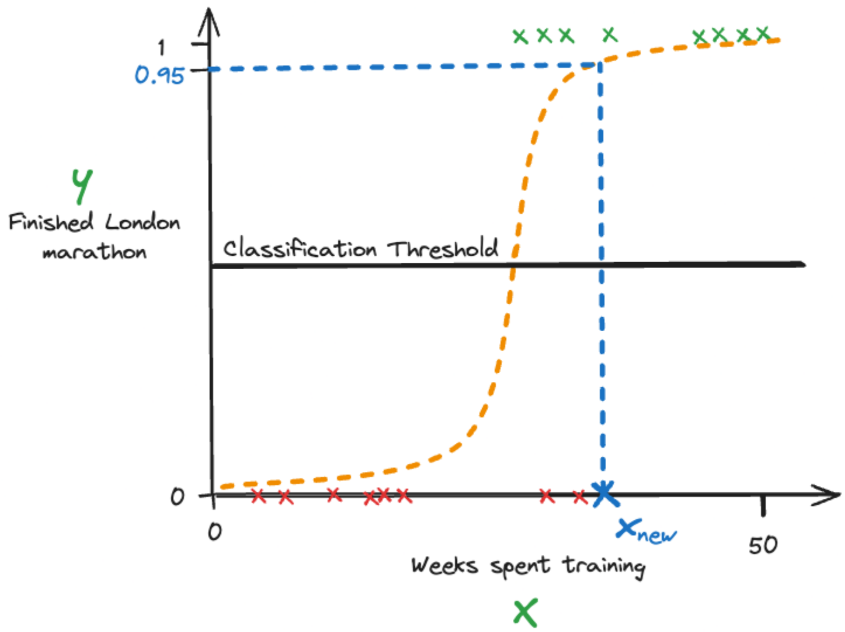
- Still gives the same optimum point as maximizing likelihood
- But is guaranteed to be convex
- More numerically stable



👀 You'll see more visualizations like this in ML weeks!

📖 [Further reading on using log loss and why we use it here! (https://medium.com/towards-data-science/why-not-mse-as-a-loss-function-for-logistic-regression-589816b5e03c)](https://medium.com/towards-data-science/why-not-mse-as-a-loss-function-for-logistic-regression-589816b5e03c)

🚀 By maximizing **Likelihood**, a Logistic Regression is really just trying to predict **probabilities**



👉 On average, for **100** observations
$x_1, \ldots, x_{100}$
that are predicted to have
$\hat{y}_i$
close to **0.95**, e.g. in [0.94-0.96]

- $\approx$ **95** of them will turn up to be true $y_i = 1$
- Your model will classify them correctly $\approx$ 95% of the time

☝️ We say that Logistic Classifiers are **calibrated** classifiers! (very important)

3. Interpreting Logistic Regression

3.1 Reading coefficients

🥋 Let's take an example!

🚢 The `Titanic` dataset contains survival outcomes (0/1) for ~900 passengers of the Titanic:

In [ ]:
```
titanic = sns.load_dataset("titanic")
titanic.head(3)
```

Out[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False

Without any feature

👉 To start, let's run the following logistic regression:

$$Survived_i = \beta_0$$

In [ ]:
```
model1 = smf.logit(formula='survived ~ 1', data=titanic).fit();
model1.params
```

```
Optimization terminated successfully.
         Current function value: 0.665912
         Iterations 4
```

Out[ ]:
```
Intercept   -0.473288
dtype: float64
```

> The log-odd of surviving titanic is -0.47

❓ What does the intercept correspond to ❓

`log_odd` = $log(\frac{p}{1-p})$ = β_0 = -0.47

⇔ `odds` = $\frac{p}{1-p}$ = $exp(-0.47)$ = 0.62

⇔ `Probability` $p = \frac{0.62}{1+0.62}$ = 38%

> Chance of surviving = 38%

🕵️ Let's double check this:

