🤖 Machine Learning

Plan of the module

1. Fundamentals of Machine Learning
2. Data preparation
3. Performance metrics
4. Under the hood
5. Model tuning
6. Workflow
7. Ensemble methods
8. Unsupervised learning
9. Time series
10. Natural Language Processing

Plan of the lecture

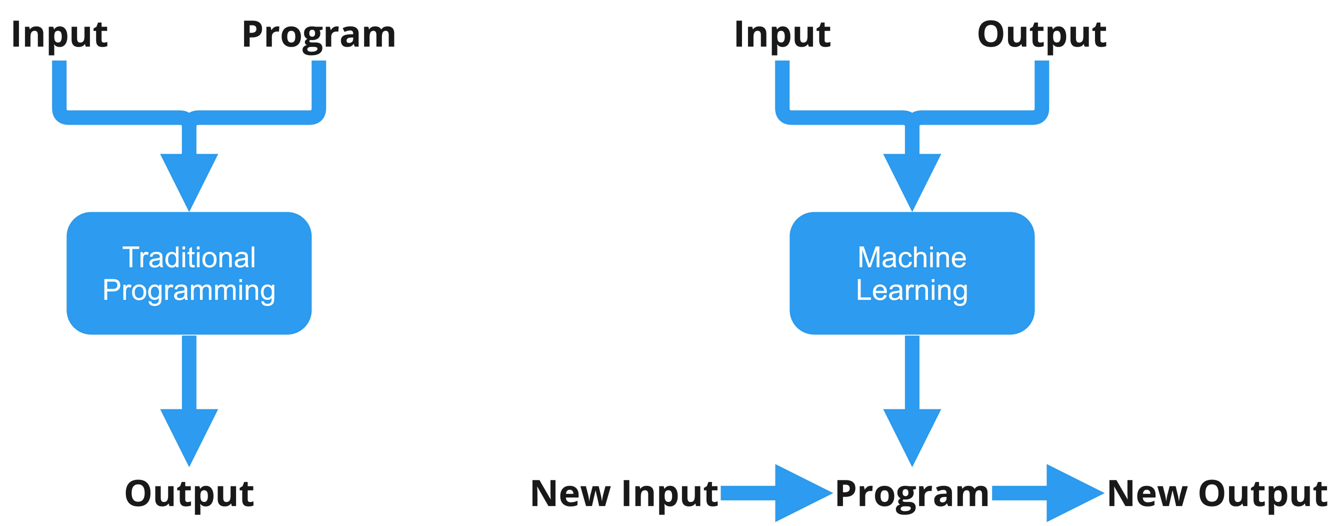
1. What is Machine Learning?
2. Scikit-learn Library
3. Linear Regression with Scikit
4. Generalization - Holdout Bias Variance
5. Cross Validation
6. Learning Curves

1. What is Machine Learning?

The area of computational science that focuses on analyzing and interpreting patterns and structures in data to enable learning, reasoning, and decision making outside of human interaction.

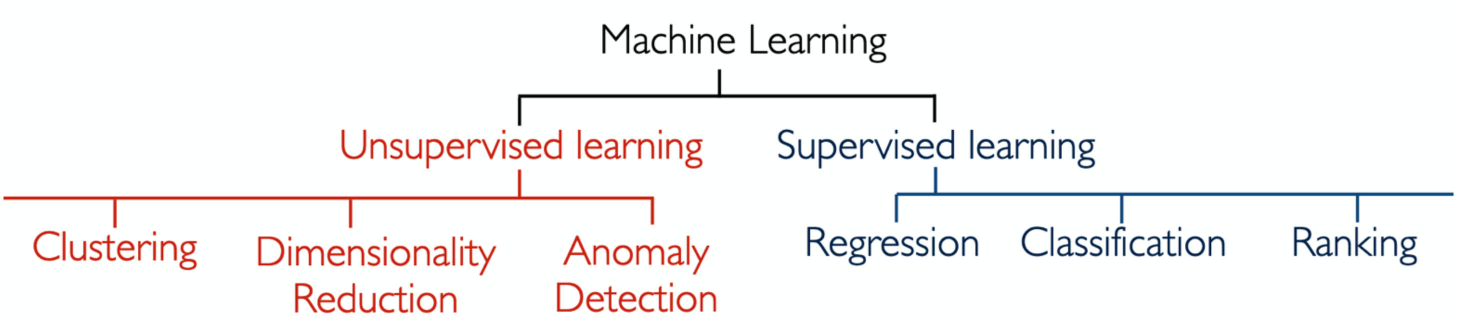
- Machine Learning is constantly and rapidly evolving
- Few standards are set in stone
- Stay tuned!

