REGRESSION VS. CLASSIFICATION

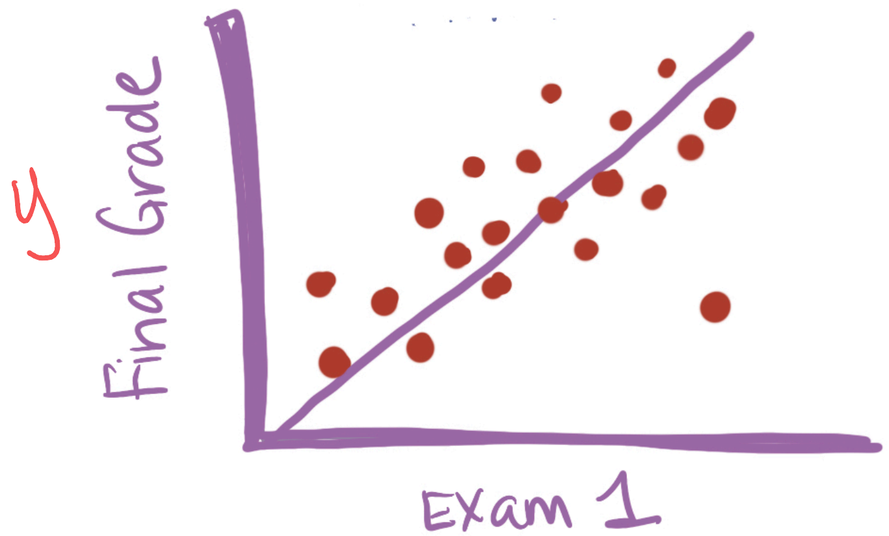
Predict Continuous Variable ←

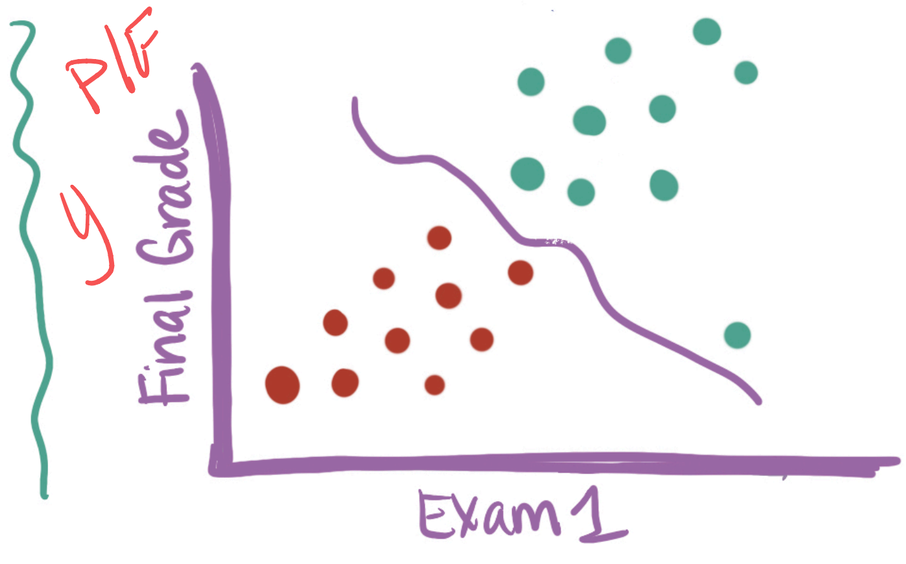
→ Predict Categorical Variable

How will each Student perform in the course?