In [ ]:
```
cross_tab = pd.DataFrame({
    'count': titanic['survived'].value_counts(),
    'percentage': titanic['survived'].value_counts(normalize=True)
    })

round(cross_tab,2)
```

Out[ ]:

	count	percentage
0	549	0.62
1	342	0.38

With 1 continuous feature

Let's add another term to the model:

$$Survived = \beta_0 + \beta_{fare} Fare$$

$Fare$
= Fare that passenger paid (in dollars, continuous variable)

In [ ]:
```python
model3 = smf.logit(formula='survived ~ fare', data=titanic).fit()
model3.params
```

```
Optimization terminated successfully.
         Current function value: 0.627143
         Iterations 6
```

Out[ ]:
```
Intercept   -0.941330
fare         0.015197
dtype: float64
```

How to interpret the `fare` coefficient?

> Increasing fare by 1 dollar increases the log odds of surviving by 0.015

Taking the exponential:
$\exp(0.015) = 1.01$

> For each additional dollar spent on fare, the odds of surviving increase by 1%

In [ ]:
```python
model3 = smf.logit(formula='survived ~ fare', data=titanic).fit()
model3.params
```

```
Optimization terminated successfully.
         Current function value: 0.627143
         Iterations 6
```

Out[ ]:
```
Intercept   -0.941330
fare         0.015197
dtype: float64
```

How to interpret the intercept?

> The log-odds of surviving for a passenger who paid nothing is -0.94

With 1 categorical feature

Let's add one term to the model:

$$Survived = \beta_0 + \beta_{class} pclass$$

$pclass$
- Corresponds to the passenger class as a categorical variable (1,2 or 3)

In [ ]:
```
model2 = smf.logit(formula='survived ~ C(pclass)', data=titanic).fit()
model2.params
```

```
Optimization terminated successfully.
         Current function value: 0.607805
         Iterations 5
```

Out[ ]:
```
Intercept         0.530628
C(pclass)[T.2]   -0.639431
C(pclass)[T.3]   -1.670399
dtype: float64
```

> 0.53 is the log-odds of surviving for a passenger who was in the first class

> -0.63 is the **decrease** in the log-odds of survival for a 2nd class passenger, **relatively** to a 1st class passenger.

$$\log(odds_2) - \log(odds_1) = -0.63$$

$\Leftrightarrow$

$$\log(\frac{odd_2}{odd_1}) = -0.63$$

$\Leftrightarrow$

$$\frac{odds_1}{odds_2} = \exp(0.63) = 1.87$$

> The odds of surviving in 2nd class is **divided** by 1.87 compared to the 1st class!

With multiple features

In [ ]:
```
model2 = smf.logit(formula='survived ~ fare + C(sex) + age', data=tita
nic).fit()
model2.params
```

```
Optimization terminated successfully.
         Current function value: 0.501450
         Iterations 6
```

Out[ ]:
```
Intercept         0.934841
C(sex)[T.male]   -2.347599
fare              0.012773
age              -0.010570
dtype: float64
```

> Holding *fare* and *age* constant, being a male in Titanic reduces your log-odds of survival by 2.34 compared to a female passenger

4. Evaluate performance

In [ ]:
```
model4 = smf.logit(formula='survived ~ fare + C(sex) + age', data=tita
nic).fit()
model4.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.501450
         Iterations 6
```

Out[ ]:

Logit Regression Results

Dep. Variable:	survived	**No. Observations:**	714
Model:	Logit	**Df Residuals:**	710
Method:	MLE	**Df Model:**	3
Date:	Tue, 16 Nov 2021	**Pseudo R-squ.:**	0.2576
Time:	04:41:04	**Log-Likelihood:**	-358.04
converged:	True	**LL-Null:**	-482.26
Covariance Type:	nonrobust	**LLR p-value:**	1.419e-53