General programming vs. Machine Learning



[Original source (https://www.sciencedirect.com/science/article/abs/pii/S1878875017316650)](https://www.sciencedirect.com/science/article/abs/pii/S1878875017316650), recreated by Bruncky

Machine Learning Taxonomy

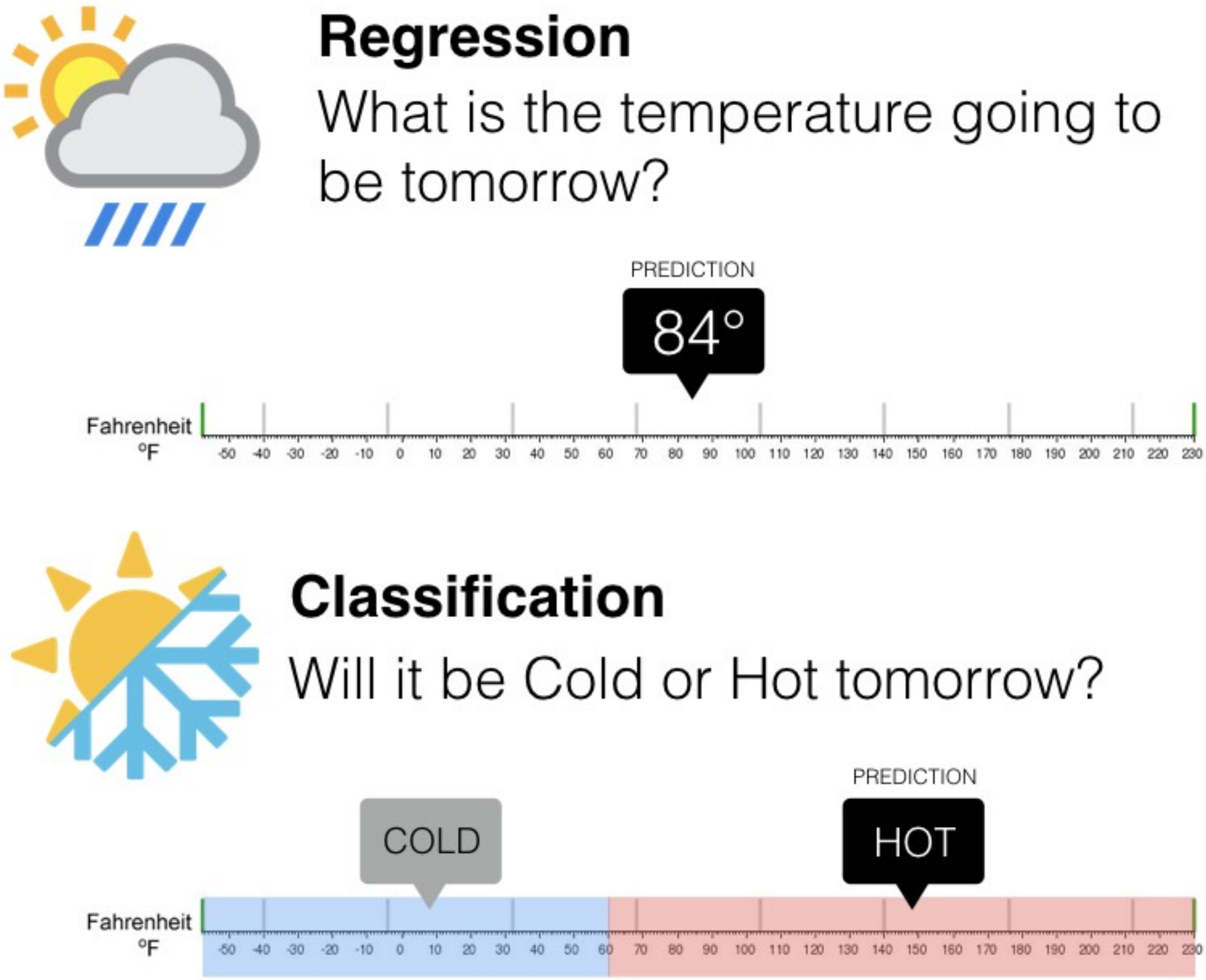


Supervised Learning

Develop predictive model based on both input and output data.

	features				target
	type (category)	# rooms (int)	surface (float m2)	public trans (boolean)	sold (float k€)
samples	Apartment	3	50	TRUE	450
	House	5	254	FALSE	430
	Duplex	4	68	TRUE	712
	Apartment	2	32	TRUE	234

Classification vs Regression



[Source (https://www.researchgate.net/figure/Example-of-classification-and-regression-problem-when-dealing-with-temperature_fig7_333221048