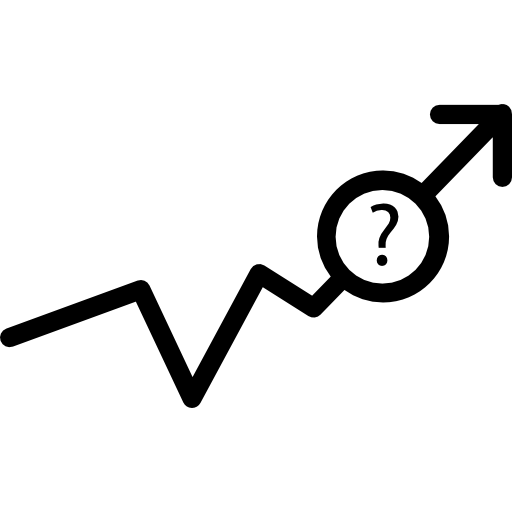
# **Performance Metrics**

## **Recap data preprocessing**

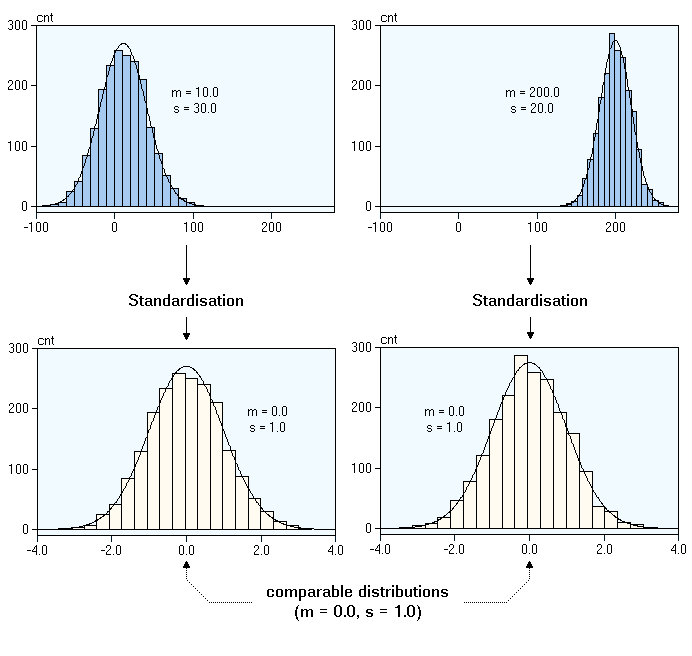
### **Missing data**

* Representations
* Locating
* Dropping
* Imputing



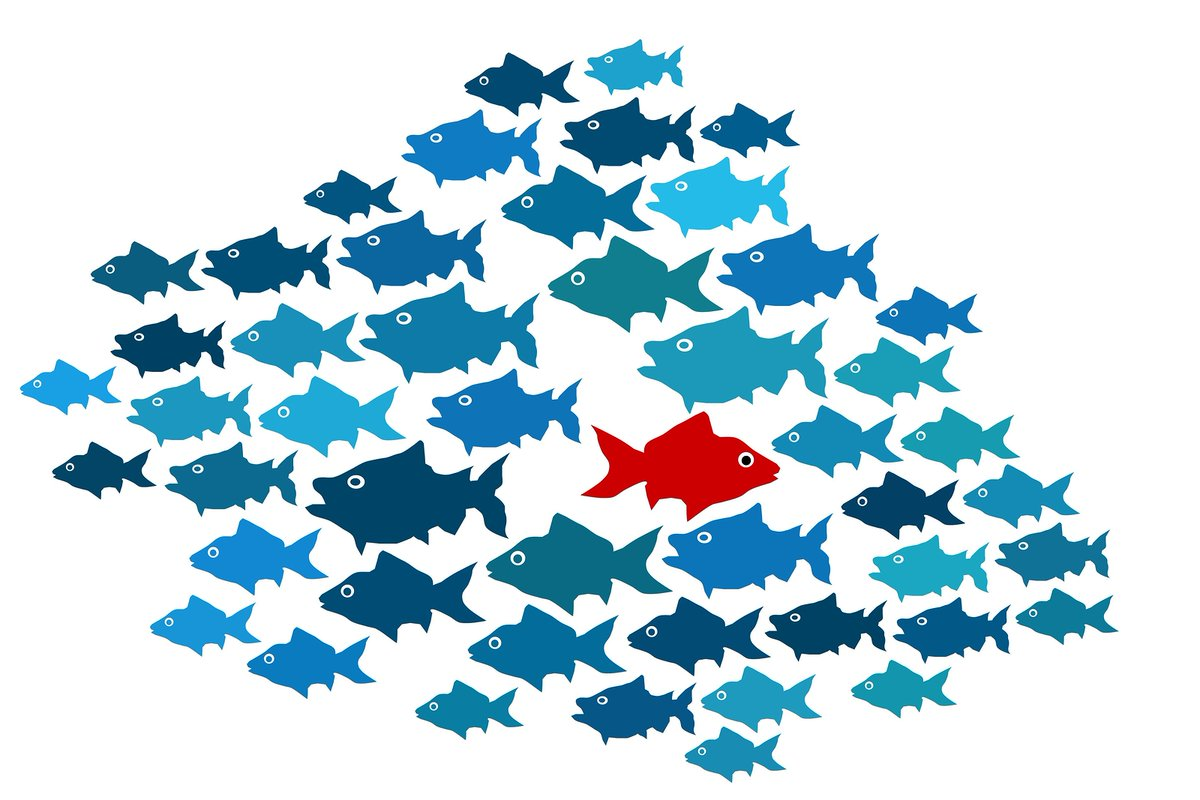
### **Scaling**

* Standardizing - centering around a **mean of 0** with an **std of 1**
* Normalizing (with MinMaxScaler) - scaling data to be **between 0 and 1**
* Robust Scaling - centering around a **median of 0** with the **IQR (Interquartile Range) of 1**

****

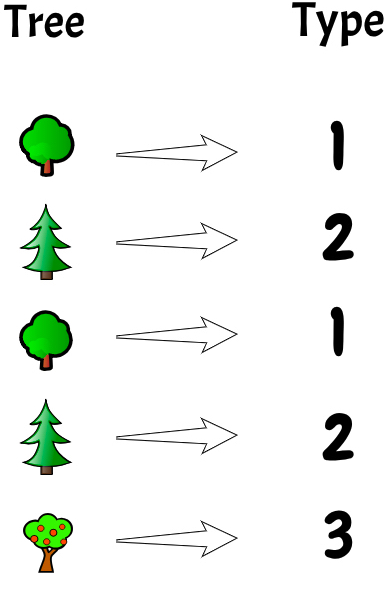
### **Outliers**

* Detecting
* Interpreting
* Dropping
* Preserving



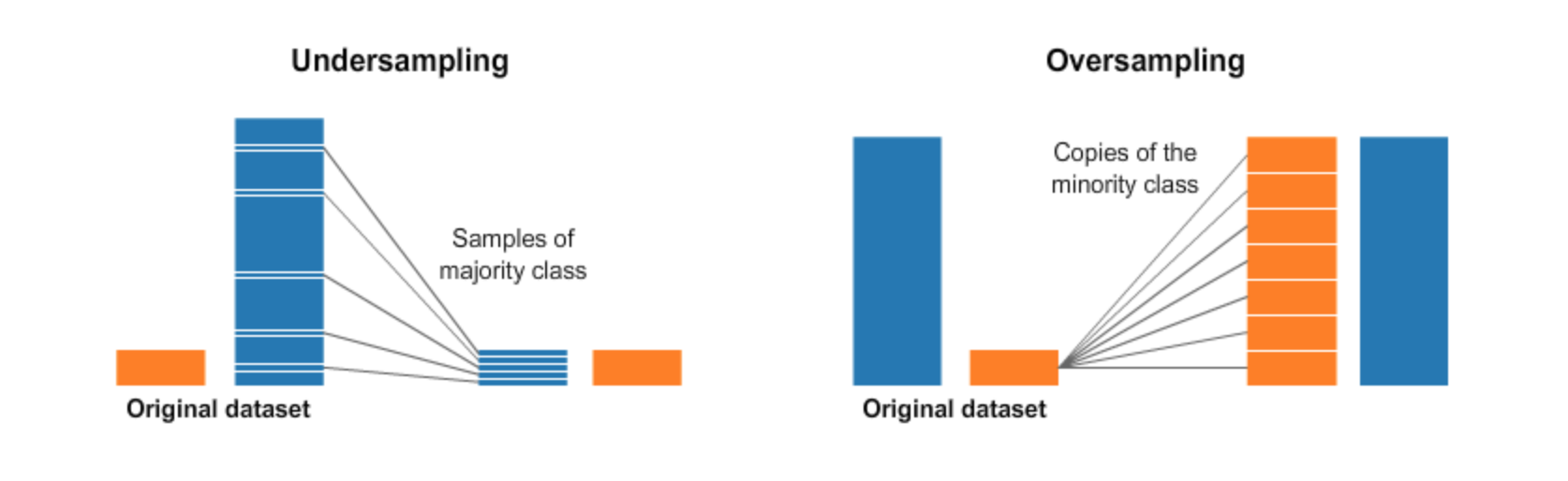
### **Encoding**

* Categorical
* Ordinal
* Label Encoding
* One Hot Encoding



### **Balancing**

* Oversampling
* Undersampling
* SMOTE
* ⚠️ Avoiding leakage



### **Discretizing**

* Continuous to categorical/ordinal
* Regression to Classification
* Binning a continuous value

### **Feature Engineering**

* Creating features with domain knowledge

### **Feature Selection**

* Garbage in, garbage out
* Feature permutation
* Correlation matrix

## **Today's Plan**

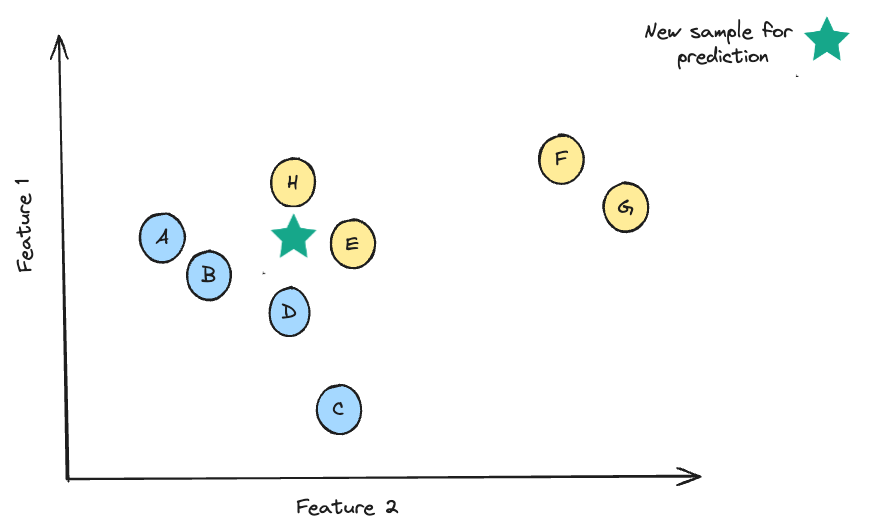
1. **New model type** - K-Nearest Neighbors
2. Model lifecycle
3. Baseline score
4. Regression Metrics
5. Classification Metrics
6. Error analysis

# **1. K-Nearest Neighbors**

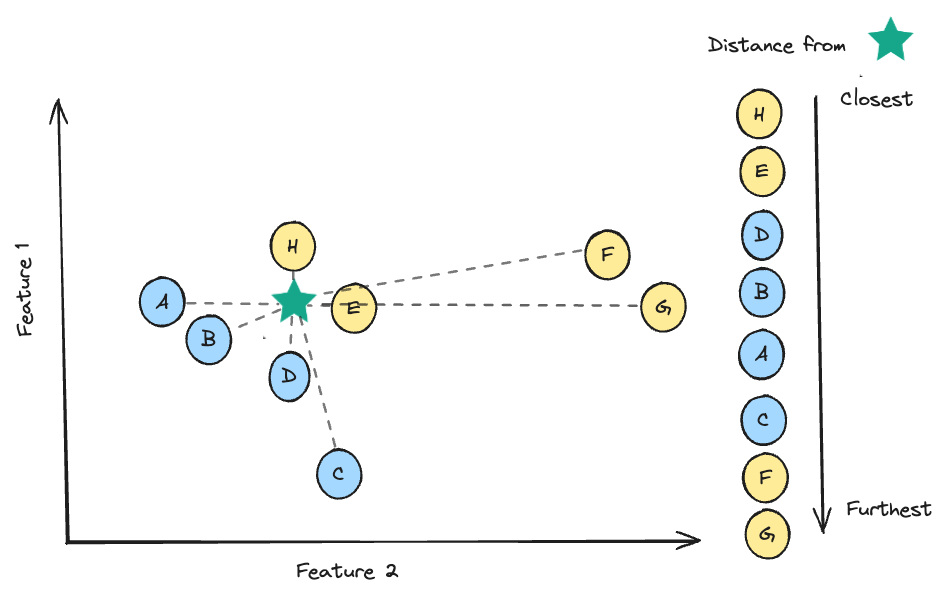
## **K-Nearest Neighbors**

K-Nearest Neighbors (KNN) is a non-linear, distance based model capable of solving both Regression and Classification tasks.