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	0.9348	0.239	3.910	0.000	0.466	1.403
C(sex)[T.male]	-2.3476	0.190	-12.359	0.000	-2.720	-1.975
fare	0.0128	0.003	4.738	0.000	0.007	0.018
age	-0.0106	0.006	-1.627	0.104	-0.023	0.002

🤓 **Fully annotated** model summary [cheatsheet (https://wagon-public-datasets.s3.amazonaws.com/data-science-images/lectures/decision-science/LogReg/log_reg_cheatsheet.png)](https://wagon-public-datasets.s3.amazonaws.com/data-science-images/lectures/decision-science/LogReg/log_reg_cheatsheet.png)

4.1 Inference

- **`p-values`** : work similarly to p-values in Linear Regression

- **`z-score`** is used instead of `t-score` because in a Bernoulli process, the variance is known and doesn't need to be estimated:
  $\sigma^2 = p(1 - p)$

Less stringent conditions compared to Linear OLS regression

✅ Random sampling

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

❌ NOT NEEDED: Residuals normally distributed and of equal variance

4.2 Goodness-of-fit (R^2 or equivalent?)

In [ ]:
```
model4.summary()
```

Out[ ]:

Logit Regression Results

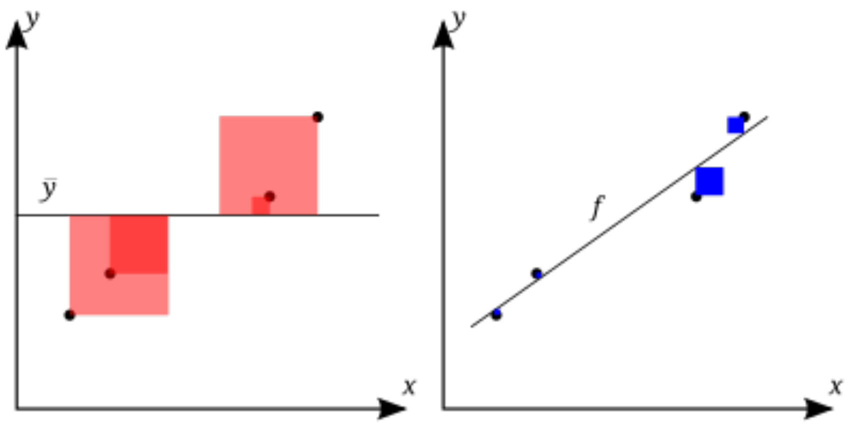
Dep. Variable:	survived	**No. Observations:**	714
Model:	Logit	**Df Residuals:**	710
Method:	MLE	**Df Model:**	3
Date:	Tue, 16 Nov 2021	**Pseudo R-squ.:**	0.2576
Time:	04:41:10	**Log-Likelihood:**	-358.04
converged:	True	**LL-Null:**	-482.26
Covariance Type:	nonrobust	**LLR p-value:**	1.419e-53

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	0.9348	0.239	3.910	0.000	0.466	1.403
C(sex)[T.male]	-2.3476	0.190	-12.359	0.000	-2.720	-1.975
fare	0.0128	0.003	4.738	0.000	0.007	0.018
age	-0.0106	0.006	-1.627	0.104	-0.023	0.002

`log-likelihood (LL)`

- `log(likelihood) =` $log\,($
- $\in [-\infty, 0]$
- the closer to 0 the better!
- **plays a similar role to "Sum of Squared Residuals" in Linear Regression**

$$\text{R-squared (linear regression)} = 1 - \frac{SS_{\text{resid}}}{SS_{\text{mean}}}$$



Pseudo R-squared for Logistic regression

$$1 - \frac{LL(predict)}{LL(\text{mean})}$$

y	1	0	1	0	1	1
y pred	0.99	0.02	0.97	0.04	0.91	0.89
y mean	0.66	0.66	0.66	0.66	0.66	0.66

$$LL(mean) \quad \log(0.99 \times 1 - 0.02 \times 0.97 \times 1 - 0.04 \times 0.91 \times 0.89)$$

$$LL(mean) \quad \log(0.66 \times 1 - 0.66 \times 0.66 \times 1 - 0.66 \times 0.66 \times 0.66)$$

`Pseudo R-squared`

✅ in `[0-1]`
✅ useful to compare models predicting the same problem from same data X

❌ is not as descriptive as the R-squared, once the classification threshold is applied

- e.g. predicted probabilities of 0.49 vs. 0.51 are extremely close...
- ... but yield opposite classification predictions!