)](https://www.researchgate.net/figure/Example-of-classification-and-regression-problem-when-dealing-with-temperature_fig7_333221048)

Unsupervised Learning

Group and interpret data based only on input data.

samples	features				target
	type (category)	# rooms (int)	surface (float m2)	public trans (boolean)	~~sold (float k$)~~
	Apartment	3	50	TRUE	~~450~~
	House	5	254	FALSE	~~430~~
	Duplex	4	68	TRUE	~~712~~
	Apartment	2	32	TRUE	~~234~~

Jargon

The **features** can also be referred to as the **input**, the **X's**, the **variables**.

The **target** can also be referred to as the **output**, the **y**, the **label**, the **class**.

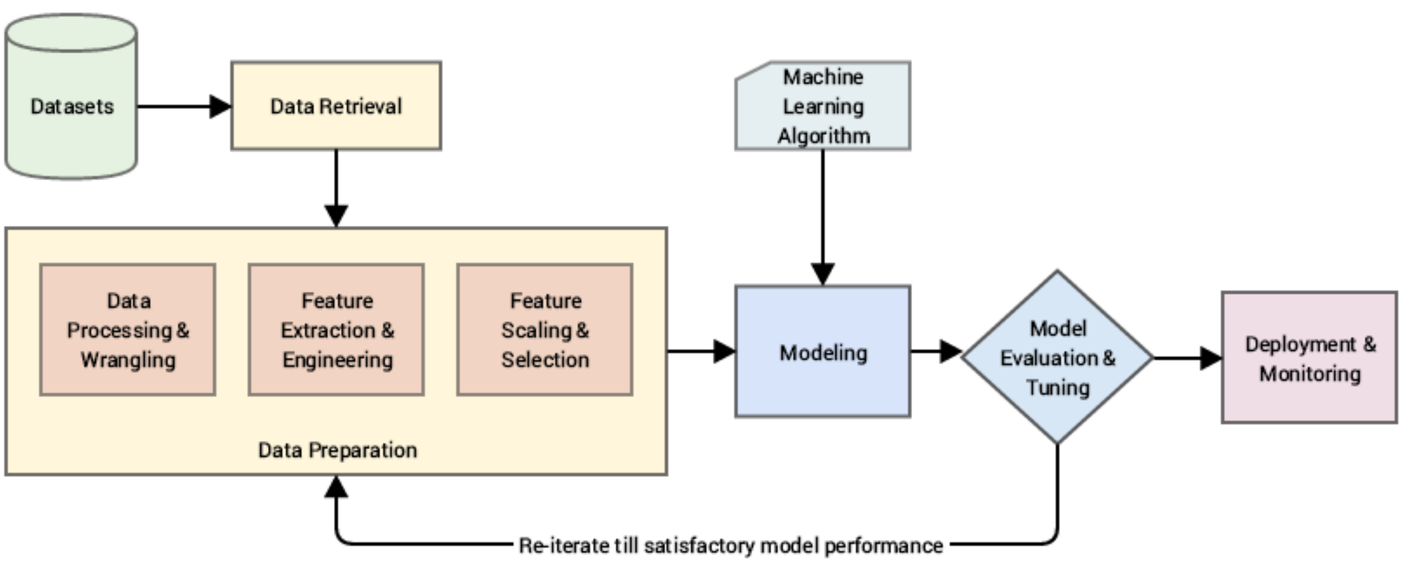
The **samples** can also be referred to as the **rows** or the **observations**