Regression: Predict Final Grade

Classification: Predict Pass or Fail





ONLY diff is the Y, target.
input features can be same as classification

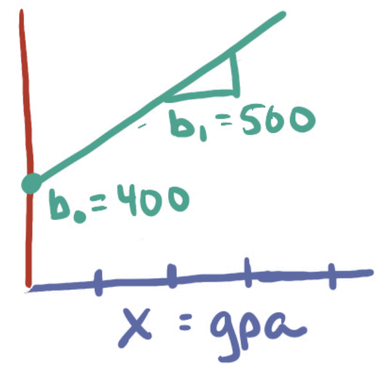
REGRESSION

Types

UNIVARIATE

SIMPLE

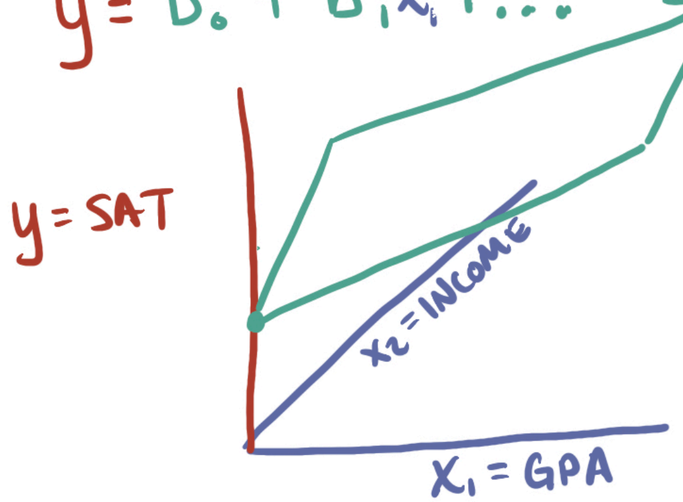
$y = b_0 + bX$



$SAT = 400 + 500 \cdot gpa$

MULTIPLE

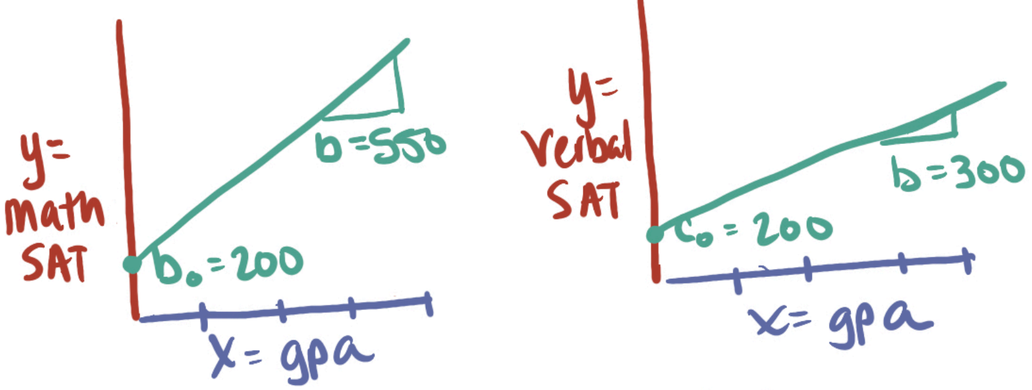
$y = b_0 + b_1 X_1 + \ldots + b_n X_n$



$SAT = 400 + 500 \cdot GPA + .005 \cdot INCOME$

MULTIVARIATE

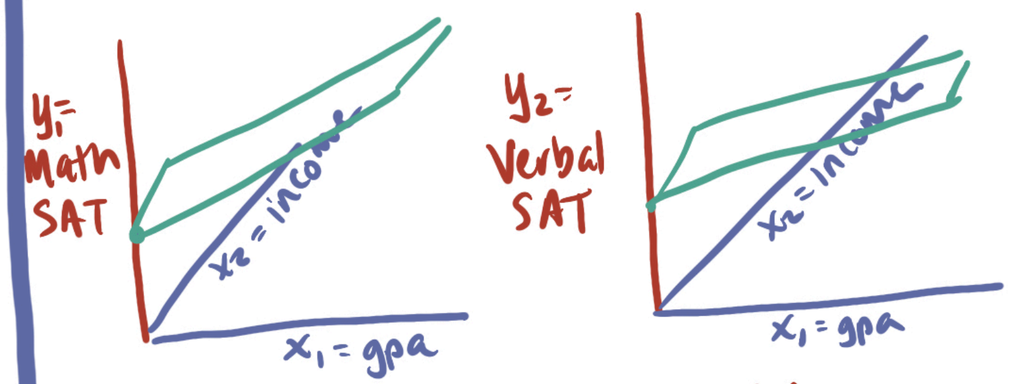
$y_1 = b_0 + bX$ $\quad y_2 = c_0 + cX$



Math SAT = $200 + 550 \cdot gpa$

Verbal SAT = $200 + 300 \cdot gpa$

$y_1 = b_0 + b_1 + \ldots + b_n$ $\quad y_2 = c_0 + c_1 + \ldots + c_n$



Math SAT = $200 + 550\ gpa + .003 \cdot income$

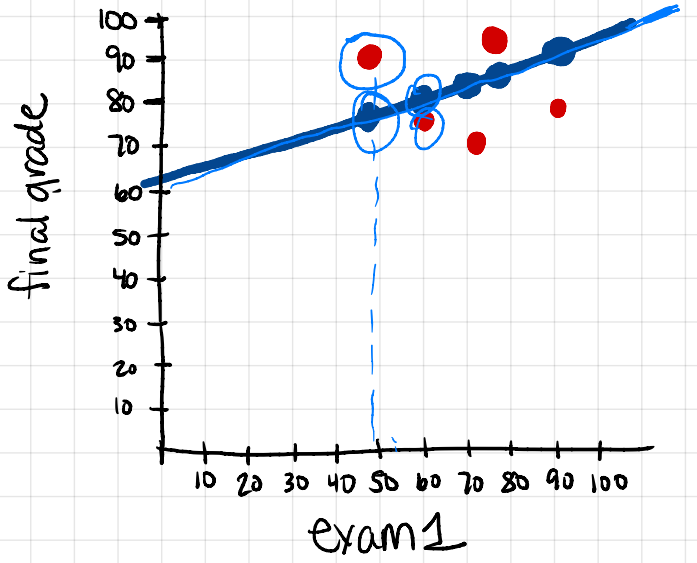
Verbal SAT = $200 + 250 \cdot gpa + .002 \cdot income$

evaluating regression Models

What does it mean to "evaluate a model"?

Student ID	Exam 1 x	Final grade y	Predicted final grade $\hat{y}$
1	48	90	77
2	60	76	80
3	71	70	84
4	76	94	87
5	90	80	92

$\hat{y} = b_0 + b_1 x$



Evaluating $\Rightarrow$

How well did my model predict the students' final grades?

Why evaluate models?

Q1 Does this model add ANY value?