📅 We will discover the most important **Performance Metrics for classification** during the Machine Learning module

(`accuracy` , `precision` , `recall` , `f1_score` , ...)

5. Multicollinearity issues in Linear/Logistic Regression

Imagine that a feature X_k is a linear combination of other features (e.g. $X_8 = X_1 + X_3$)

🤔 How can you "Vary X_k **while holding all other features constant**" ❓❗ You can't.

4.1 Strict multicollinearity

Which feature matrix is best suited for regression?

In [ ]:
```
A
```

Out[ ]:
```
array([[1., 0., 1.],
       [0., 1., 1.],
       [0., 0., 0.]])
```

In [ ]:
```
B
```

Out[ ]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [ ]:
```
print(' rank(A):', np.linalg.matrix_rank(A), '\n',
      'rank(B):', np.linalg.matrix_rank(B))
```

```
 rank(A): 2
 rank(B): 3
```

☝️ The rank of a matrix is the dimension of the vector space generated by its columns

⚠️ Feature Matrix needs to be "full rank" to run any Linear/Logistic model!

❗
$$C_3 = C_1 + C_2$$
: the third column/feature is a linear combination of the first two columns so would need to remove it to perform a correct Linear/Logistic Model

Example

In [ ]:
```python
mpg = sns.load_dataset('mpg').dropna().drop(columns=['origin', 'name',
'displacement'])
mpg.corr().style.background_gradient(cmap='coolwarm')
```

Out[ ]:

	mpg	cylinders	horsepower	weight	acceleration	model_year
mpg	1.000000	-0.777618	-0.778427	-0.832244	0.423329	0.580541
cylinders	-0.777618	1.000000	0.842983	0.897527	-0.504683	-0.345647
horsepower	-0.778427	0.842983	1.000000	0.864538	-0.689196	-0.416361
weight	-0.832244	0.897527	0.864538	1.000000	-0.416839	-0.309120
acceleration	0.423329	-0.504683	-0.689196	-0.416839	1.000000	0.290316
model_year	0.580541	-0.345647	-0.416361	-0.309120	0.290316	1.000000

In [ ]:
```python
mpg['lin_comb'] = 10 * mpg['cylinders'] - 0.3 * mpg['horsepower']
mpg.head(3)
```

Out[ ]:

	mpg	cylinders	horsepower	weight	acceleration	model_year	lin_comb
0	18.0	8	130.0	3504	12.0	70	41.0
1	15.0	8	165.0	3693	11.5	70	30.5
2	18.0	8	150.0	3436	11.0	70	35.0

In [ ]:
```
# Matrix is not full-rank!
print(mpg.shape)
np.linalg.matrix_rank(mpg)
```

```
(392, 7)
```

Out[ ]:
```
6
```

In [ ]:
```
smf.ols(formula='weight ~ cylinders + horsepower + lin_comb', data=mp
g).fit().params
```

Out[ ]:
```
Intercept     528.876711
cylinders       3.375029
horsepower     16.840512
lin_comb       28.698140
dtype: float64
```

In [ ]:
```
# Now, change just a bit one single observation by 1% just on one feat
ure
mpg.loc[0,'horsepower'] = mpg.loc[0,'horsepower']*1.01
smf.ols(formula='weight ~ cylinders + horsepower + lin_comb', data=mp
g).fit().params
```