features | target

samples

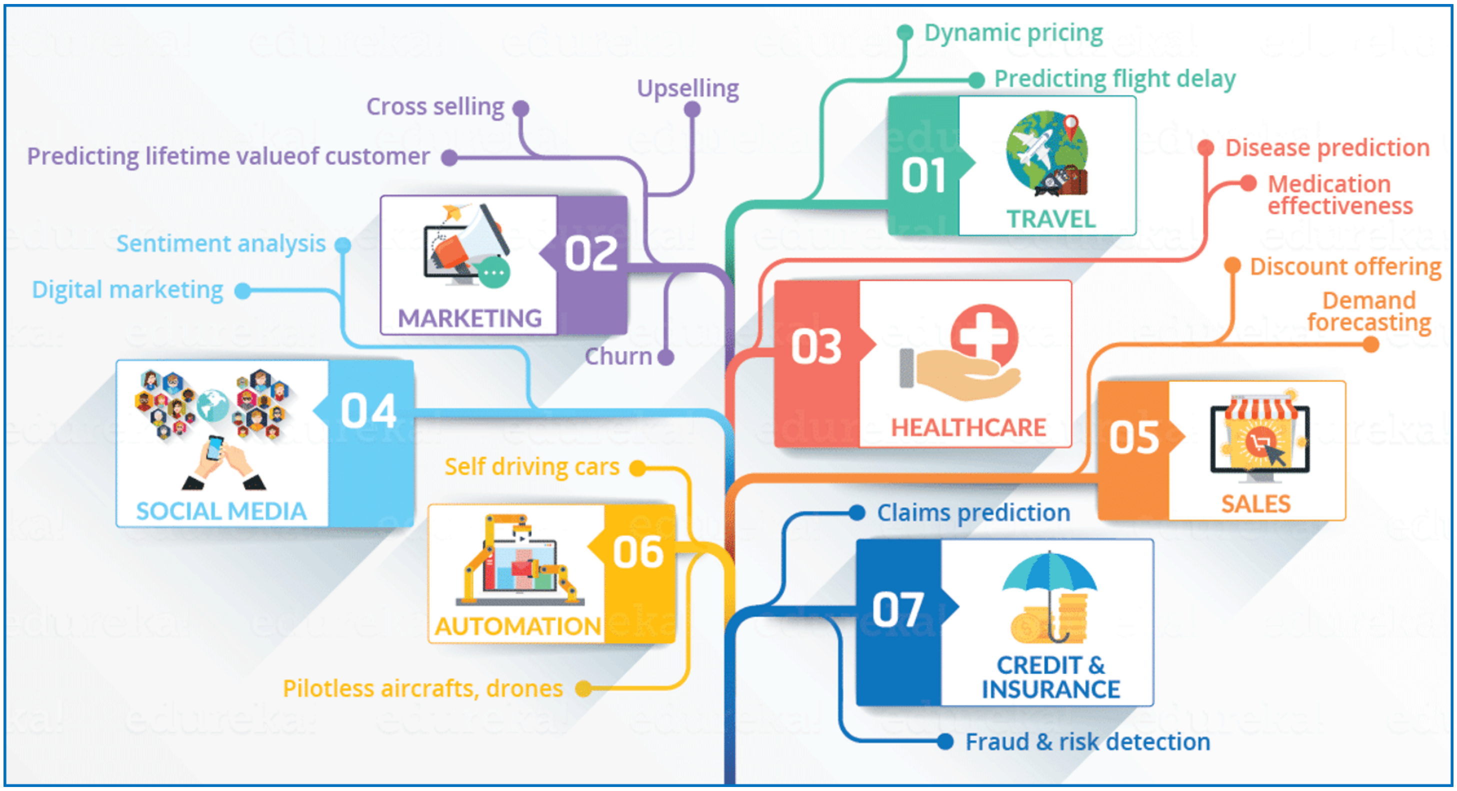
type (category)	# rooms (int)	surface (float m2)	public trans (boolean)	sold (float k€)
Apartment	3	50	TRUE	450
House	5	254	FALSE	430
Duplex	4	68	TRUE	712
Apartment	2	32	TRUE	234