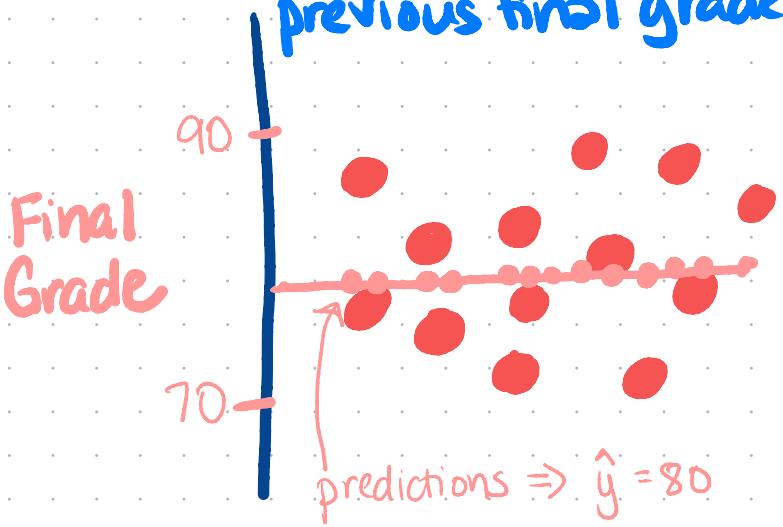
Q2 Which of these 2 models is better?

Q3 How much confidence should I have in this model?

Q1 Does my model add ANY value? Is it "Good Enough"?

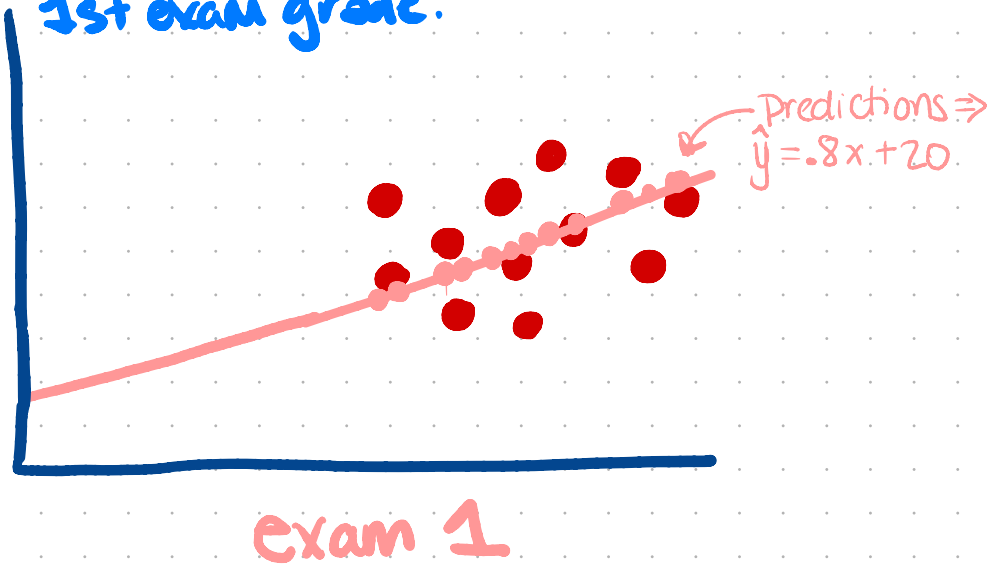
No Model ⇒

Predict final grade using only previous final grades.



My Model ⇒

predict final grade using the student's 1st exam grade.



Which is better?

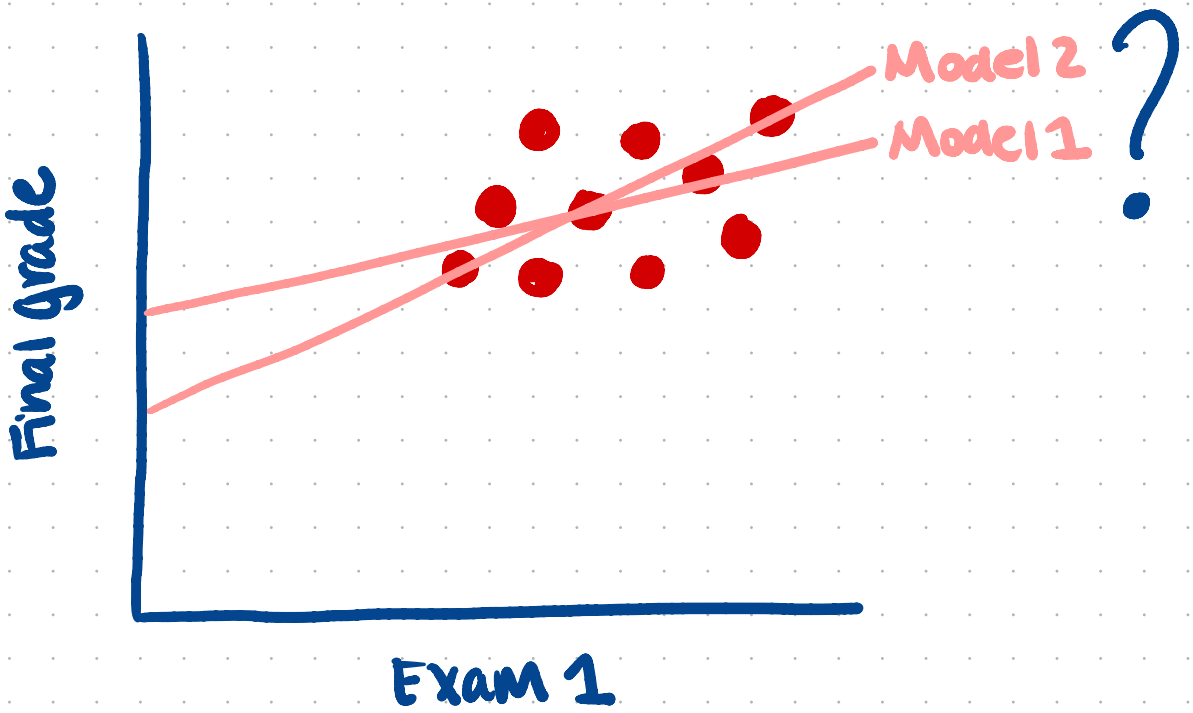
Predict final grade for each student to be 80!