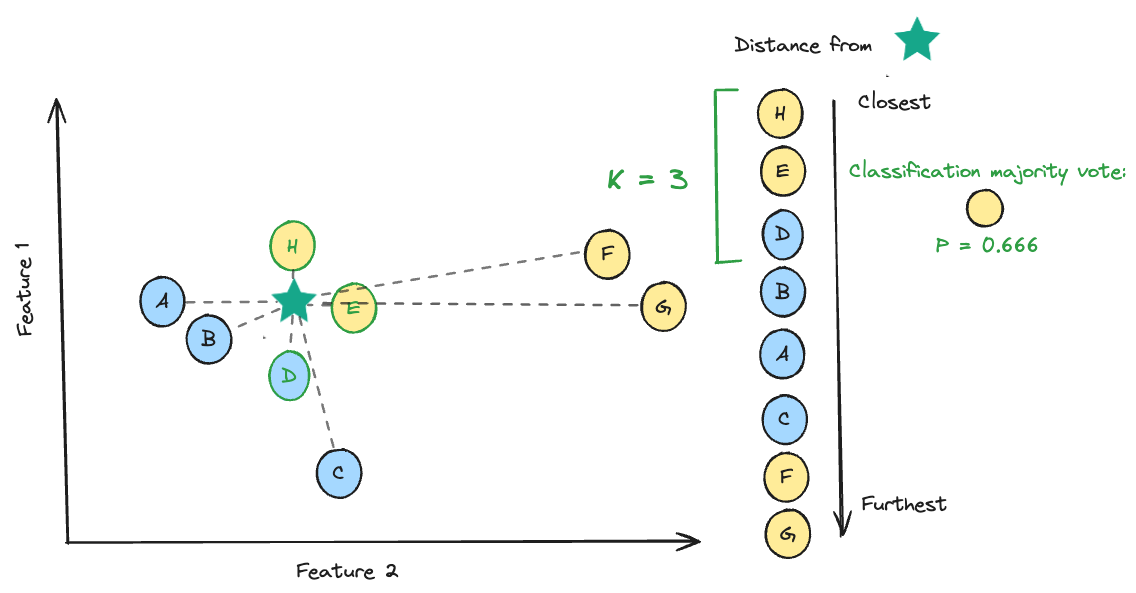
* Looks at K closest samples to make a prediction
* Up to us to determine K (hyperparameter)
* [Sklearn neighbors API](https://scikit-learn.org/stable/modules/classes.html?highlight=neighbors#module-sklearn.neighbors)



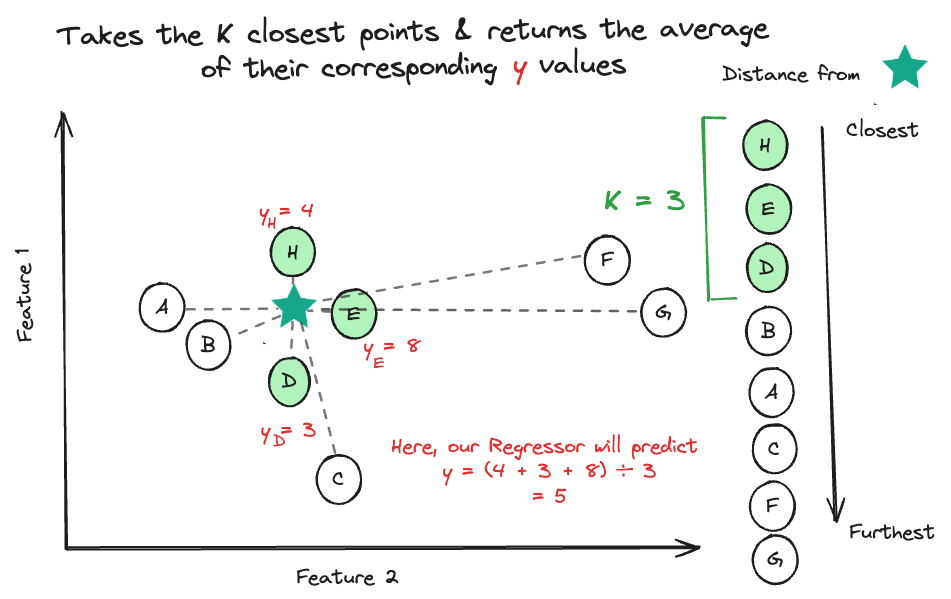
### **Calculate distance from sample**

****

### **Majority vote for classification with K=3 neighbours**

****

### **Average y values for regression for K = 3**

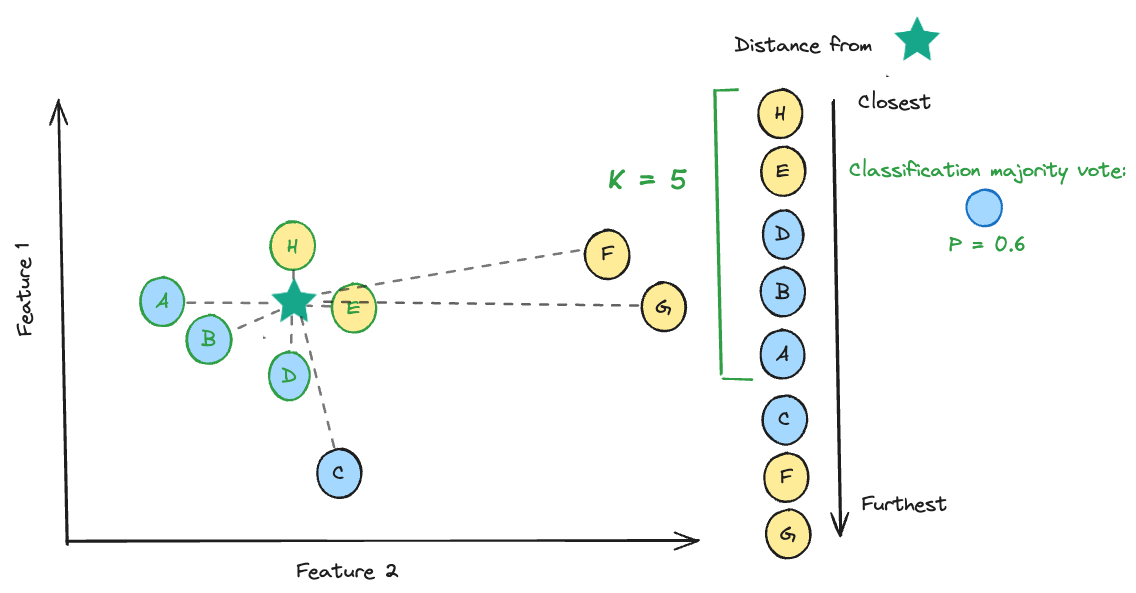
****

### **Choosing k**

The optimal 𝑘 will vary from dataset to dataset.

* Lower k values, less observations to use to make a prediction, prone to overfitting
* Higher k values, signal can be diluted, prone to underfitting

### **Changing K might change prediction**

****

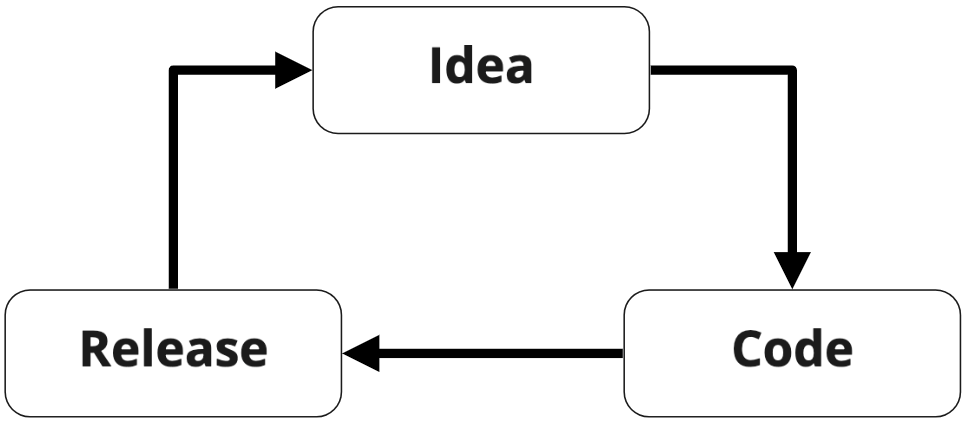
### **You can use KNN as a:**

* Regressor with [KNeighborsRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html)
* Classifier with [KNeighborsClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)

And more! ⭐

* Similarity detection and clustering with [NearestNeighbors](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html) - *more on this during Unsupervised Learning!*
* Imputing missing data with [KNNImputer](https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html)

# **2. Model lifecycle**

****

🧐 So how do we know if the current idea is better than a previous iteration? 🧐

### **Evaluation metrics**

Evaluation metrics are used to measure how well a machine learning model can perform a task.

🚨 A single metric will only tell one side of the story. Ensure you investigate a model's effectiveness from multiple angles 🚨

# **3. Baseline Score**

When evaluating a machine learning pipeline, you need a basis for comparison. A baseline score utilizes very simple strategies for prediction:

* **Classification**: Predicts a random or most frequent class
* **Regression**: Predicts a central tendency measure e.g. mean, median or mode

Examples will be shown on the following [dataset](https://wagon-public-datasets.s3.amazonaws.com/insurance_ML3.csv).

data.head()

|  | **age** | **bmi** | **children** | **smoker** | **charges** | **price\_range** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 19 | 27.900 | 0 | True | 16884.92400 | expensive |
| **1** | 18 | 33.770 | 1 | False | 1725.55230 | cheap |
| **2** | 28 | 33.000 | 3 | False | 4449.46200 | cheap |
| **3** | 33 | 22.705 | 0 | False | 21984.47061 | expensive |
| **4** | 32 | 28.880 | 0 | False | 3866.85520 | cheap |

## **💻 Regression baseline**

Using the [dummy estimators](https://scikit-learn.org/stable/modules/classes.html?highlight=dummy#module-sklearn.dummy) from sklearn.

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.dummy** **import** DummyRegressor

**from** **sklearn.linear\_model** **import** LinearRegression

*# Prepare X and y*

X = data[['age','bmi','children','smoker']]

y = data['charges']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,

test\_size = 0.3,

random\_state = 6) *# Holdout*

baseline\_model = DummyRegressor(strategy="mean") *# Baseline*

baseline\_model.fit(X\_train, y\_train) *# Calculate value for stratgy*

baseline\_model.score(X\_test, y\_test) *# Score model based on consistently predicting the strategy*

-0.001233635021205659

model = LinearRegression().fit(X\_train, y\_train) *# instantiate and fit model*

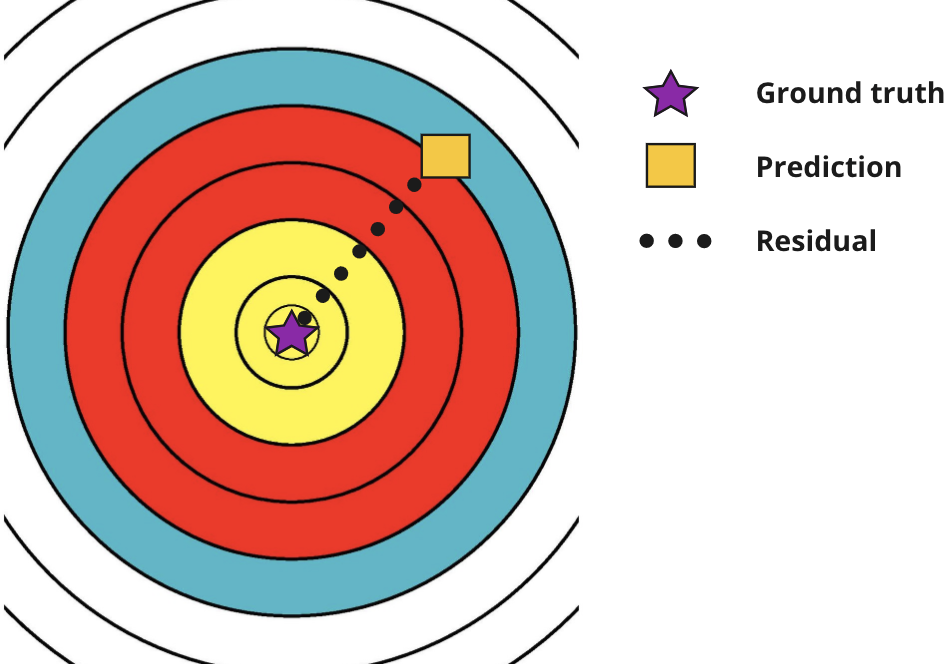
model.score(X\_test, y\_test) *# Score model*

0.773425820295562

🧐 Using a dummy model at the start allows us to move rapidly through each step of pipeline construction. This gives us the ability to identify obstacles downstream more quickly. 🧐

# **4. Regression Metrics**

### **Regression metrics**

****

There are lots of ways to measure the difference between two points...

## **Mean Squared Error (MSE)**

Mean of the squared differences between ground truth and predicted values.

* Useful to penalize large errors
* No sense of direction
* Not expressed in the same units as the target
* Very sensitive to outliers

MSE

=

1

n

n

∑

i

=

1

(

y

i

−

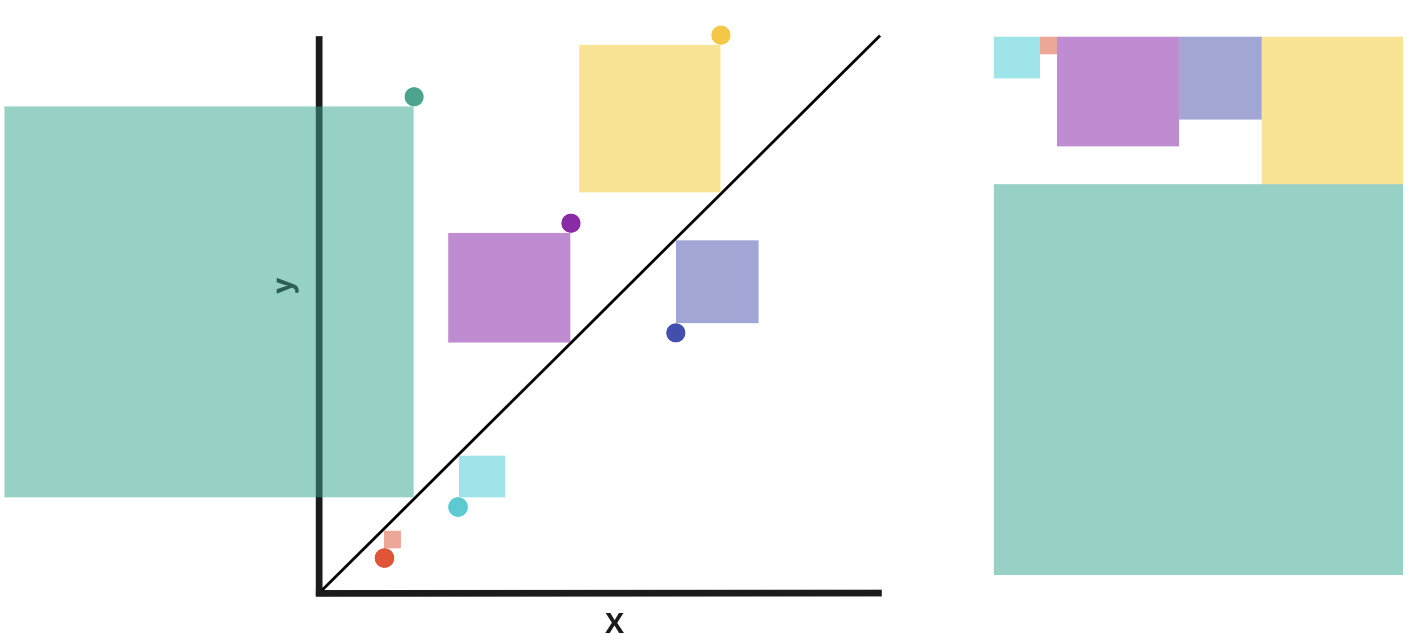
^

y

i

)

2



👉 Use MSE when:

* Larger errors have a disproportionally bigger impact, hence should be more penalized
  + Example: clinical trials, where an error of 4mg can be *more than twice as bad* as an error of 2mg
* Direction and unit of error does not matter
* Comparing the sensitivity of different models/methods to large errors

## **Root Mean squared error (RMSE)**

Square root of the Mean Square Error.