ML stages



[Source (https://web2.qatar.cmu.edu/~gdicaro/15488/)](https://web2.qatar.cmu.edu/~gdicaro/15488/)

ML domains and tasks



[Source (https://www.edureka.co/blog/what-is-machine-learning/)](https://www.edureka.co/blog/what-is-machine-learning/)

2. Scikit-learn

Scikit-learn (Sklearn) is a Machine Learning library that provides data preprocessing, modeling, and model selection tools.

👉 [https://scikit-learn.org (https://scikit-learn.org)](https://scikit-learn.org)

Installing Sklearn

In your terminal, type the following:

```
pip install scikit-learn
```

👉 [Installation Documentation (https://scikit-learn.org/stable/install.html)](https://scikit-learn.org/stable/install.html)

Sklearn structure

- Sklearn is organized by **modules**
- Each module contains tools in the form of **classes**

`linear_model` module

- `linear_model` is a module
- `LinearRegression` is a class

👉 [Sklearn `linear_model` documentation (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model)](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model)

Module and Class imports

There are many ways to import modules and classes in notebooks, but there is a best practice.

```
🚫 import sklearn # this will not work with sklearn
model = sklearn.linear_model.LinearRegression() # ❌
```

```
🚫 import sklearn.linear_model # import of entire module
model = sklearn.linear_model.LinearRegression() # must type library and mo
dule prefix every time
```

```
🚫 from sklearn import linear_model # import of entire module
model = linear_model.LinearRegression() # must type module prefix every ti
me
```

```
🚫 from sklearn.linear_model import * # import of entire module
model = LinearRegression()
```

"Explicit is better than implicit" - The Zen of Python

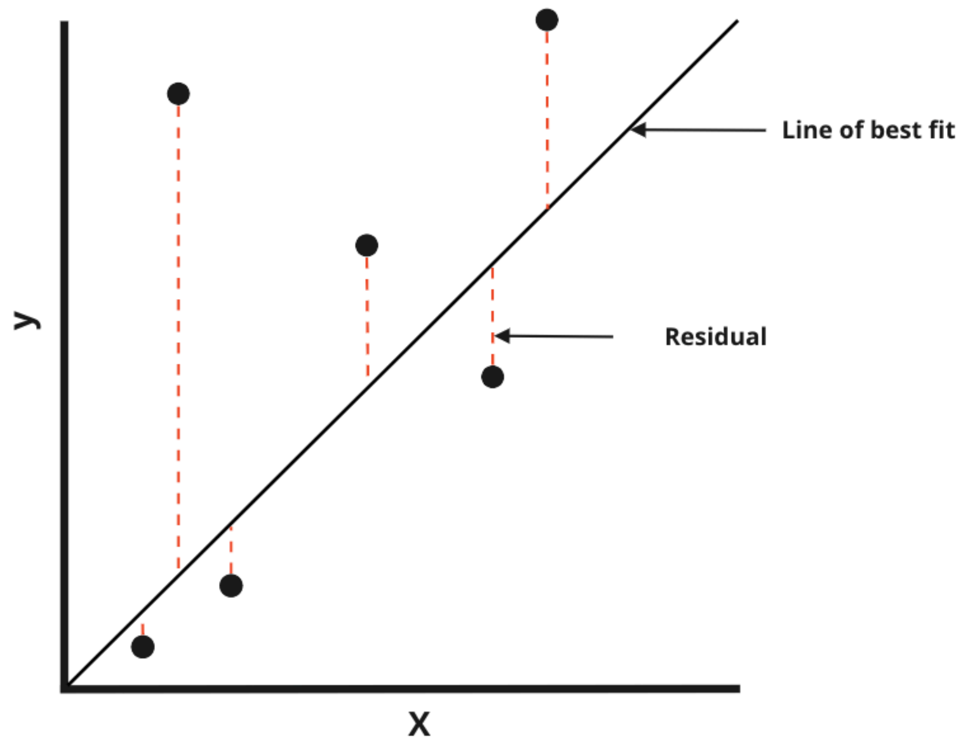
```
✅ from sklearn.linear_model import LinearRegression # explicit class imp
ort from module
model = LinearRegression() #=> we know where this object comes from
```