??

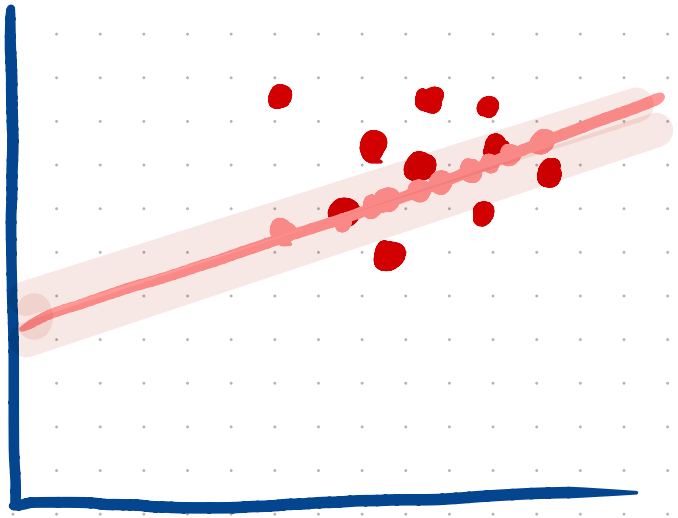
Predict the final grade for each student to be 80% of their 1st exam score + 20 points.

Q2: Which Model is better?

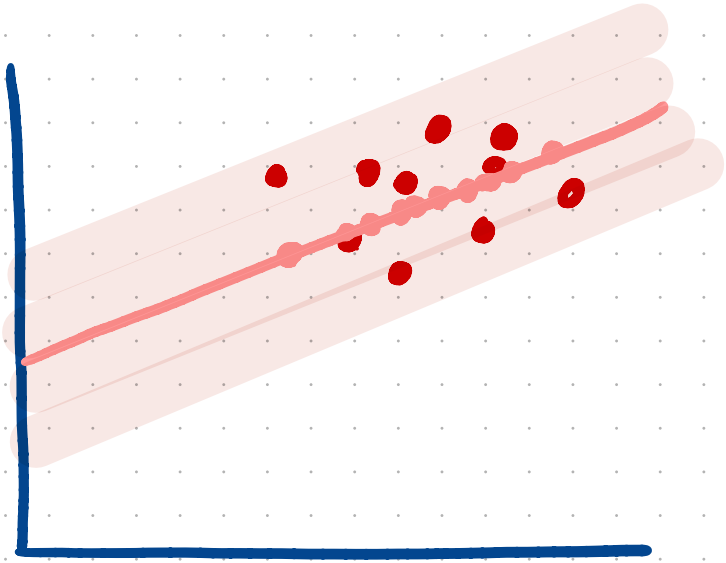
How can I determine which predicts better?



Q3: How much confidence should I have in this model?



It's gonna be close!



Yeah, we really have no idea ...

[Does my model add value?] [Which model is better?]

To Answer, we need a **single metric** that measures how well a model predicts.

Common Metrics: [Root Mean Squared Error]
Mean Absolute Error
Median Absolute Error
[Mean Squared Error]

RMSE (via MSE)

RESIDUALS
Error of each prediction ⇒ **SSE** square each error then sum ⇒ **MSE** Avg of the squared errors ⇒ **RMSE** sqrt of the avg.

Manual Calculation

$\hat{y}_1 - y_1 = r_1$
$\hat{y}_2 - y_2 = r_2$
$\vdots$
$\hat{y}_n - y_n = r_n$

residuals

$\rightarrow r_1^2 + r_2^2 + \ldots + r_n^2 = \text{SSE} \rightarrow \frac{\text{SSE}}{n} = \text{MSE} \rightarrow \sqrt{\text{MSE}} = \text{RMSE}$

RMSE in plain english:
"our predictions are off by an average of 5 pts on the final grades."

n = # of observations/rows
r_1 = residual, error for the 1st row/observation
$\hat{y}_1$ = **predicted** value for the 1st row/observation
y_1 = **actual** value for the 1st row/observation

RMSE: sklearn.metrics.mean_squared_error
math.sqrt

```
RMSE =
    sqrt(mean_squared_error(df.y, df.yhat))
```

Does my model work better than having no model?

guessing?
pred. mean?
pred mode?

* Applies to ordinary least squares algorithms such as StatsModels.api.ols & sklearn.linear_model.LinearRegression

Answer: run an f-regression test on R^2

if the p-value of the f-statistic is $< \alpha$ (~.05), then our model is better than having no model.

[R^2 = the explained variance score
= the coefficient of determination
= (pearson's R)^2 (when simple univariate regression)]

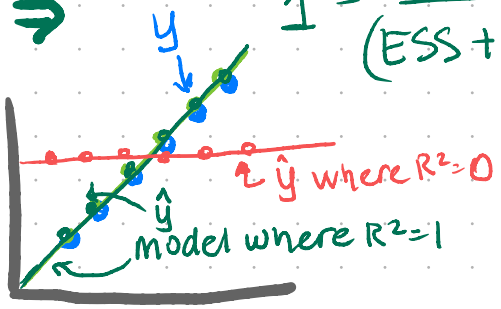
R^2

$$R^2 = \frac{ESS}{TSS} = \frac{\text{Explained SUM of Squares}}{\text{Total SUM of Squares}} = \frac{ESS}{(ESS + SSE)} = \frac{\text{Explained SS}}{(\text{Explained SS} + \text{SS Error})}$$

$R^2 = 1 \Rightarrow$ $1 = \frac{ESS}{(ESS+SSE)} \Rightarrow (ESS+SSE) = ESS \Rightarrow SSE = 0$ ←

<ALGEBRA>

NO ERROR!
Predictions = Actual
$\hat{y} = y$



$R^2 = 0 \Rightarrow$ $0 = \frac{ESS}{(ESS+SSE)} \Rightarrow (ESS+SSE)\cdot 0 = ESS \Rightarrow 0 = ESS$

Nothing is explained!
All error!