RMSE

=

⎷

1

n

n

∑

i

=

1

(

y

i

−

^

y

i

)

2

👉 Use RMSE when you want the MSE to be represented in the unit of the target, making it more interpretable

## **Mean absolute error**

The mean of the absolute differences between true values and predicted values.

* Less sensitive to outliers

MAE

=

1

n

n

∑

i

=

1

|

y

i

−

^

y

i

|

👉 Use MAE when errors can be penalized proportionally to their size

* Example: weather forecast, where an error of 4 degrees is simply *twice as bad* as an error of 2 degrees

## **Max Error**

The biggest error made by the model when predicting.

M

E

=

n

max

i

=

1

|

y

i

−

^

y

i

|

👉 Use Max Error when you want to limit the magnitude of the errors.

* Example: temperature management for a piece of equipment that can overheat *maximum* 2 degrees

## **Coefficient of determination**

## R

## 2

The proportion of the variance in the dependent variables that is explained by the independent variables.

* Typically ranges from 0 to 1
* Optimal score of 1

R

2

=

1

−

∑

n

i

=

1

(

y

i

−

^

y

i

)

2

∑

n

i

=

1

(

y

i

−

¯¯¯

y

)

2

👉 Use

R

2

when:

* The unit of the error is not important
* You want to compare between different datasets

## **💻 Comparing metrics**

**from** **sklearn.metrics** **import** mean\_squared\_error, mean\_absolute\_error, r2\_score, max\_error

**import** **math**

mse = mean\_squared\_error(y, y\_pred)

rmse = math.sqrt(mse)

mae = mean\_absolute\_error(y, y\_pred)

rsquared = r2\_score(y, y\_pred)

max\_error = max\_error(y, y\_pred)

print('MSE =', round(mse, 2))

print('RMSE =', round(rmse, 2))

print('MAE =', round(mae, 2))

print('R2 =', round(rsquared, 2))

print('Max Error =', round(max\_error, 2))

MSE = 36776200.68

RMSE = 6064.34

MAE = 4234.0

R2 = 0.75

Max Error = 29184.77

### **💻 Metrics during Cross-validation**

We can specify the scoring metric with the scoring parameter of cross\_validate

* A metric of choice can be specified directly with algorithms.
* If scoring is not set, the model's default scoring metric (using model's .score() method) is used.

**from** **sklearn.model\_selection** **import** cross\_validate

*# 5-Fold Cross validate model*

model = LinearRegression()

cv\_results = cross\_validate(model, X, y, cv=5,

scoring=['max\_error',

'r2',

'neg\_mean\_absolute\_error',

'neg\_mean\_squared\_error']

)

pd.DataFrame(cv\_results) *# Cross validation output*

|  | **fit\_time** | **score\_time** | **test\_max\_error** | **test\_r2** | **test\_neg\_mean\_absolute\_error** | **test\_neg\_mean\_squared\_error** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.005042 | 0.003597 | -24053.301788 | 0.760959 | -4210.447467 | -3.714635e+07 |
| **1** | 0.002747 | 0.002527 | -23060.600919 | 0.708823 | -4219.303638 | -3.781926e+07 |
| **2** | 0.002308 | 0.002045 | -26395.712047 | 0.776167 | -4026.911043 | -3.285122e+07 |
| **3** | 0.001178 | 0.001304 | -23282.860901 | 0.731409 | -4297.280431 | -3.987217e+07 |
| **4** | 0.001095 | 0.001199 | -29700.376643 | 0.756647 | -4218.673537 | -3.696269e+07 |

cv\_results['test\_r2'].mean() *# Cross validation results*

0.7468009563921238

To check all the available scoring metrics type 👉 [sklearn.metrics.SCORERS.keys()documentation](https://scikit-learn.org/stable/modules/model_evaluation.html).

Choosing a metric:

👉 Use **MSE** when you need to penalize large errors more strictly than small ones.

👉 Use **RMSE** when you need to penalize large errors, but see it in the unit of the target.

👉 Use **MAE** when all errors, large or small, have equal importance and you need easily interpretable results

👉 Use **Max Error** when you want to limit the magnitude of the errors.

👉 Use **R2** as a general "goodness-of-fit" metric to compare performance across changes to models and the data

There are many more regression metrics that you can check out in the [documentation](https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics).

# **5. Classification Metrics**

Two ways for a model to have a correct prediction:

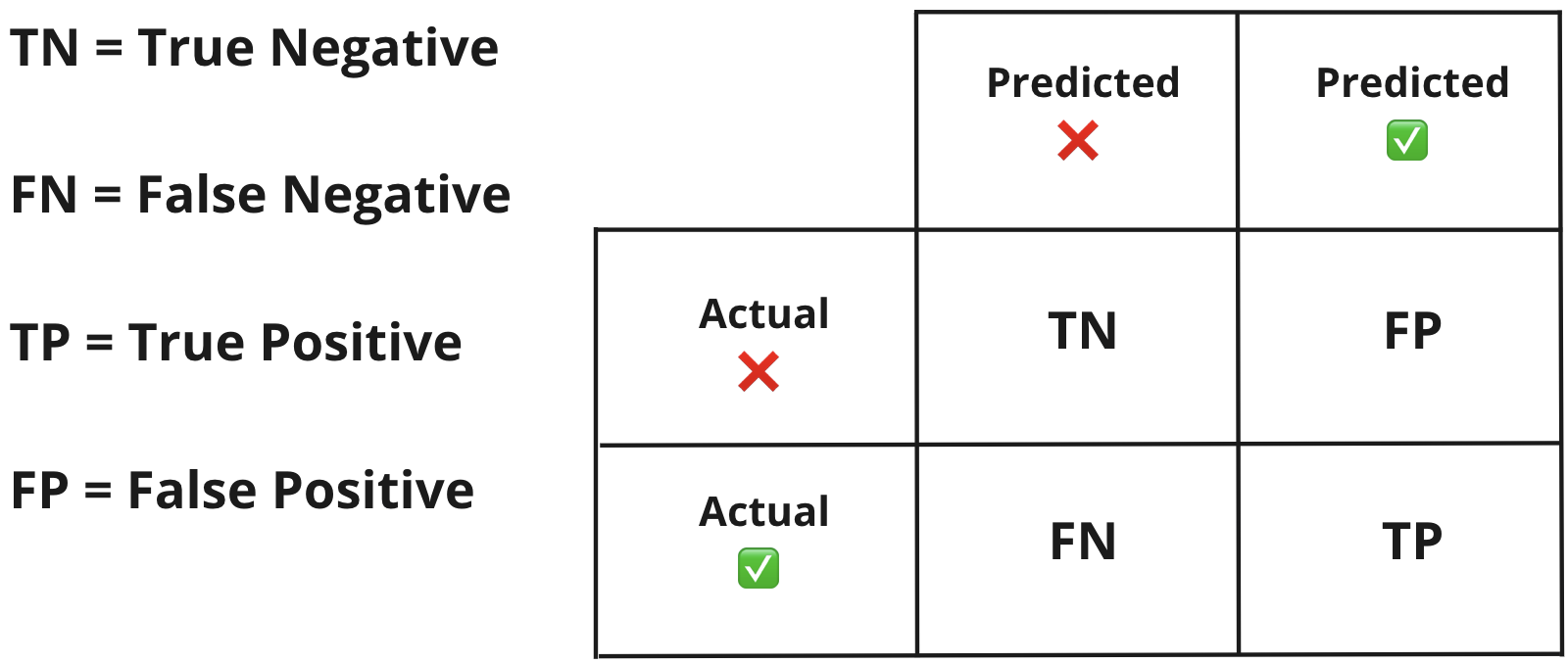
* **True Positive-** Correctly identifying a positive sample
  + Predicted ✅
  + Actual ✅
* **True Negative-** Correctly identifying a negative sample
  + Predicted ❌
  + Actual ❌

Two ways a model can have an incorrect prediction:

* **False Positive-** Incorrectly identifying a positive sample
  + Predicted ✅
  + Actual ❌
* **False Negative-** Incorrectly identifying a negative sample
  + Predicted ❌
  + Actual ✅

## **Confusion Matrix**

We grade and tally our predictions in a confusion matrix.



y\_test = [0, 1, 0, 0, 1, 0, 1, 1, 0, 1] *# actual truths*

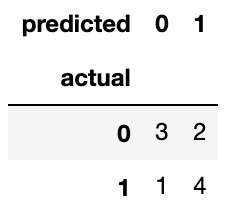
preds = [0, 0, 0, 0, 1, 1, 1, 1, 1, 1] *# predictions*

results\_df = pd.DataFrame({"actual": y\_test,

"predicted": preds}) *#Store results in a dataframe*

confusion\_matrix = pd.crosstab(index= results\_df['actual'],

columns = results\_df['predicted'])

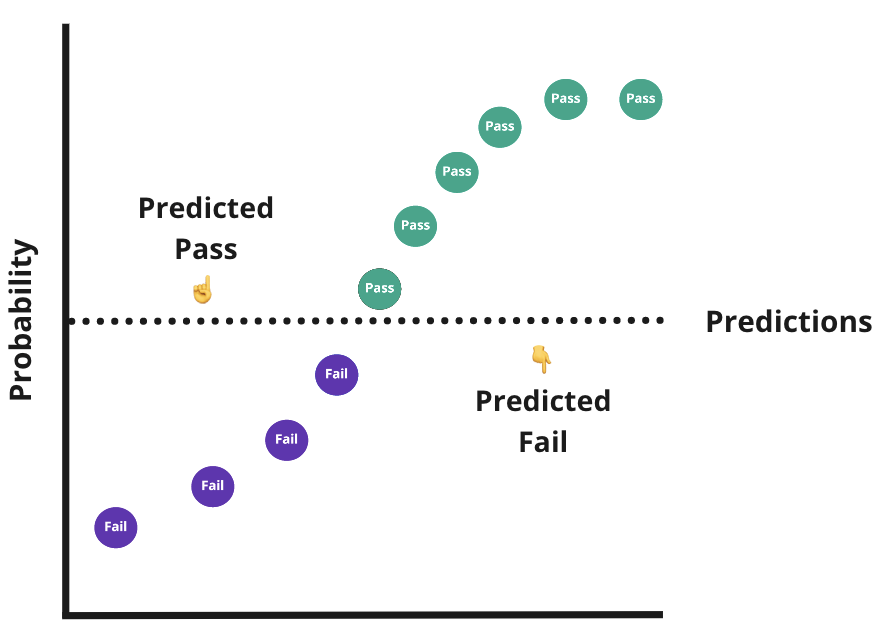


[Pandas crosstab documentation](https://pandas.pydata.org/docs/reference/api/pandas.crosstab.html)

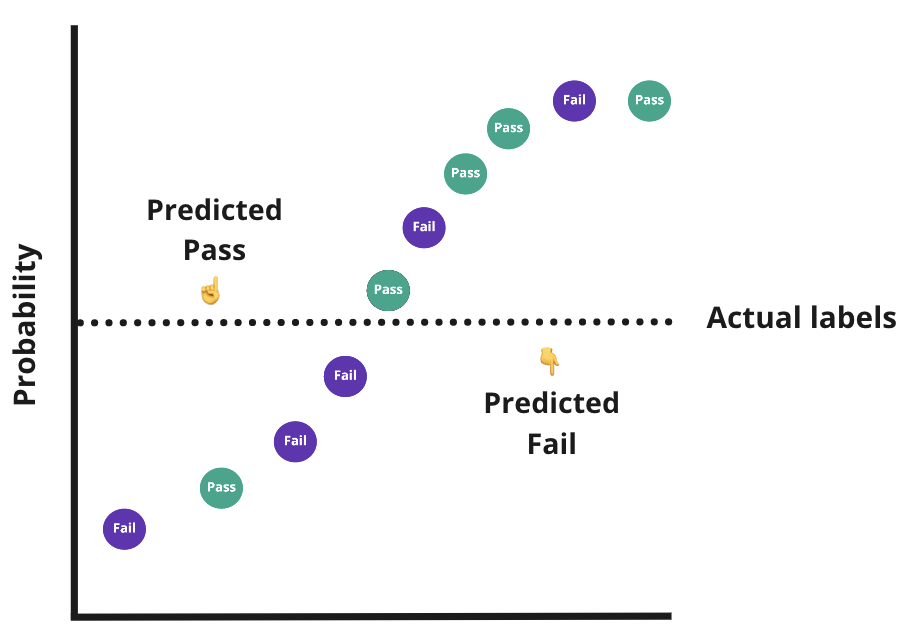
We're building a model to predict the likelihood of a student passing an exam.