3. Linear Modeling with Sklearn

Linear Regression Recap

A Linear Regression (OLS) maps a linear relationship between the input `X` and the output `y`. It optimizes slope `a` and intercept `b` by reducing the residuals between the actual `y` and the predicted `y`.

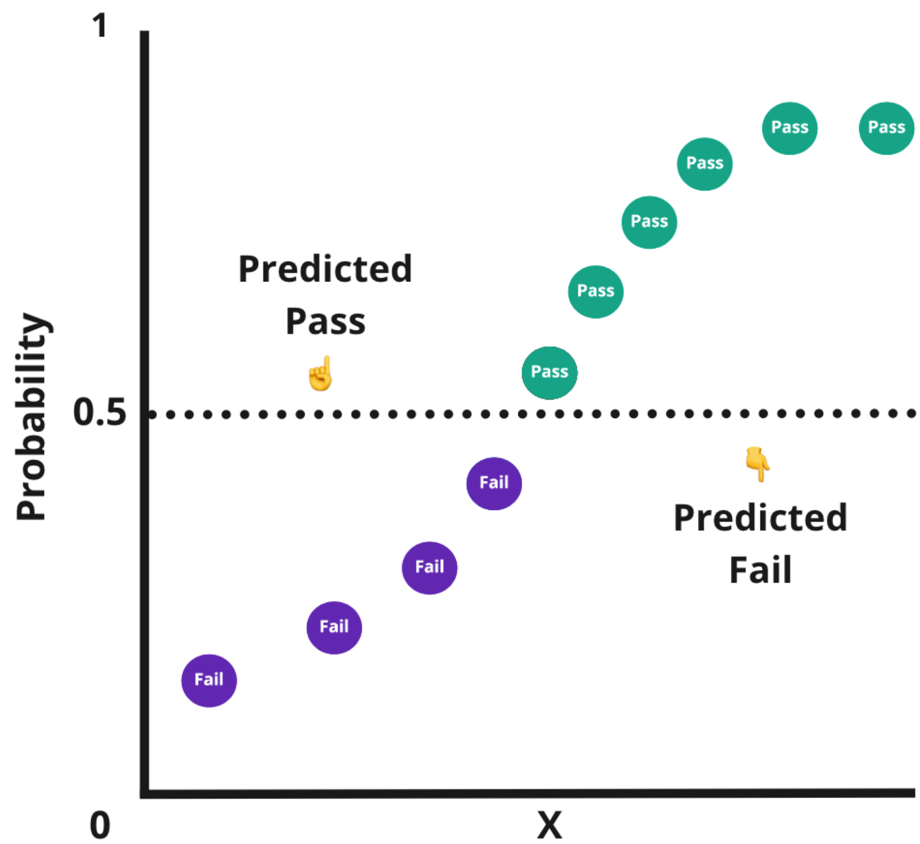
$y = aX + b$



Logistic Regression recap

Despite having "regression" in its name, Logistic Regression is actually a classifier. It uses a Sigmoid Function to map the probability of belonging to a class.