Python | Manual

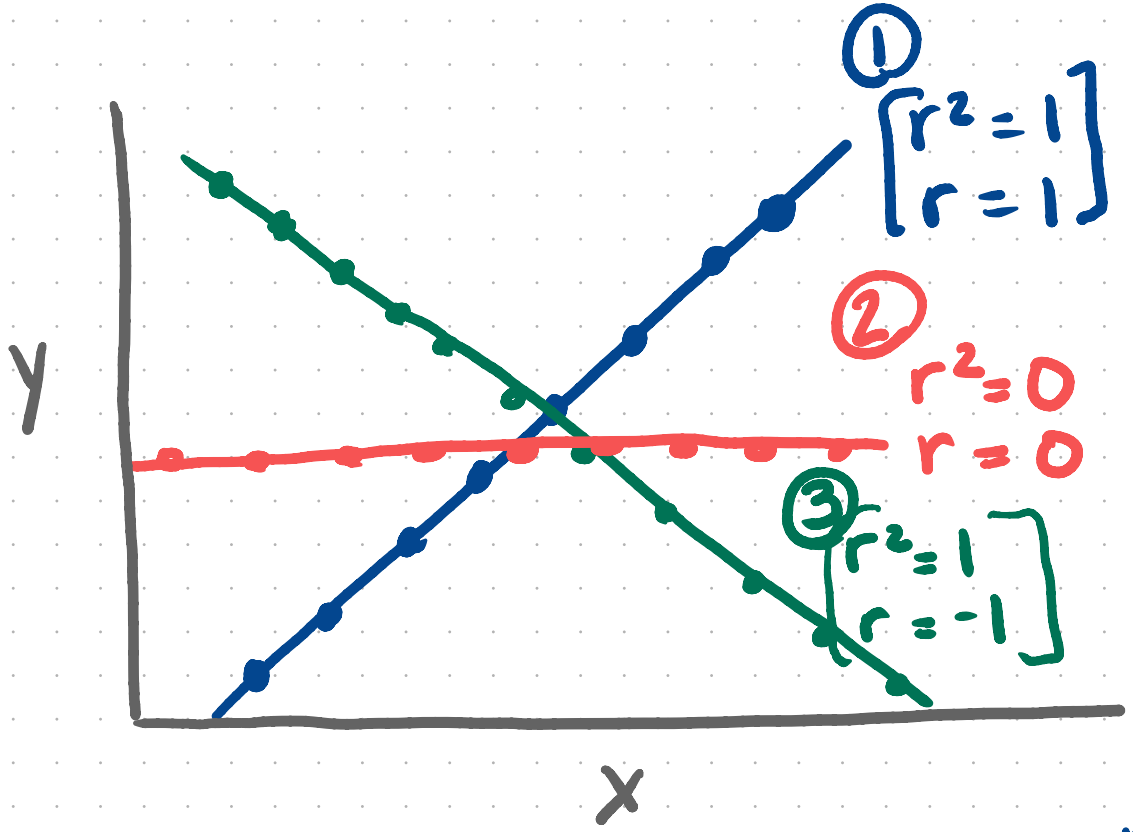
p-value

sklearn.metrics.explained_variance_score(df.y, df.yhat)

StatsModels.api.ols ⇒ My_ols_model.rsquared

my_ols_model.f_pvalue / sklearn.feature_selection.f_regression

r & r^2



1. As x↑, y↑ by a constant value
2. As x↑, y↓ by a constant value
3. As x↑, y doesn't change → x does not affect y in ANY way

- r → Perfect positive correlation
- r² → The variance in x explains ALL of the variance in y
- r → Perfect negative correlation
- r² → The variance in x explains ALL of the variance in y
- r → No correlation
- r² → The variance in x explains NONE of the variance in y.

Evaluate Models

Always use

★ 1. A metric related to the errors/residuals (RMSE, MSE, MAE)
- used more in practice
- more options for other types of regression

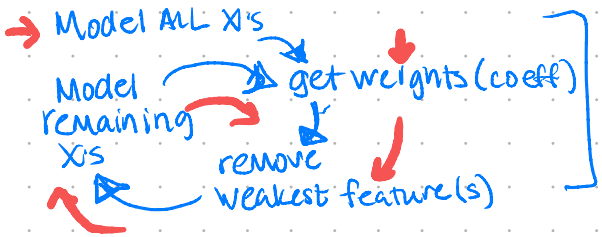
Sometimes use

2. R^2 and f_regression test
- use only with OLS, linear regression models.

Feature Selection ⇒ Evaluate Features

Algorithmic way to evaluate the impact features have on a target

1. Recursive Feature Elimination (RFE)



RFE repeats until you have your desired # of features

[ML Model = Linear_Regression
# of features = 2]

2. Select K Best

- run a statistical test, like f-regression, on each [feature + target] combination
- select the K features with the best score (⇒ lowest pvalue, e.g.)