Let's try and grade a model's predictions💪

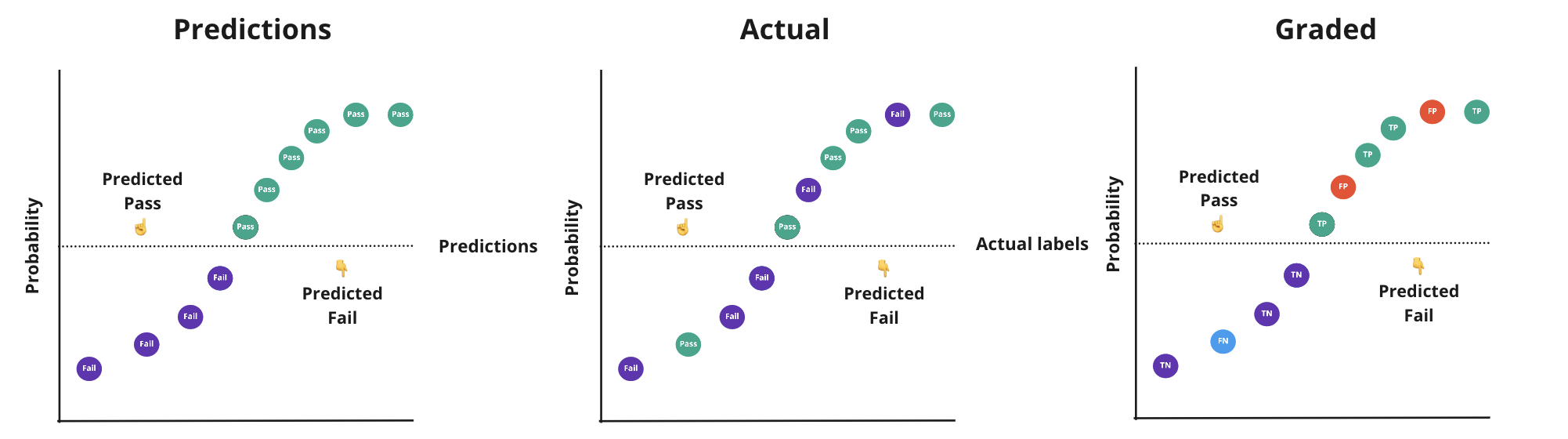
## **Predicted labels**

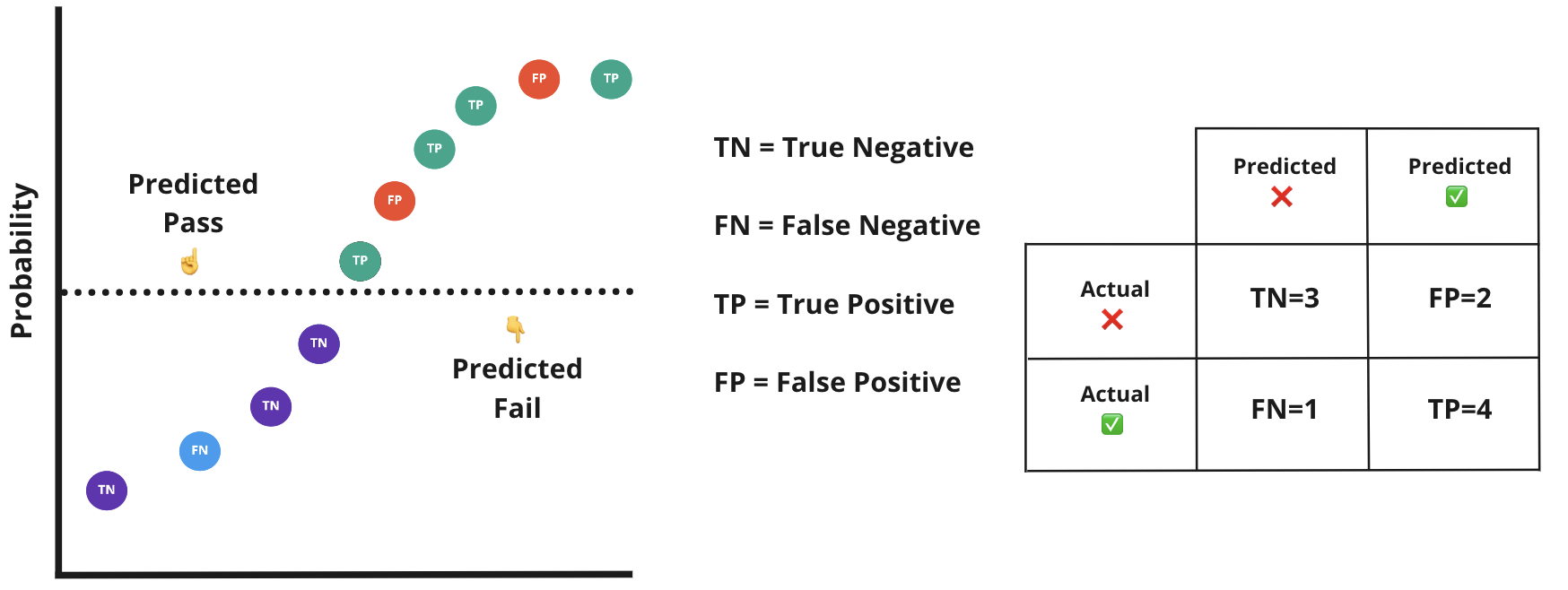
****

## **Actual labels**

****

## **Grading labels**

****

****

## **Accuracy**

Sum of the correct predictions divided by the sum of the overall number of predictions

a

c

c

u

r

a

c

y

=

T

P

+

T

N

T

P

+

T

N

+

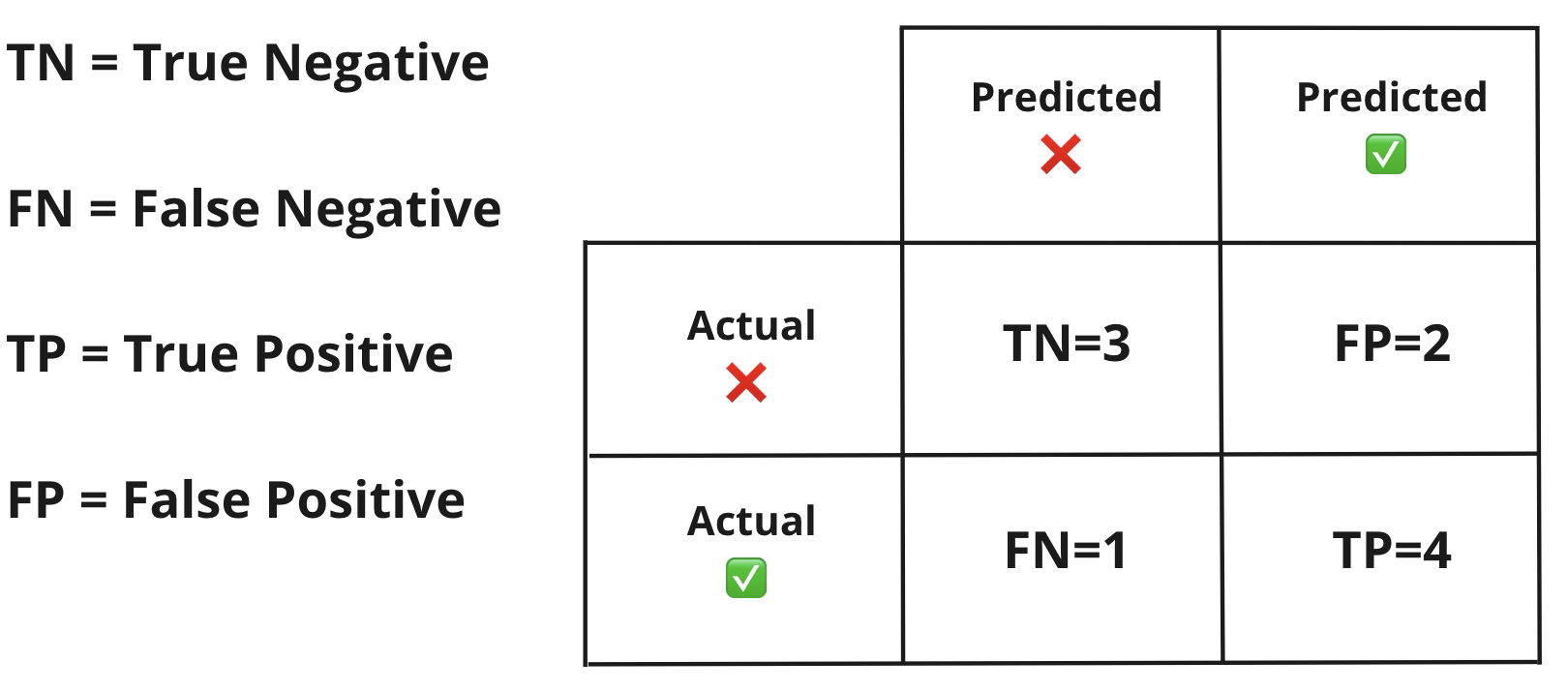
F

P

+

F

N



0.7

=

4

+

3

4

+

3

+

2

+

1

👉 [Sklearn's accuracy\_score documentation](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score)

🧐 But what about the model's mistakes? 🧐

🧐 What would the accuracy be for this example? 🧐

a

c

c

u

r

a

c

y

=

T

P

+

T

N

T

P

+

T

N

+

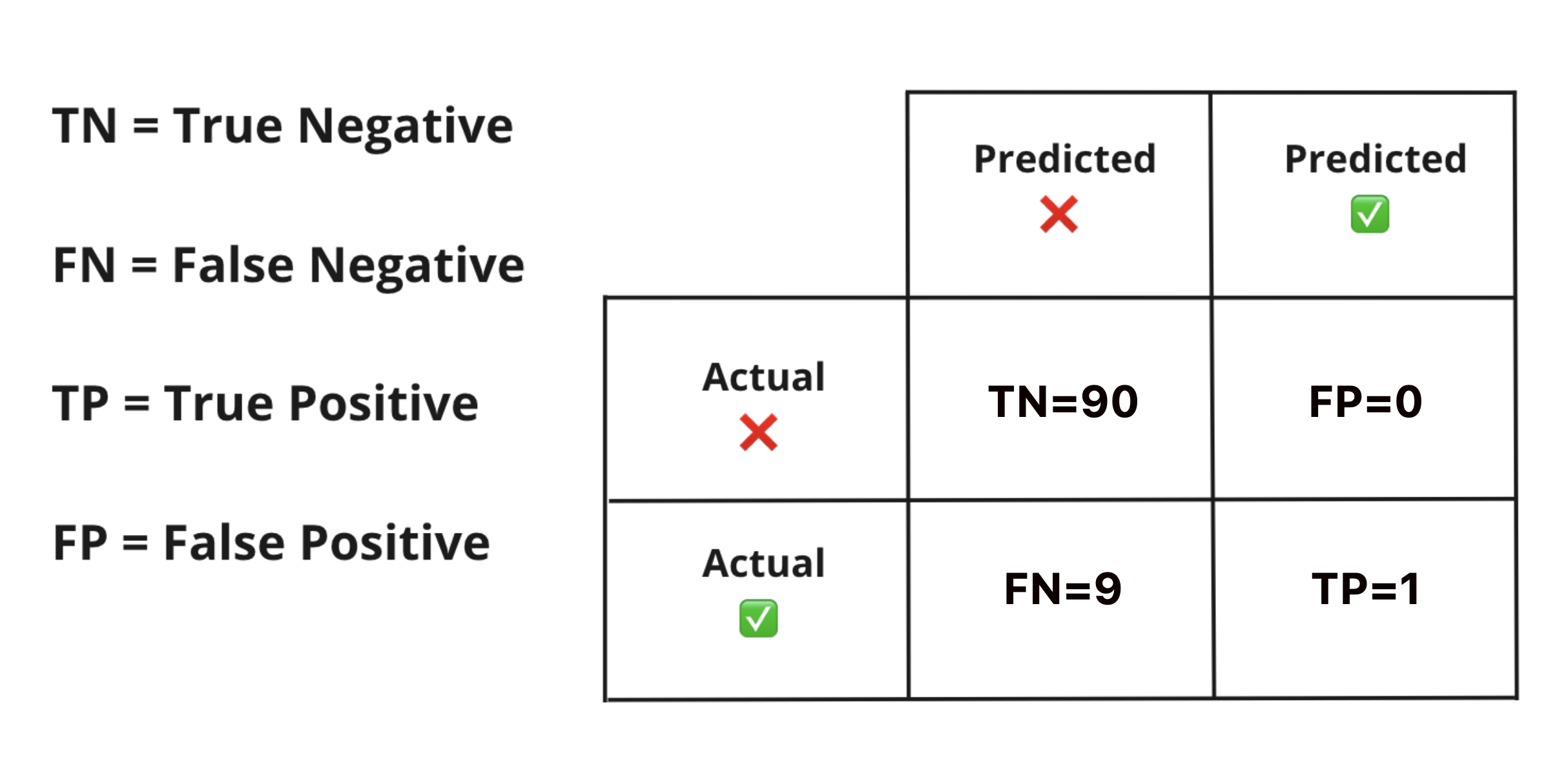
F

P

+

F

N



0.91

=

1

+

90

0

+

90

+

1

+

9

Anything strange about the performance?