Out[ ]:
```
Intercept       524.838981
cylinders     11398.211049
horsepower     -325.000813
lin_comb      -1110.583430
dtype: float64
```

In [ ]:
```
# Statsmodels gives us a clear WARNING [2]
# Summary table also reads 'Covariance Type: nonrobust'
smf.ols(formula='weight ~ cylinders + horsepower + lin_comb', data=mp
g).fit().summary()
```

Out[ ]:

OLS Regression Results

Dep. Variable:	weight	**R-squared:**	0.846
Model:	OLS	**Adj. R-squared:**	0.845
Method:	Least Squares	**F-statistic:**	712.9
Date:	Tue, 16 Nov 2021	**Prob (F-statistic):**	1.99e-157
Time:	04:41:18	**Log-Likelihood:**	-2832.3
No. Observations:	392	**AIC:**	5673.
Df Residuals:	388	**BIC:**	5689.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	524.8390	56.865	9.230	0.000	413.038	636.640
cylinders	1.14e+04	8616.113	1.323	0.187	-5541.902	2.83e+04
horsepower	-325.0008	258.480	-1.257	0.209	-833.198	183.196
lin_comb	-1110.5834	861.454	-1.289	0.198	-2804.285	583.118

Omnibus:	12.222	**Durbin-Watson:**	1.199
Prob(Omnibus):	0.002	**Jarque-Bera (JB):**	24.236
Skew:	0.069	**Prob(JB):**	5.46e-06
Kurtosis:	4.210	**Cond. No.**	5.84e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.84e+04. This might indicate that there are strong multicollinearity or other numerical problems.

⚠️ `partial coefficients` of collinear features become **extremely sensitive to changes in the data**

⚠️ Can trust **neither** your `partial coefficients` **nor** their associated `p-values`

✅ No impact on `R2`
✅ No impact on partial coefficients for non-collinear features

5.2 Strong (but not strict) multicollinearity is still a problem!

In [ ]:
```
mpg['lin_comb'] = mpg['lin_comb'] +  0.05 * np.random.rand(mpg.shape[0])
np.linalg.matrix_rank(mpg)
```

Out[ ]:
```
7
```

In [ ]:
```
smf.ols(formula='weight ~ cylinders + horsepower + lin_comb', data=mpg).fit().params
```

Out[ ]:
```
Intercept      544.560499
cylinders     8645.424045
horsepower    -242.419334
lin_comb      -835.251174
dtype: float64
```

In [ ]:
```
# Again, change just a bit one single observation in the dataset and check the OLS results
mpg.loc[0,'horsepower'] = mpg.loc[0,'horsepower']*0.8
smf.ols(formula='weight ~ cylinders + horsepower + lin_comb', data=mpg).fit().params
```

Out[ ]:
```
Intercept     524.231116
cylinders       5.429155
horsepower     16.782814
lin_comb       28.688715
dtype: float64
```

5.3 How to detect multicollinearity

In [ ]:
```python
# Correlation matrix is not sufficient to detect soft or event strict multicollinearity
mpg.corr().style.background_gradient(cmap='coolwarm')
```

Out[ ]:

	mpg	cylinders	horsepower	weight	acceleration	model_year	lin_comb
mpg	1.000000	-0.777618	-0.777708	-0.832244	0.423329	0.580541	-0.445217
cylinders	-0.777618	1.000000	0.841000	0.897527	-0.504683	-0.345647	0.762619
horsepower	-0.777708	0.841000	1.000000	0.863997	-0.687454	-0.413903	0.292037
weight	-0.832244	0.897527	0.863997	1.000000	-0.416839	-0.309120	0.554680
acceleration	0.423329	-0.504683	-0.687454	-0.416839	1.000000	0.290316	-0.067723
model_year	0.580541	-0.345647	-0.413903	-0.309120	0.290316	1.000000	-0.113352
lin_comb	-0.445217	0.762619	0.292037	0.554680	-0.067723	-0.113352	1.000000

❗ Correlation matrix detects **bivariate** collinearity between 2 features only ❗

🚀 VIF: Variance Inflation Factor

- A measure of the amount of multicollinearity per feature
- The higher, the more multicollinear, the less useful the feature with a high VIF...
- Computed by **regressing one feature as function of all others** features and measuring `R-squared`

$$VIF(X_k) = \frac{1}{1 - R_k^2}$$

❗ Warning ❗

When you plan to use multiple features in a linear model, do not forget to scale your data...