$P(y_i = 1 \mid X) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$



💻 Linear Regression with Sklearn

Consider the following dataset ([download here (https://wagon-public-datasets.s3.amazonaws.com/Machine%20Learning%20Datasets/ML_Houses_dataset.csv)](https://wagon-public-datasets.s3.amazonaws.com/Machine%20Learning%20Datasets/ML_Houses_dataset.csv)). It is a collection of houses and their characteristics, along with their sale price. The full documentation of the dataset is available [here (https://wagon-public-datasets.s3.amazonaws.com/Machine%20Learning%20Datasets/ML_Houses_dataset_description.txt)](https://wagon-public-datasets.s3.amazonaws.com/Machine%20Learning%20Datasets/ML_Houses_dataset_description.txt).

In [ ]:
```
import pandas as pd

data = pd.read_csv(file)
# Shuffling the data
data = data.sample(frac=1)
```

In [ ]:
```
data.head()
```

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCont
1557	358	120	RM	44.0	4224	Pave	NaN	Reg	
1004	1005	120	RL	43.0	3182	Pave	NaN	Reg	
1518	179	20	RL	63.0	17423	Pave	NaN	IR1	
348	349	160	RL	36.0	2448	Pave	NaN	Reg	
794	795	60	RL	NaN	10832	Pave	NaN	IR1	

5 rows × 85 columns

Let's start simple by modeling the `SalePrice` (`y`) according to the `GrLivArea` (`X`).

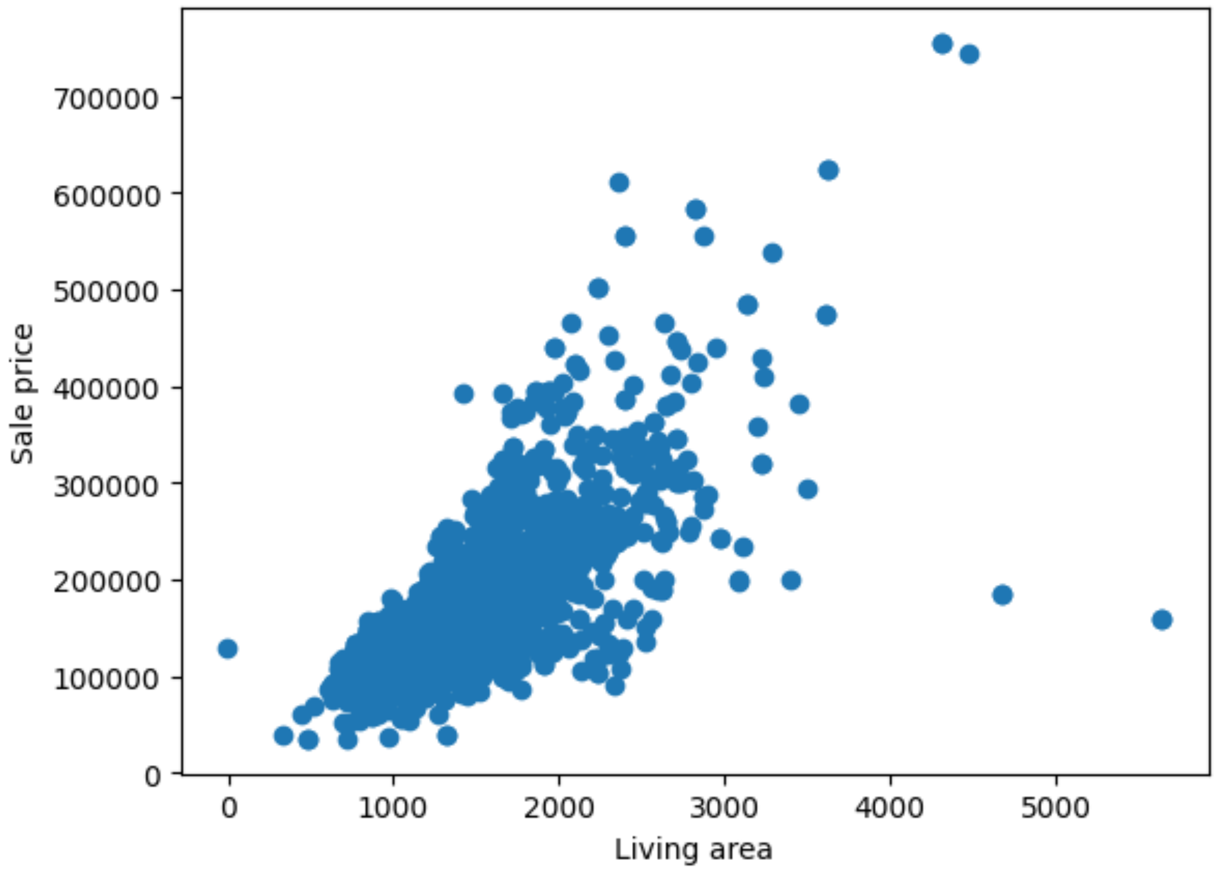
In [ ]:
```
livecode_data = data[['GrLivArea', 'SalePrice']]

livecode_data.head()
```