🧐 Looks like our model almost always predicts ❌ 🧐

🚨 Using accuracy **alone** can give overly confident scores, especially when dealing with imbalanced datasets 🚨

👉 Use

a

c

c

u

r

a

c

y

when:

* Target classes are balanced
* Prediction of each class is equally important

## **Recall**

Measures the ability of the model to detect occurrences of a class.

r

e

c

a

l

l

=

T

P

T

P

+

F

N

0.8

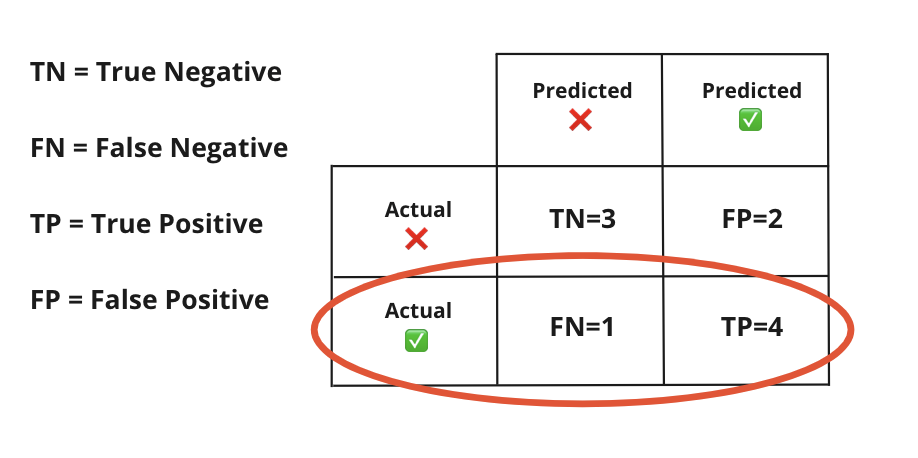
=

4

4

+

1



👉 [Sklearn's recall\_score documentation](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html#sklearn.metrics.recall_score)

Looking at the example from before, what is the

r

e

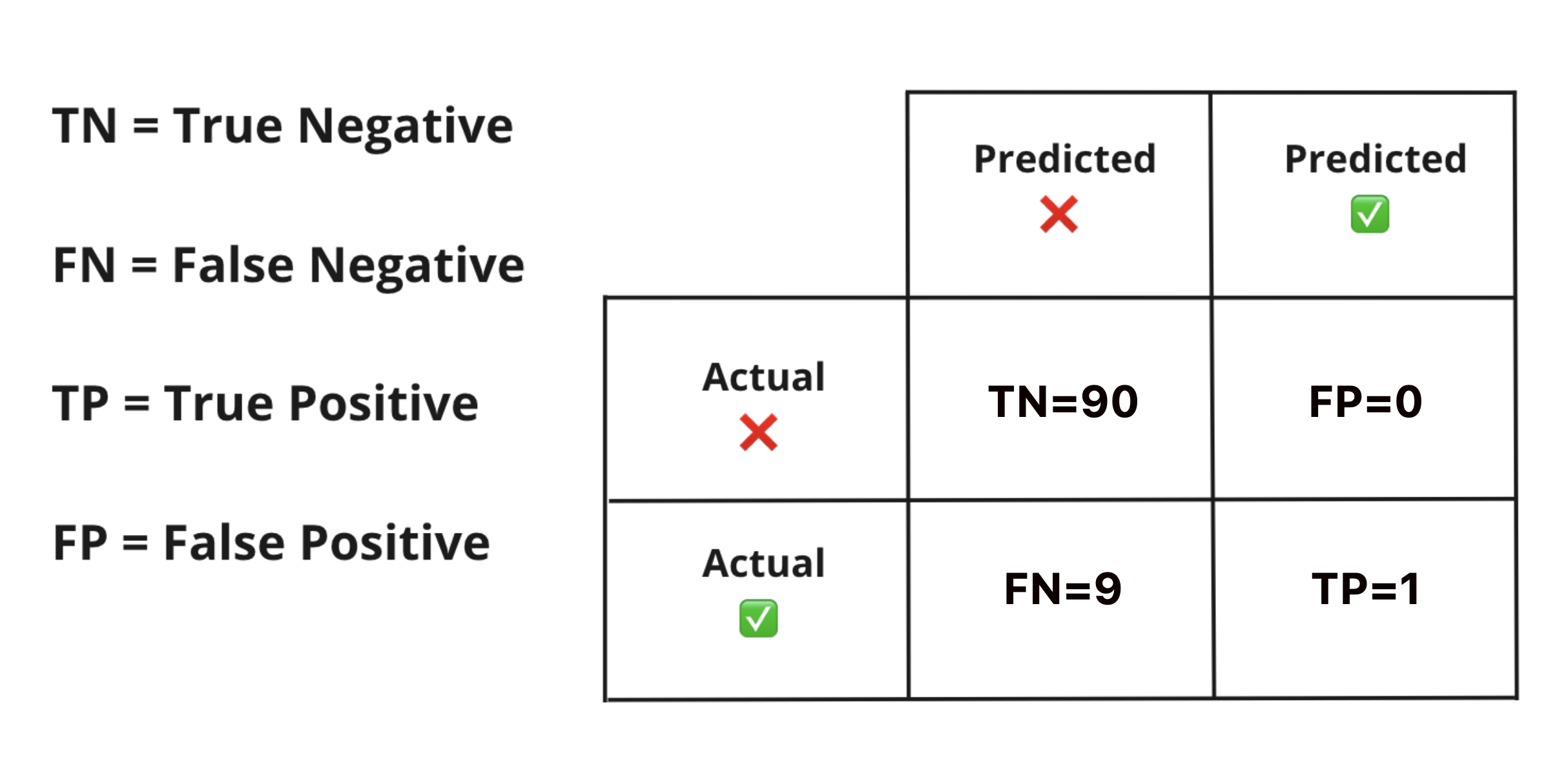
c

a

l

l

❓



0.1

=

1

9

+

1

👎 This model is really bad at identifying the Positive class.

👉 Use

r

e

c

a

l

l

when:

* It is important to identify as many occurrences of a class as possible, reducing false negatives but potentially increasing false positives
* You don't want to miss any positive classes
* E.g. Detecting fraudulent transactions, cases of a novel disease or potential sales leads

🧐 When **could** this metric alone be problematic? 🧐

r

e

c

a

l

l

=

T

P

T

P

+

F

N

0.9

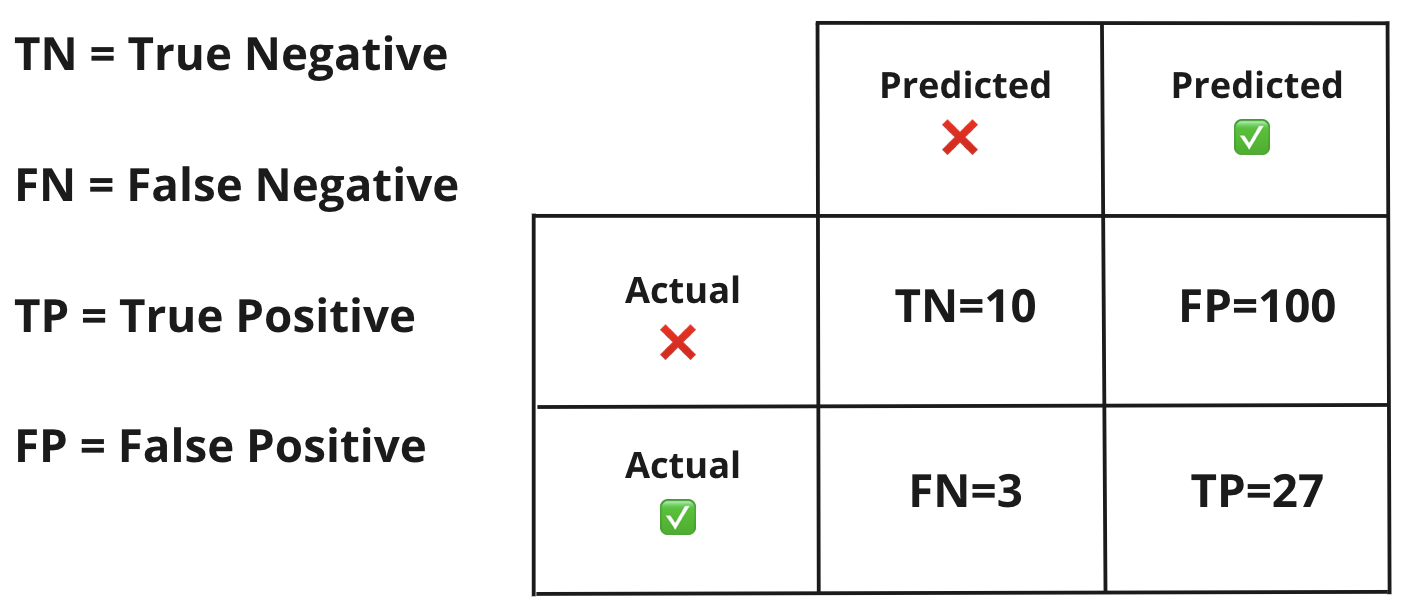
=

27

27

+

3



🧐 We could get too many false alarms 🧐

* These results could be great for identifying fraudulent payments at a bank
* But could also be terrible for a smoke alarm

## **Precision**

Measures the ability of a model to avoid false alarms for a class, or the confidence of a model when predicting a specific class.

p

r

e

c

i

s

i

o

n

=

T

P

T

P

+

F

P

0.67

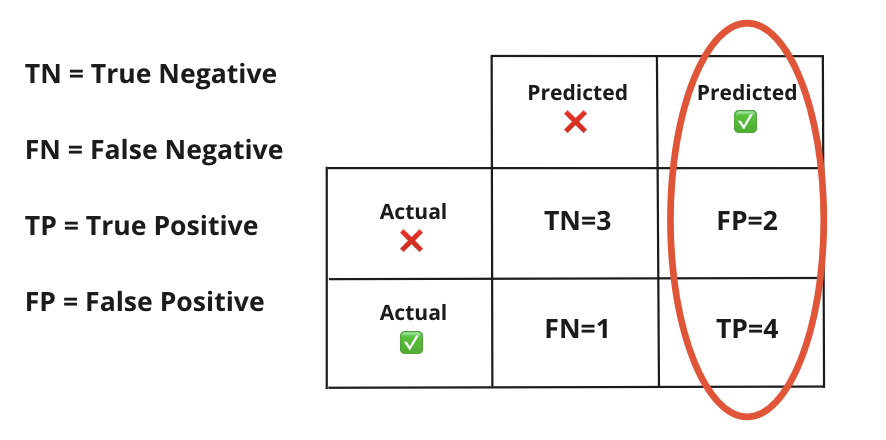
=

4

4

+

2



👉 [Sklearn's precision\_score documentation](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html#sklearn.metrics.precision_score)

👉 Use

p

r

e

c

i

s

i

o

n

when:

* It is important to be correct when identifying a class, reducing false positives but potentially increasing false negatives
* E.g. Targeted advertising, food and drug safety (you'd like the model to be *really confident*when classifying a drug as safe)

🧐 When could using precision alone be problematic? 🧐

p

r

e

c

i

s

i

o

n

=

T

P

T

P

+

F

P

0.89

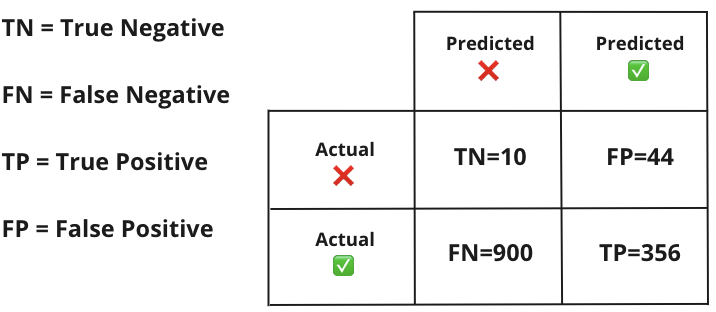
=

356

356

+

44



r

e

c

a

l

l

=

T

P

T

P

+

F

N

0.28

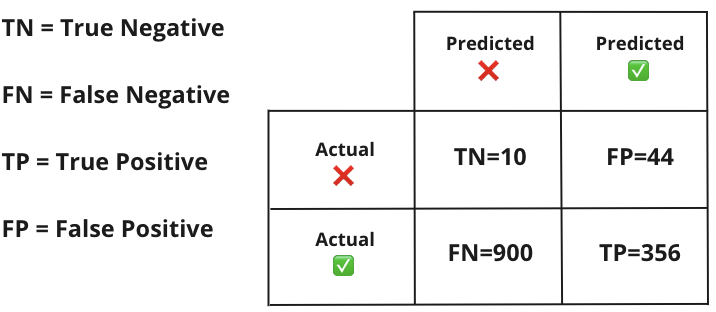
=

356

356

+

900



## F

## 1

## 

## s

## c

## o

## r

## e

A combination of precision and recall into a single metric.