In [ ]:
```
mpg_scaled = mpg.copy()

for feature in mpg_scaled.columns:
    mu = mpg[feature].mean()
    sigma = mpg[feature].std()
    mpg_scaled[feature] = mpg_scaled[feature].apply(lambda x: (x-mu)/s
igma)

mpg_scaled
```

Out[ ]:

	mpg	cylinders	horsepower	weight	acceleration	model_year	lin_comb
0	-0.697747	1.482053	0.016488	0.619748	-1.283618	-1.623241	1.832112
1	-1.082115	1.482053	1.575127	0.842258	-1.464852	-1.623241	0.740041
2	-0.697747	1.482053	1.185207	0.539692	-1.646086	-1.623241	1.208893
3	-0.953992	1.482053	1.185207	0.536160	-1.283618	-1.623241	1.210761
4	-0.825870	1.482053	0.925261	0.554997	-1.827320	-1.623241	1.523028
...	...	...	...	...	...	...	...
393	0.455359	-0.862911	-0.478450	-0.220842	0.021267	1.634321	-0.956249
394	2.633448	-0.862911	-1.362268	-0.997859	3.283479	1.634321	0.104867
395	1.095974	-0.862911	-0.530439	-0.803605	-1.428605	1.634321	-0.895028
396	0.583482	-0.862911	-0.660413	-0.415097	1.108671	1.634321	-0.738686
397	0.967851	-0.862911	-0.582429	-0.303253	1.398646	1.634321	-0.832209

392 rows × 7 columns

In [ ]:
```
from statsmodels.stats.outliers_influence import variance_inflation_fa
ctor as vif
# compute VIF factor for feature index 0
vif(mpg_scaled.values, 0)
```

Out[ ]: 5.236622675084839

In [ ]:
```
df = pd.DataFrame()

df["features"] = mpg_scaled.columns

df["vif_index"] = [vif(mpg_scaled.values, i) for i in range(mpg_scaled.shape[1])]

round(df.sort_values(by="vif_index", ascending = False),2)
```

Out[ ]:

	features	vif_index
1	cylinders	2091.85
2	horsepower	943.73
6	lin_comb	668.13
3	weight	11.25
0	mpg	5.24
4	acceleration	2.61
5	model_year	1.90

☝️ Consider VIF value
$\geq$
10 as a potential cause for concern (rule of thumb)

Summary: Regression Cheat Sheet

Check	Description	Diagnosis (OLS)	Diagnosis (Logit)
Goodness-of-fit	How well does our y_{pred} explain the true y ?	R-squared	pseudo R-squared
Statistical significance	Are regression coefficients trustworthy?	p-values and t-tests	p-values and z-tests
Inference conditions	Can we trust the p-values ?	Residual plots	Not needed
Multicollinearity	Minimal dependence between features	VIF analysis	VIF analysis

Bibliography 📚

- [StatsQuest - Logistic Regression (https://www.youtube.com/playlist?list=PLblh5JKOoLUKxzEP5HA2d-Li7IJkHfXSe)](https://www.youtube.com/playlist?list=PLblh5JKOoLUKxzEP5HA2d-Li7IJkHfXSe) (1-h youtube, very good intuitive summary)

🚀 Your turn!

📆 Challenge 01

- Applying your skills in Logistic Regression

🔥 Challenge 02

- You have 1.5 days to answer the CEO's request based on all the analysis, notebooks, logics that you have been coding so far