[Statistic score, K = # of features]

Select K Best

select the best 'K' features

K? # of desired features

Best? best is defined by a statistical test -
the test selected depends on the type of problem.

Test / score functions

REGRESSION:

- **f_regression**: test **correlation** (LINEAR**)
- **mutual_info_regression**: test **dependence**
  - 0 ⇒ completely independent
  - 1 ⇒ completely dependent

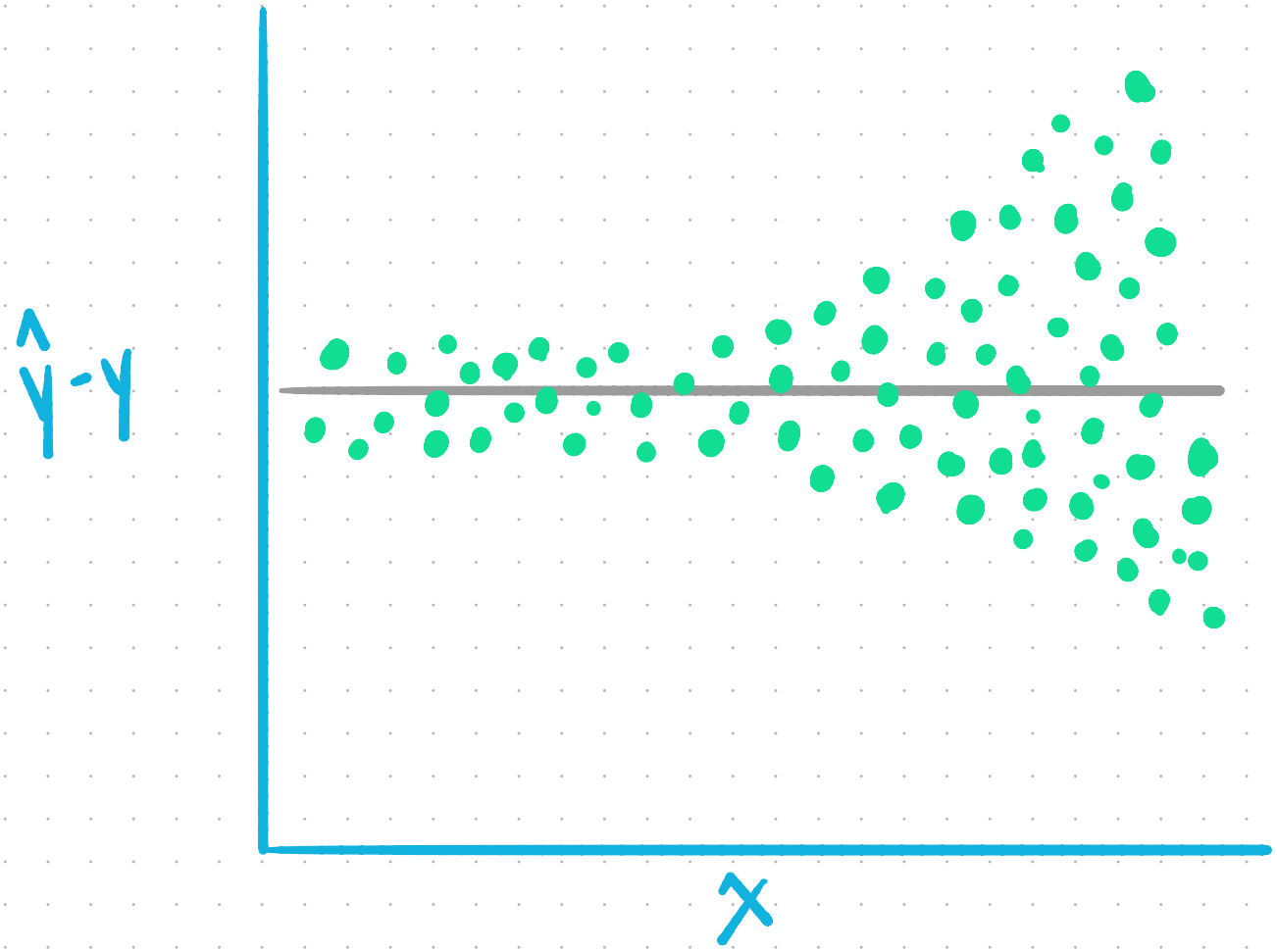
CLASSIFICATION:

- **f_classif**
- **mutual_info_classif**
- **chi2**
- **SelectFPR**
- ...

```
X_new = sklearn.feature_selection.SelectKBest(f_regression, k=3)
```