F

1

=

2

⋅

p

r

e

c

i

s

i

o

n

×

r

e

c

a

l

l

p

r

e

c

i

s

i

o

n

+

r

e

c

a

l

l

0.73

=

2

⋅

0.67

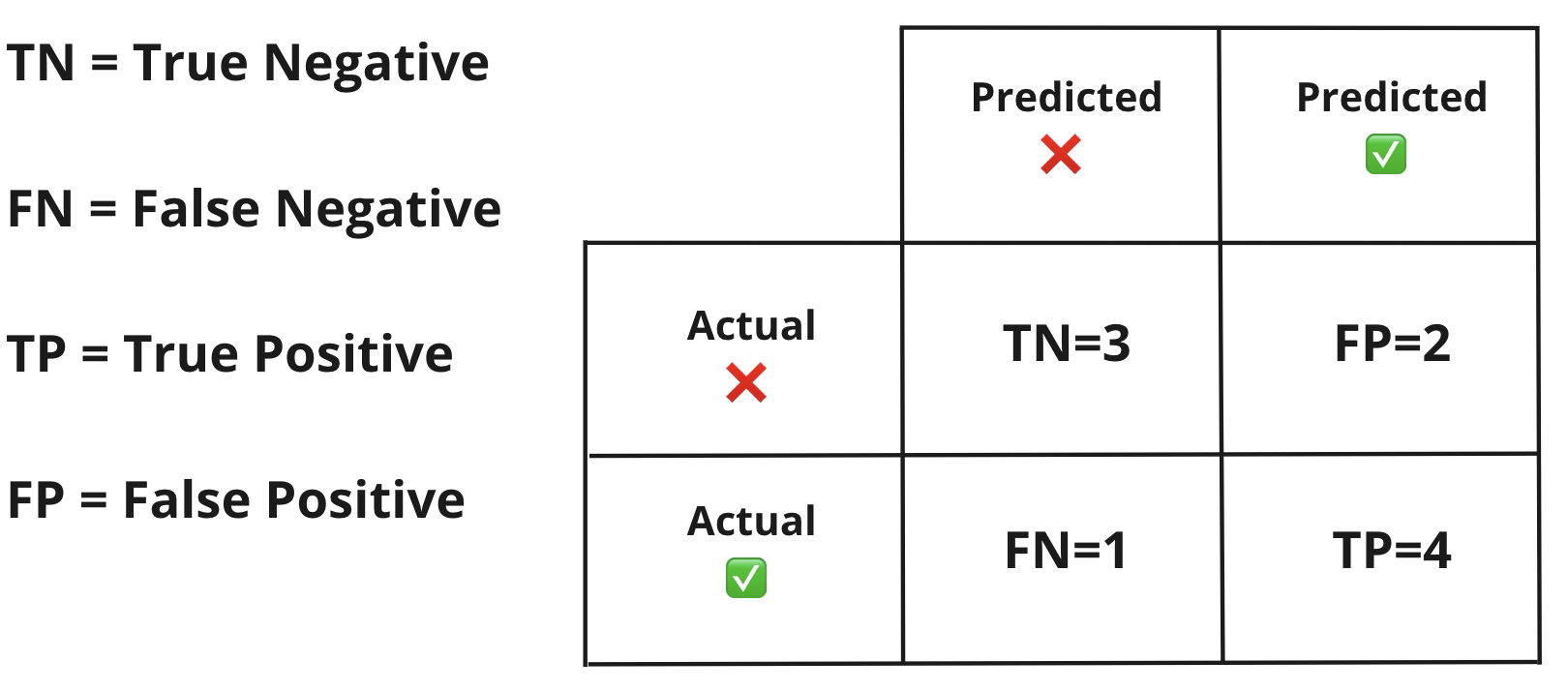
×

0.8

0.67

+

0.8



👉 [Sklearn's f1\_score documentation](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score)

* Also known as the harmonic mean of precision and recall
* It will be influenced more by the lower of the two values

👉 Use

F

1

s

c

o

r

e

when you want:

* A general metric to compare across models and datasets
* Combine the Precision/Recall tradeoff in a single metric

## **💻 Comparing metrics**

**from** **sklearn.metrics** **import** accuracy\_score, precision\_score, recall\_score, f1\_score

y\_true = [0, 1, 0, 0, 1, 0, 1, 1, 0, 1]

y\_pred = [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]

print('Accuracy =', round(accuracy\_score(y\_true, y\_pred), 2)) *# Accuracy*

print('Precision =', round(precision\_score(y\_true, y\_pred), 2)) *# Precision*

print('Recall =', round(recall\_score(y\_true, y\_pred), 2)) *# Recall*

print('F1 score =', round(f1\_score(y\_true, y\_pred), 2)) *# F1 score*

Accuracy = 0.7

Precision = 0.67

Recall = 0.8

F1 score = 0.73

## **Different applications, different metrics**

* We build models for specific tasks, each task will have a desired outcome
* We must pick a metric that suits the task

🧐 We're building a model to detect the safety of seatbelts. What metric should we optimize for? 🧐

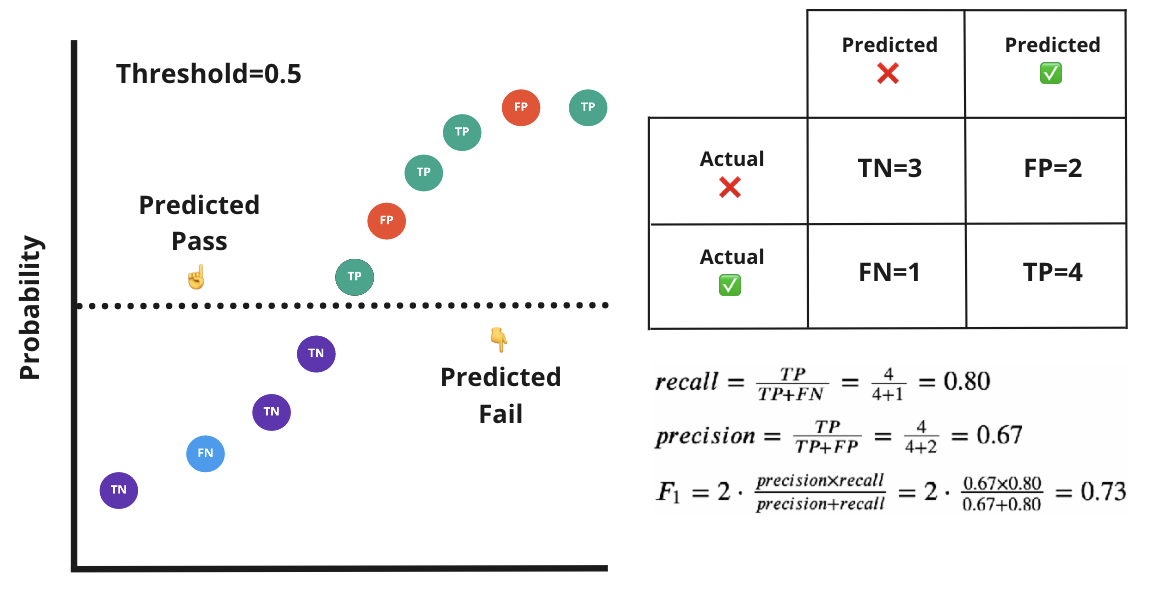
* Seat belt safe = 1
* Seat belt faulty = 0
* What's the cost of an incorrect prediction?
* Is it worse to release a faulty seatbelt, or to throw away a safe seatbelt?
* Optimize for **precision**
* We'd rather prevent injury (reduce false positives), and we're okay with the cost of throwing away a "maybe" safe seatbelt (false negatives)

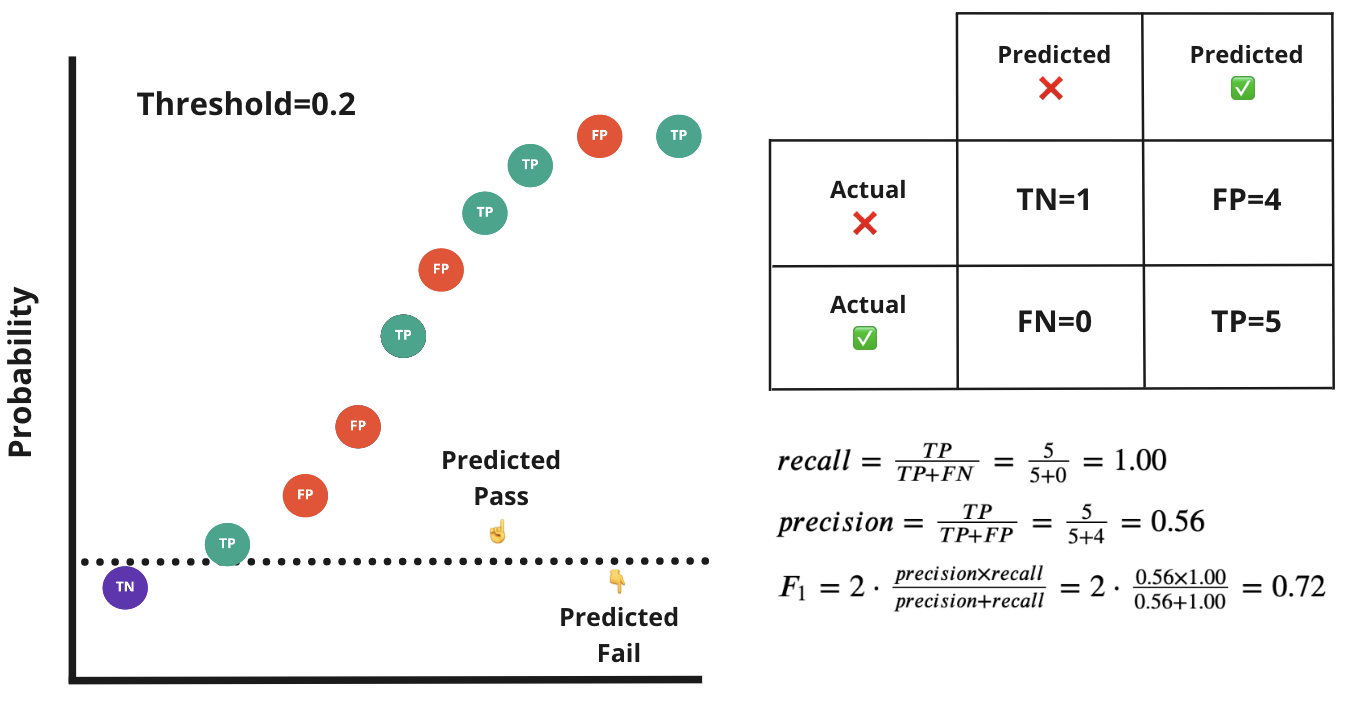
🧐 We can optimize for **a** metric by adjusting probability thresholds 🧐

## **The Precision-Recall Tradeoff**

* There is an inverse relationship between precision and recall
* Typically, we will trade one off against the other

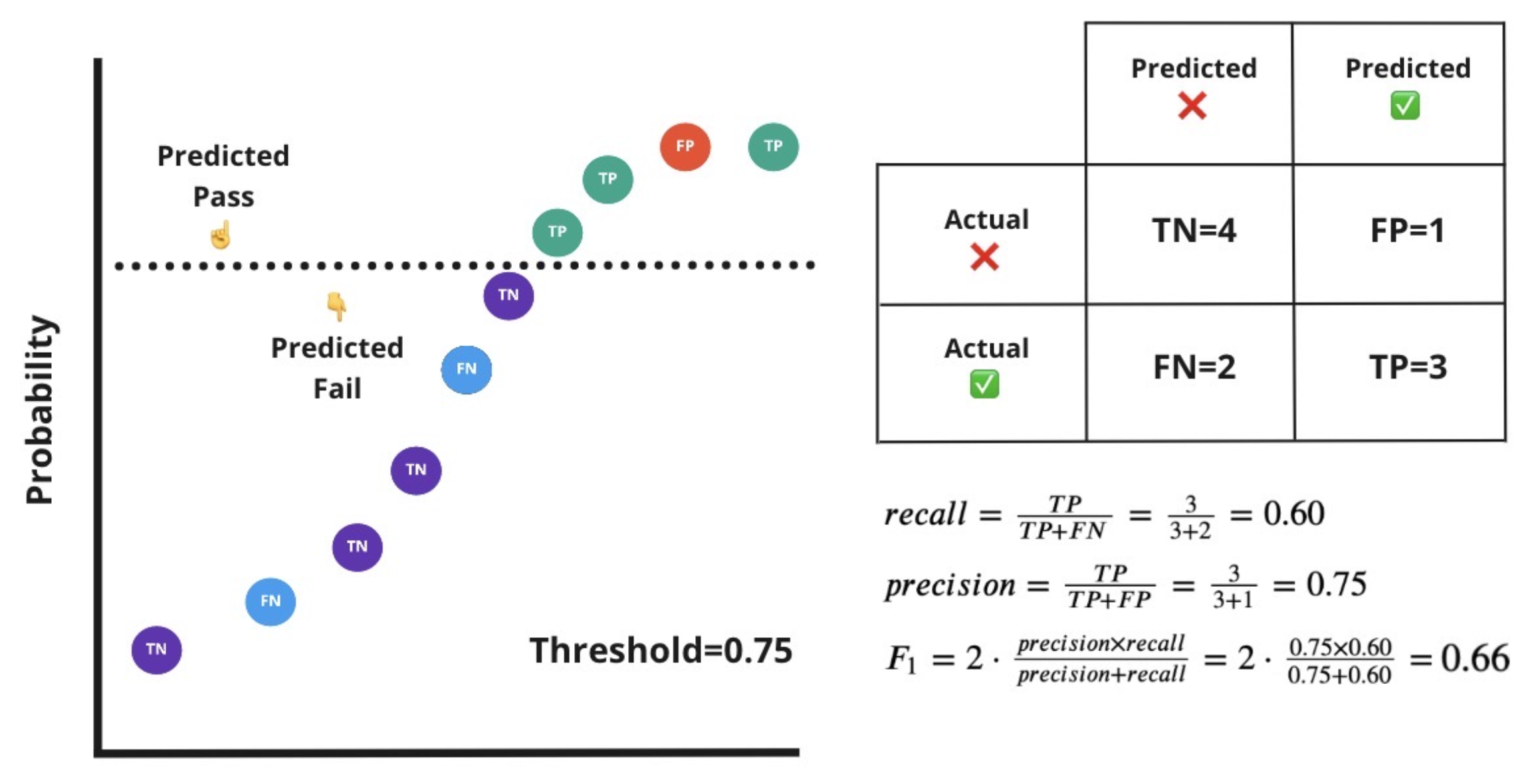
Let's go back to our exam predictions...





Recall has gone up

Precision has gone down



Precision has gone up

Recall has gone down

## **🖥 precision\_recall\_curve**

The precision\_recall\_curve lets us compare precision and recall across a variety of thresholds.

We can use this to find a threshold that guarantees a score for one metric whilst maintaining a minimum score for the other.

Let's put it into practice in a business application...

We are working at an insurance company trying to predict whether a policy will be expensive or not. Our boss has informed us that our previous model gave out cheaper than expected policies to customers, causing the company to lose money.

They want at least 80% of the **expensive policy customers to be identified**.

Since that's the class of focus, let's consider expensive policy holders as class 1

❓ Which metric should we optimize for?

❓ How would we adjust our thresholds for our next iteration?

ℹ️ We should optimize for

r

e

c

a

l

l

(Do you understand why? 🤔)

ℹ️ We should **lower** the threshold to be classified as a 1 (expensive policy holder)

🧐 Could there be any repercussions to this? 🧐

Customers who would normally get cheap policies, may get mis-sold expensive policies!

First, let's encode the target into 0 and 1.

❓ Which sklearn encoder you would you use in this case?

**from** **sklearn.preprocessing** **import** LabelEncoder

le = LabelEncoder()

le.fit(data['price\_range'])

print(le.classes\_) *# to check the order of classes that will be encoded*

data['price\_range\_encoded'] = le.transform(data['price\_range'])

data[['price\_range', 'price\_range\_encoded']].head()

['cheap' 'expensive']

|  | **price\_range** | **price\_range\_encoded** |
| --- | --- | --- |
| **0** | expensive | 1 |
| **1** | cheap | 0 |
| **2** | cheap | 0 |
| **3** | expensive | 1 |
| **4** | cheap | 0 |

**from** **sklearn.linear\_model** **import** LogisticRegression

**from** **sklearn.model\_selection** **import** cross\_val\_predict

**from** **sklearn.metrics** **import** precision\_recall\_curve

**import** **matplotlib.pyplot** **as** **plt**

model = LogisticRegression()

*# Predict class probabilties*

data['proba\_cheap'], data['proba\_expensive'] =cross\_val\_predict(model,

X,

data['price\_range\_encoded'],

cv=5,

method='predict\_proba').T

*# precision recall data*

precision, recall, threshold = precision\_recall\_curve(data['price\_range\_encoded'],

data['proba\_expensive'])

print(f'precision- **{**precision[:5]**}**')

print(f'recall- **{**recall[:5]**}**')

print(f'threshold- **{**threshold[:5]**}**')

precision- [0.31390135 0.31413613 0.31437126 0.31460674 0.31484258]

recall- [1. 1. 1. 1. 1.]

threshold- [0.0234146 0.02359012 0.02403223 0.02427382 0.02464116]

scores = pd.DataFrame({'threshold':threshold,

'precision': precision[:-1],

'recall':recall[:-1]}) *# Store in a dataframe*

scores

|  | **threshold** | **precision** | **recall** |
| --- | --- | --- | --- |
| **0** | 0.023415 | 0.313901 | 1.000000 |
| **1** | 0.023590 | 0.314136 | 1.000000 |
| **2** | 0.024032 | 0.314371 | 1.000000 |
| **3** | 0.024274 | 0.314607 | 1.000000 |
| **4** | 0.024641 | 0.314843 | 1.000000 |
| **...** | ... | ... | ... |
| **1332** | 0.995472 | 1.000000 | 0.011905 |
| **1333** | 0.995481 | 1.000000 | 0.009524 |
| **1334** | 0.995697 | 1.000000 | 0.007143 |
| **1335** | 0.995876 | 1.000000 | 0.004762 |
| **1336** | 0.996311 | 1.000000 | 0.002381 |

1337 rows × 3 columns

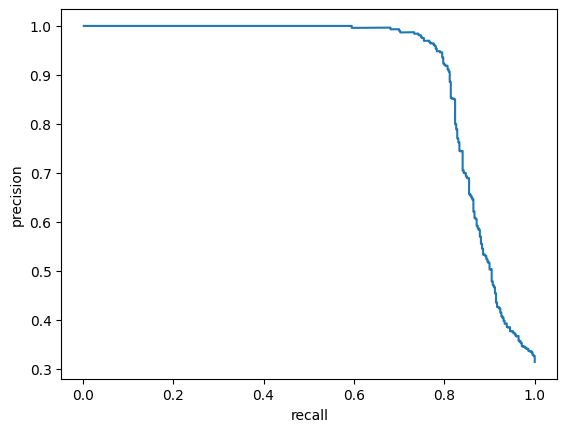
## **Plot the tradeoff**

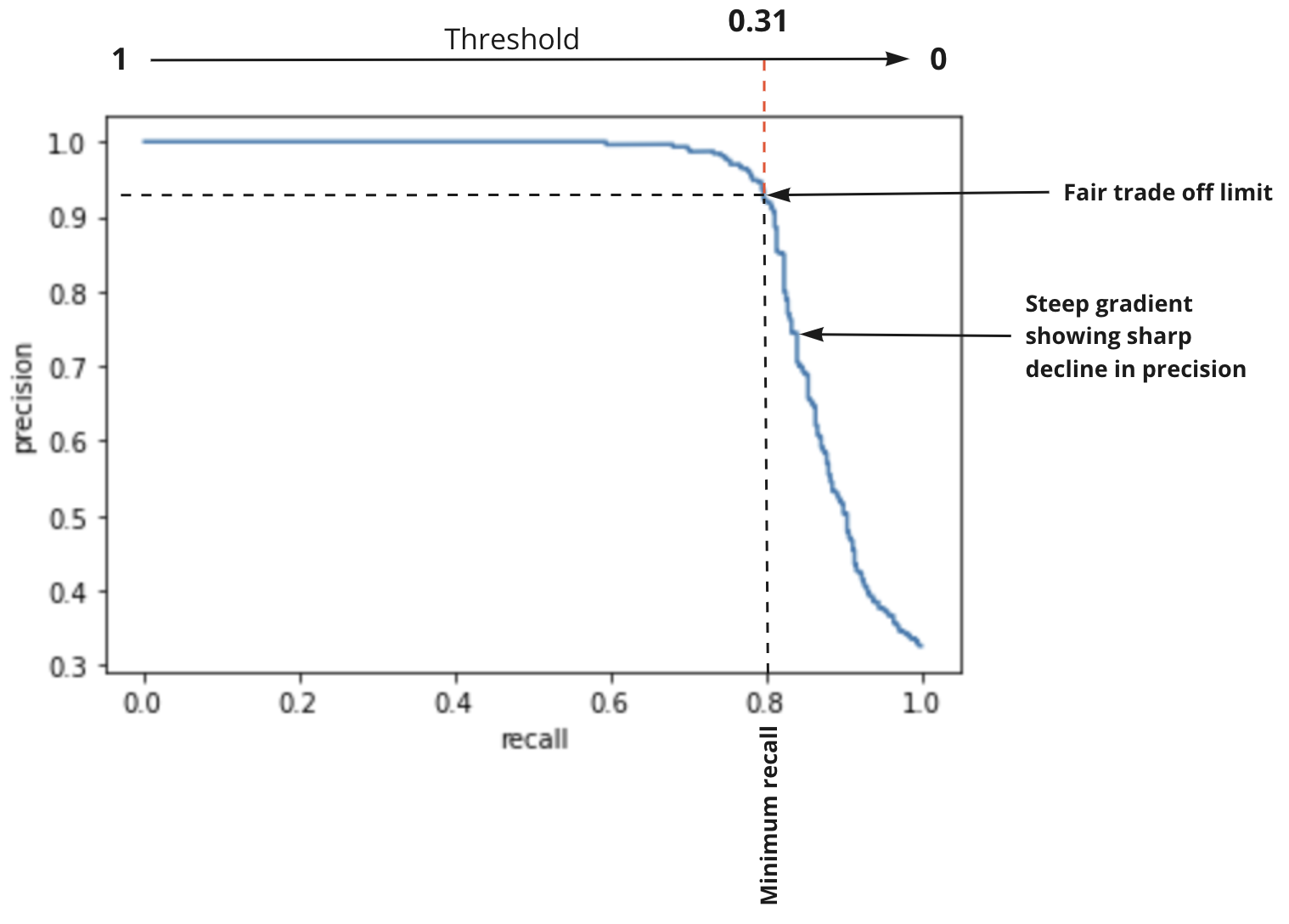
plt.plot(scores['recall'],scores['precision'])

plt.ylabel('precision')

plt.xlabel('recall')

Text(0.5, 0, 'recall')





#### **Find the threshold that guarantees a 0.8 recall score**

scores[scores['recall'] >= 0.8].threshold.max()

0.3055393000256149

We'll now update our threshold for our model using a custom prediction wrapper

model = LogisticRegression()

model.fit(X, data['price\_range\_encoded'])

**def** custom\_predict(X, custom\_threshold):

probs = model.predict\_proba(X) *# Get likelihood of each sample being classified as 0 or 1*

expensive\_probs = probs[:, 1] *# Only keep expensive likelihoods (1)*

**return** (expensive\_probs > custom\_threshold).astype(int) *# Boolean outcome converted to 0 or 1*

updated\_preds = custom\_predict(X=X, custom\_threshold=0.305539) *# Update predictions*

print("Recall:", recall\_score(data['price\_range\_encoded'], updated\_preds)) *# Rerun recall*

print("Precision:", precision\_score(data['price\_range\_encoded'], updated\_preds)) *# Rerun precision*

print("F1 Score:", f1\_score(data['price\_range\_encoded'], updated\_preds)) *# Rerun f1*

Recall: 0.8071428571428572

Precision: 0.9287671232876712

F1 Score: 0.8636942675159236

We've identifed that the optimum threshold for the task requiring 80% recall was ~0.31 and updated our decision thresholds accordingly 💪

## **ROC-AUC (Receiver Operating Characteristic-Area Under Curve)**

Similar to the Precision-Recall curve, the ROC curve summarizes the ability of the model to trade-off between two metrics across different thresholds.

T

r

u

e

p

o

s

i

t

i

v

e

r

a

t

e

=

s

e

n

s

i

t

i

v

i

t

y

=

r

e

c

a

l

l

=

T

P

T

P

+

F

N

F

a

l

s

e

p

o

s

i

t

i

v

e

r

a

t

e

=

(

1

−

s

p

e

c

i

f

i

c

i

t

y

)