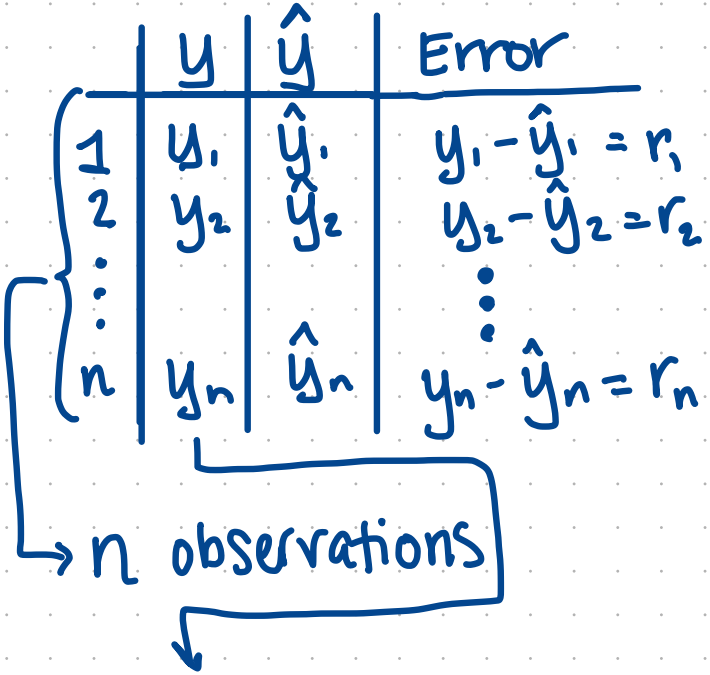
Heteroskedasticity

non-constant variance in ERRORS



Heteroskedasticity will bias standard errors

Mean Absolute Error



$$MAE = \frac{|r_1| + |r_2| + \ldots + |r_n|}{n}$$

$$= \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

y_i = actual value for observation/row i

$\hat{y}_i$ = predicted target value for observation/row i

r_i = residual, error, difference between actual & predicted value.

Mean Squared Error

	y	$\hat{y}$	Error
1	y_1	$\hat{y}_1$	$y_1 - \hat{y}_1 = r_1$
2	y_2	$\hat{y}_2$	$y_2 - \hat{y}_2 = r_2$
⋮			⋮
n	y_n	$\hat{y}_n$	$y_n - \hat{y}_n = r_n$

n observations

$$MSE = \frac{r_1^2 + r_2^2 + \ldots + r_n^2}{n}$$

$$= \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

y_i = actual value for observation/row i

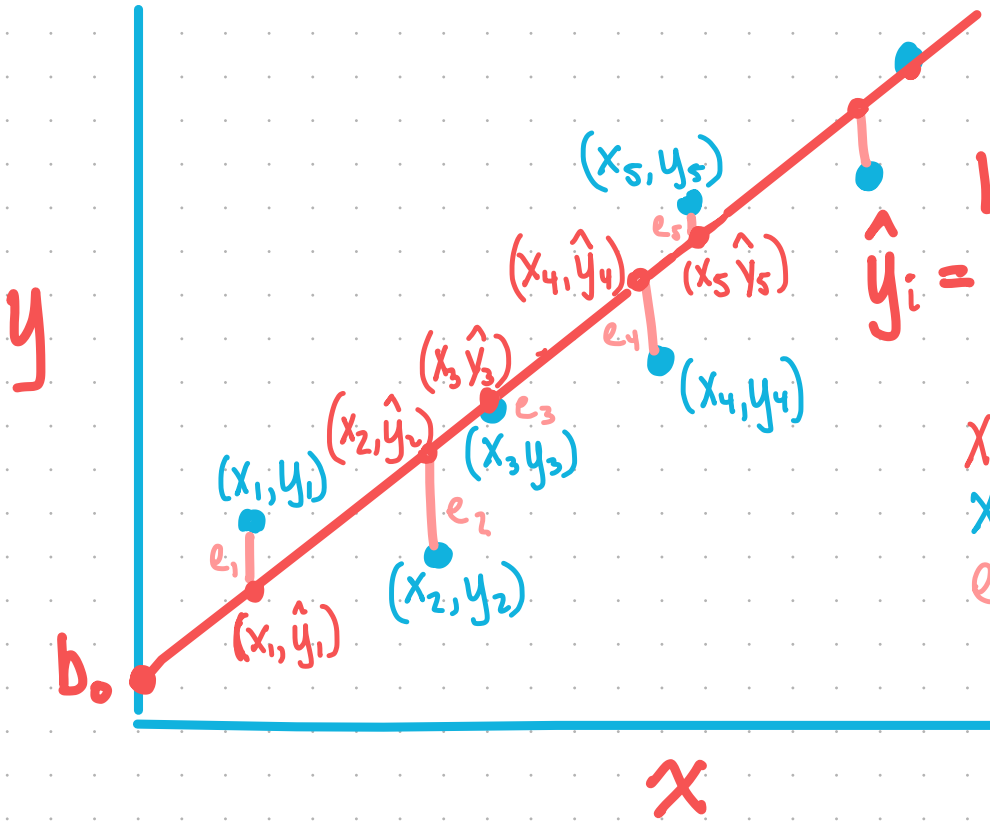
$\hat{y}_i$ = predicted target value for observation/row i

r_i = residual, error, difference between actual & predicted value.

Ordinary Least Squares

OLS

- Linear regression algorithm that finds the line of best fit that returns the smallest SSE $= (e_1^2 + e_2^2 + \ldots + e_n^2)$
- Often used as a baseline.