=

F

P

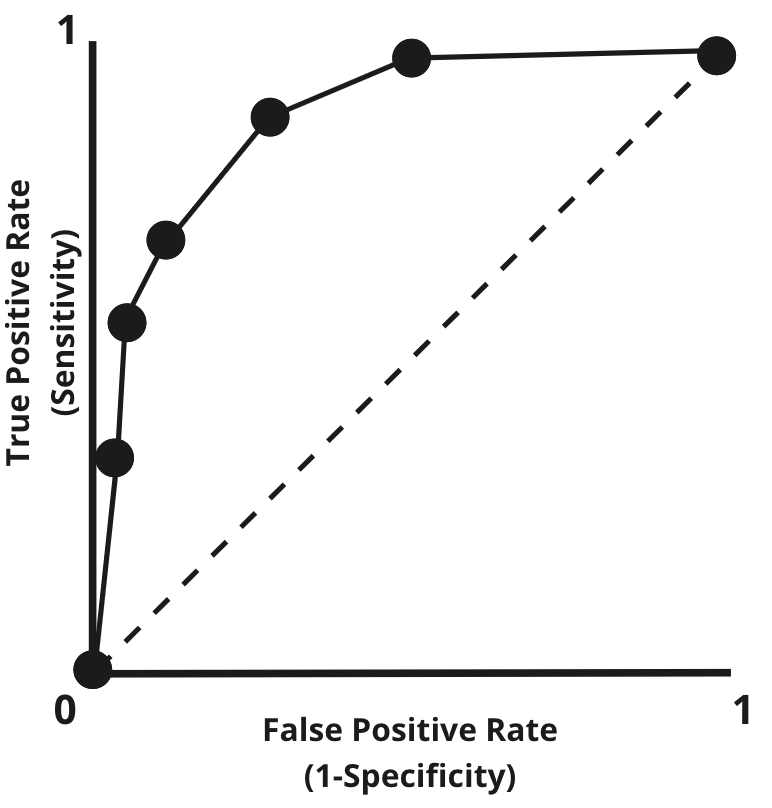
F

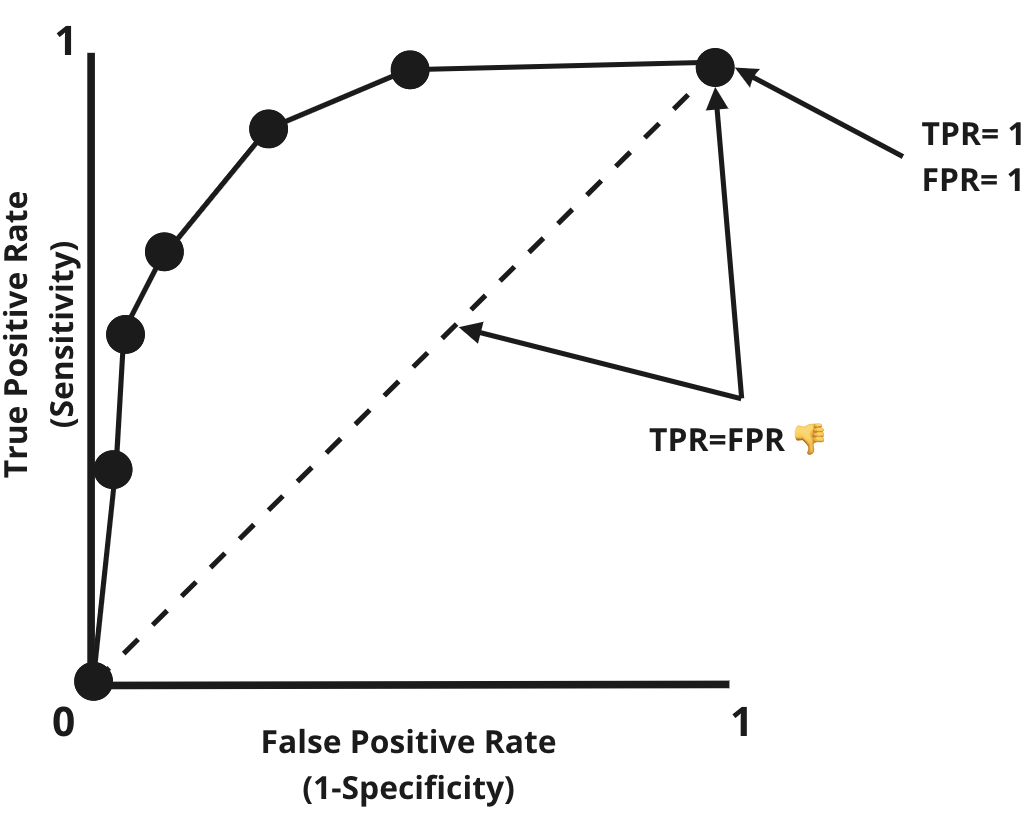
P

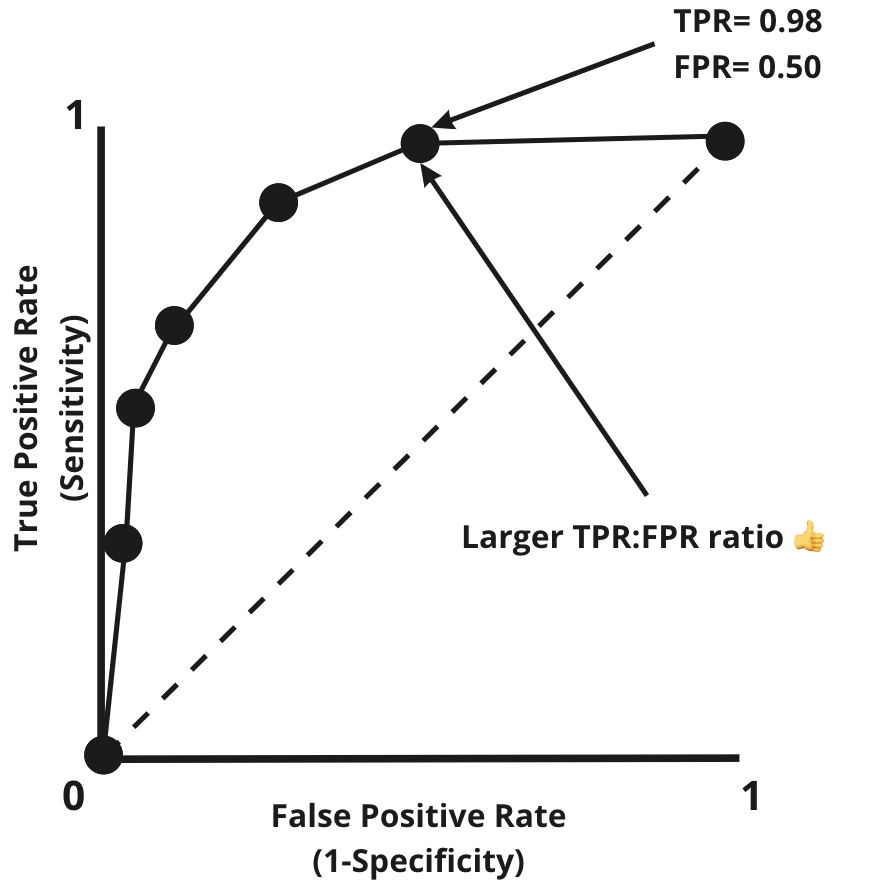
+

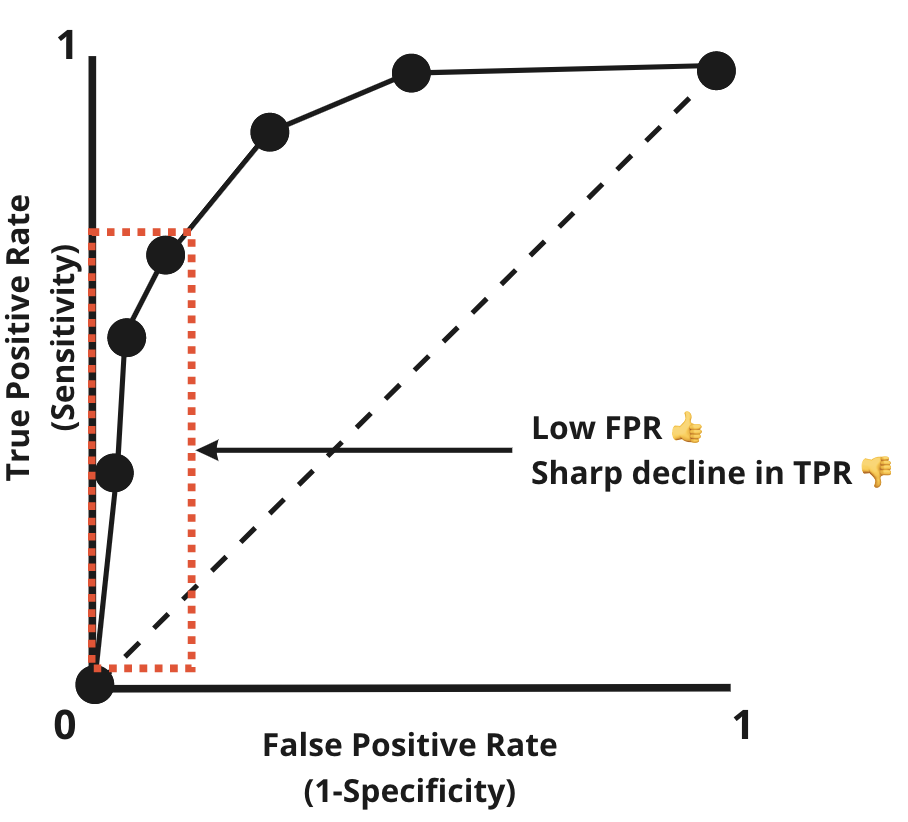
T

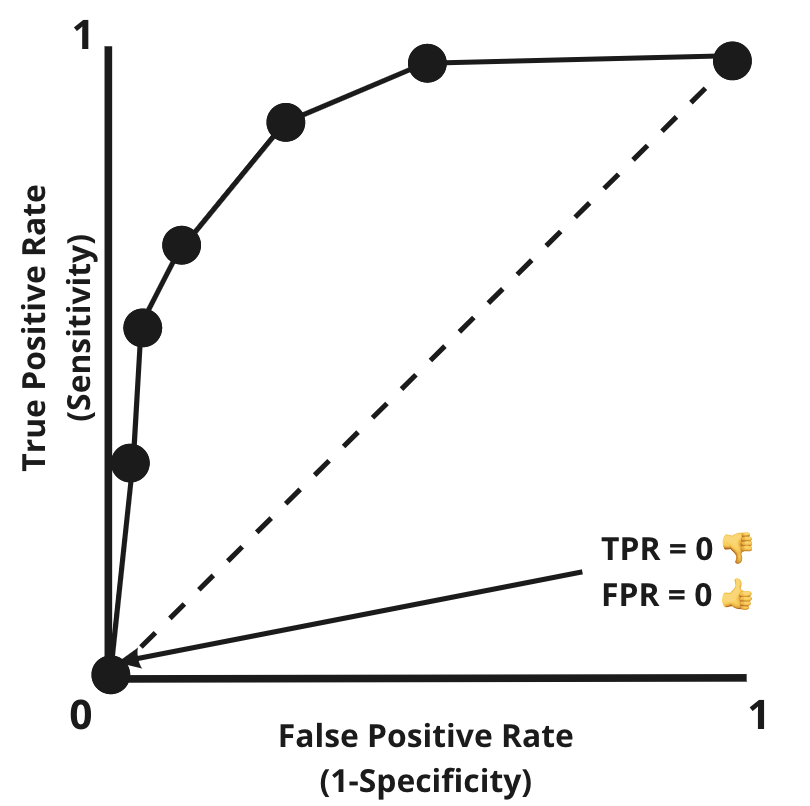
N





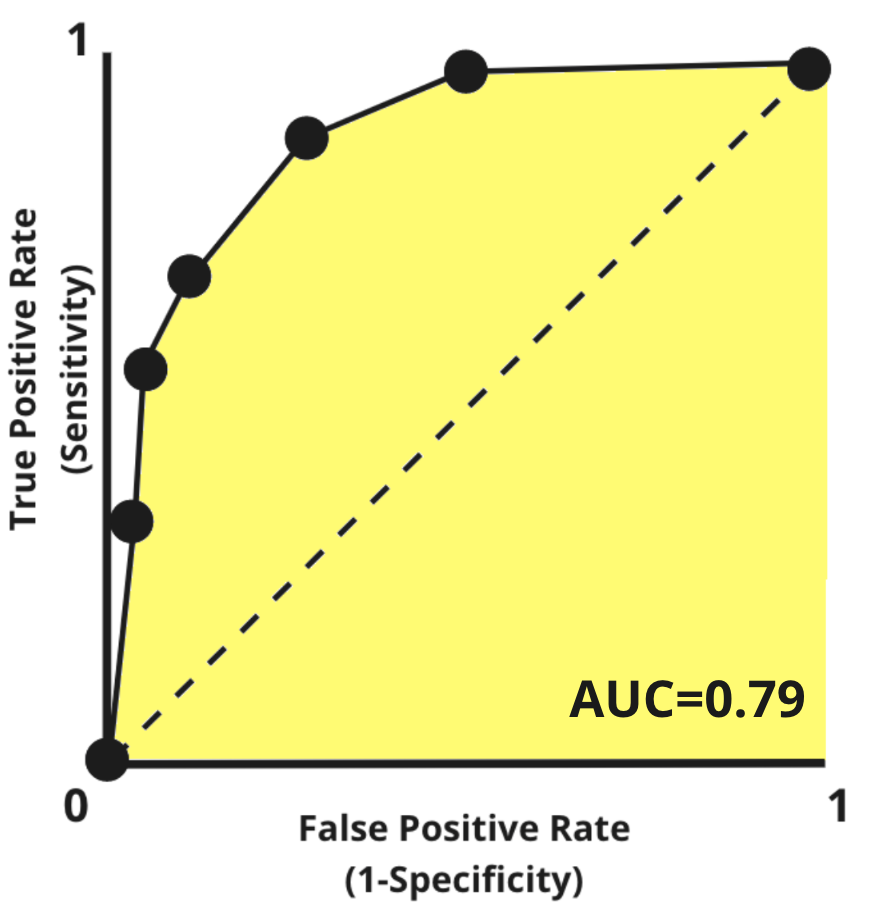




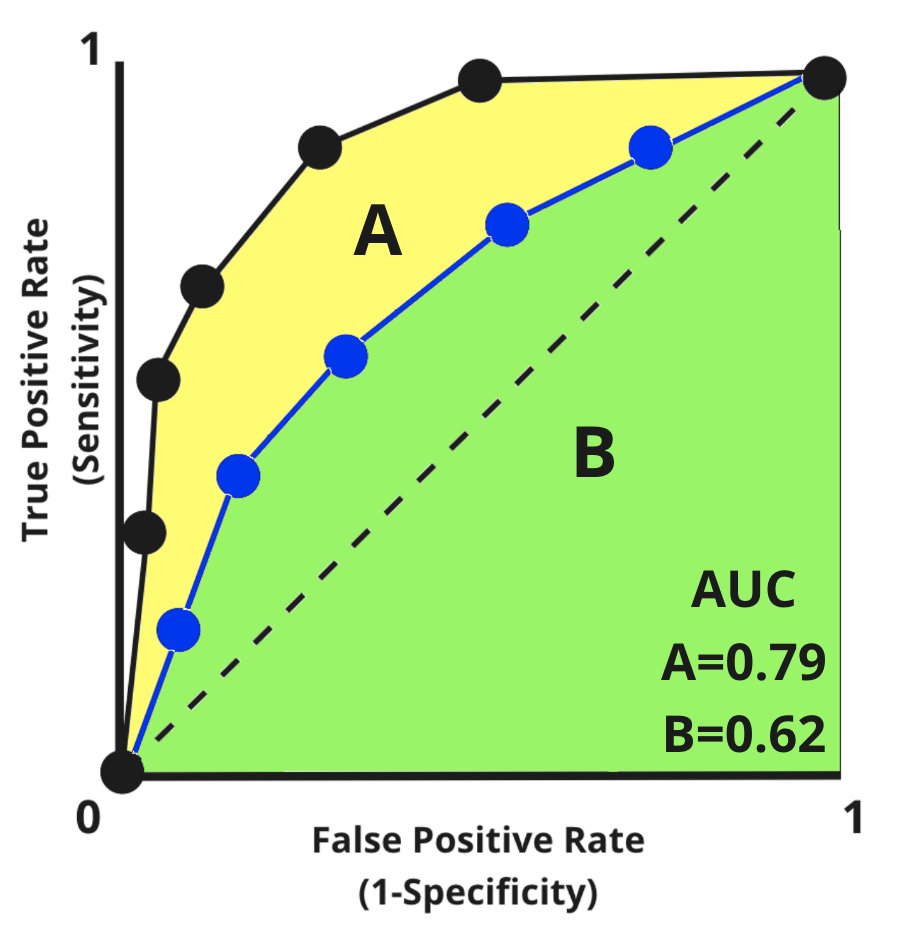


### **AUC (Area Under Curve)**

The AUC gives an indication of overall model performance.



### **Comparing models**

The larger the AUC, the greater the overall general performance.

* Unlike F1, recall, precision and accuracy- AUC is not dependent on a chosen threshold
* This makes it a great metric for measuring a model's general performance 💪

### **In Sklearn...**

**from** **sklearn.metrics** **import** roc\_curve

**from** **sklearn.metrics** **import** roc\_auc\_score

*# Extract associated metrics and thresholds*

fpr, tpr, thresholds = roc\_curve(data['price\_range\_encoded'], data['proba\_expensive'])

*# Compute AUC score*

auc\_score = roc\_auc\_score(data['price\_range\_encoded'], data['proba\_expensive'])

auc\_score

0.9090388007054675

👉 [Sklearn roc\_auc\_score documentation](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score)

👉 [Sklearn roc\_curve documentation](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve)

### **Summary - Classification Metrics**

| **Metric** | **Use when...** |
| --- | --- |
| Accuracy | You want to know if your model predicts well overall, without focus on a particular class |
| Recall | You want to capture as many occurrences of a class, even if you capture wrong ones too |
| Precision | You want to be really sure that you predicted the right class when you predict it |
| F1 Score | You want a harmonic measurement of precision and recall; good for comparing across models and datasets |
| ROC-AUC Score | General model robustness metric, showing how well the model is able to distinguish between two classes across all thresholds. |

# **6. Error analysis**

### **Error analysis**

An iterative process for identifying common themes within our model's mistakes

* Do specific cohorts within our X data perform better or worse than others?
* Is one class consistently identified better than others?
* Are some errors so large they drag overall performance down?

These questions can lead you to more data collection or enhanced feature engineering.

🚨 Remember, each metric tells a **specific** story, but not the **full** story 🚨

## **Your turn! 🚀**