Line of best fit

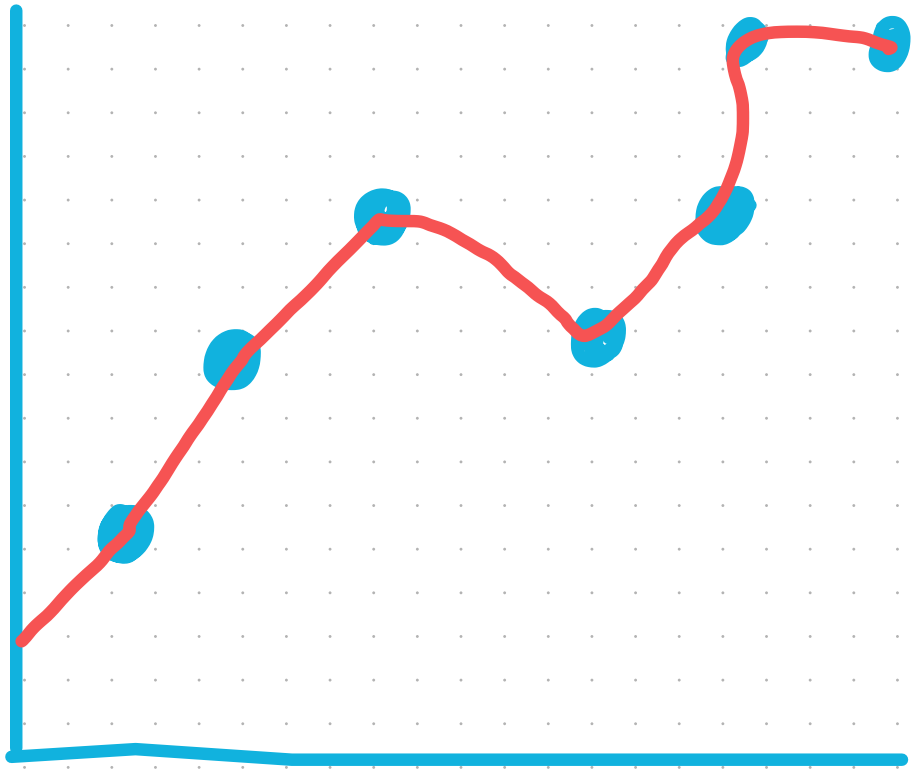
$$\hat{y}_i = b_0 + b_1 x_i + \varepsilon_i$$

$x_i, \hat{y}_i$ = predicted point
x_i, y_i = actual point
e_i = error, residual

```
y_pred = linear_model.LinearRegression().fit(X, y).predict(X)
```

[SkLearn]

OVERFITTING



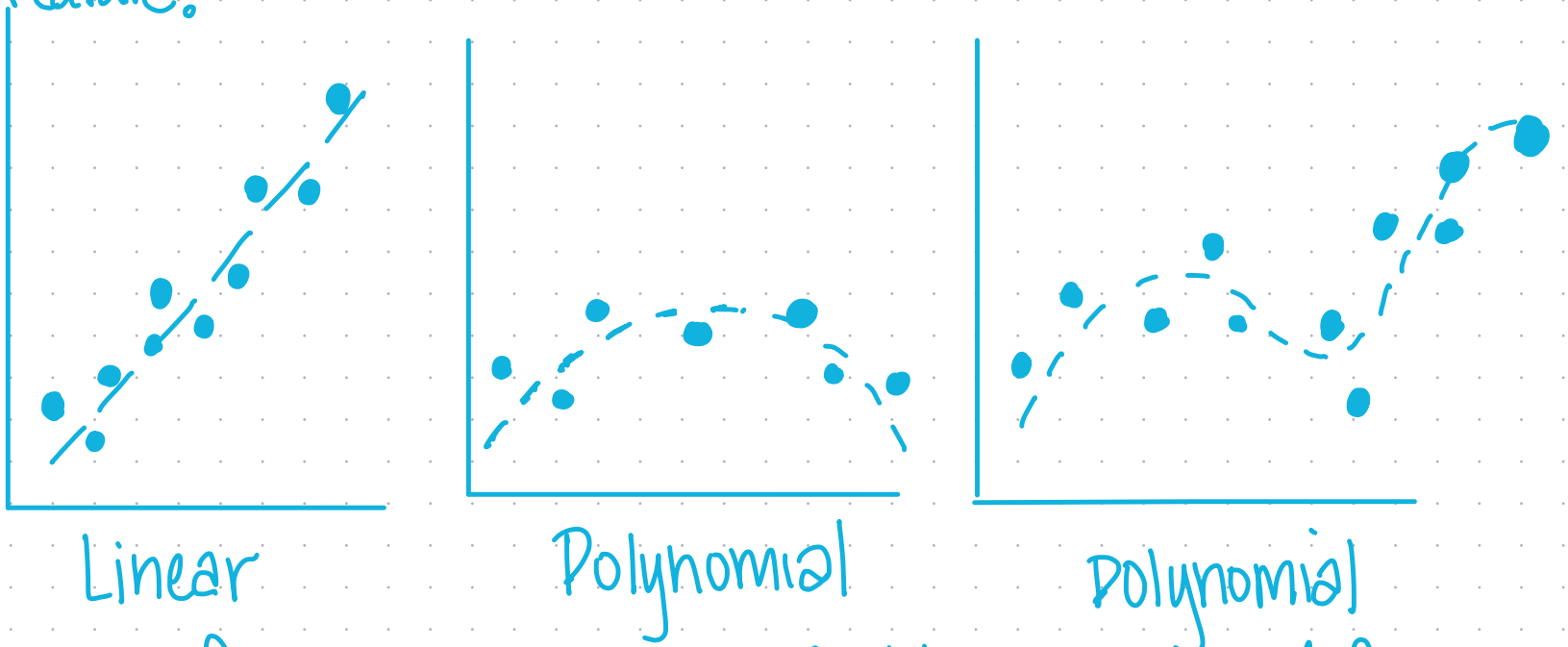
When a model "memorizes" specifics of training data and then loses the ability to generalize.

Polynomial Regression

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_d x_i^d + e_i$$

Polynomial terms

Still considered to be a linear model → the coeff are still linear!
x^2 is only a feature.



So... the original features must be converted into polynomial features.

```
x_poly = preprocessing.PolynomialFeatures(degree=2).fit_transform(x)
y_poly_pred = linear_model.LinearRegression().fit_predict(x_poly, y)
```

Sklearn