Out[ ]:

	GrLivArea	SalePrice
1557	1142	134000
1004	1504	181000
1518	2234	501837
348	1626	154000
794	1895	194500

Exploration

In [ ]:
```
import matplotlib.pyplot as plt

# Plot Living area vs Sale price
plt.scatter(data['GrLivArea'], data['SalePrice'])

# Labels
plt.xlabel("Living area")
plt.ylabel("Sale price")

plt.show()
```



Training

Training a Linear Regression model with Sklearn `LinearRegression`

In [ ]:
```
# Import the model
from sklearn.linear_model import LinearRegression

# Instantiate the model (💡 in Sklearn often called "estimator")
model = LinearRegression()

# Define X and y
X = data[['GrLivArea']]
y = data['SalePrice']

# Train the model on the data
model.fit(X, y)
```

Out[ ]:
```
▾ LinearRegression
LinearRegression()
```

👉 [Sklearn `LinearRegression` documentation (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

At this stage, the model has learned the optimal **parameters** - slope `a` and intercept `b` - needed to map the relationship between `X` and `y` .

Model Attributes

`a` (slope) and `b` (intercept) are stored as model attributes and can be accessed.

In [ ]:
```
# View the model's slope (a)
model.coef_
```

Out[ ]:
```
array([105.00927564])
```

In [ ]:
```
# View the model's intercept (b)
model.intercept_
```

Out[ ]:
```
22104.121010020754
```

Scoring

Each Scikit-learn algorithm has a default scoring metric.

`LinearRegression` uses the **Coefficient of Determination** (R^2) by default.

- R^2 represents the proportion of the variance of the target explained by the features.
- The score typically varies between 0 and 1
- The higher score the better the model

In [ ]:
```
# Evaluate the model's performance
model.score(X, y)
```

Out[ ]:
```
0.48960426399689116
```

💡 Different models will have different default scoring metrics. You can look them up in the [`.score()` method in the model's docs (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linearregression#sklearn.linear_model.LinearRegression.score)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linearregression#sklearn.linear_model.LinearRegression.score).

For example, a classifier like `LogisticRegression` will default scoring to [accuracy (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logisticregression#sklearn.linear_model.LogisticRegression.score)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logisticregression#sklearn.linear_model.LogisticRegression.score).

Predicting

The trained model can be used to predict new data

In [ ]:
```
# Predict on new data
new_data = pd.DataFrame({'GrLivArea': [1000]})

model.predict(new_data)
```

Out[ ]:
```
array([127113.39664561])
```

👉 An apartment with a surface area of 1000 ft^2 has a predicted value of about U$127K.

❗ Note that your `X` (features) almost always need to be a 2D-array when passed as an argument to an `sklearn` API method

Sklearn modeling flow

1. Import the model: `from sklearn import model`
2. Instantiate the model: `model = model()`
3. Train the model: `model.fit(X, y)`
4. Evaluate the model: `model.score(new_X, new_y)`
5. Make predictions: `model.predict(new_X)`

❓ What did we do wrong when scoring the model's performance?

👉 We scored the model on the same data it was trained on!

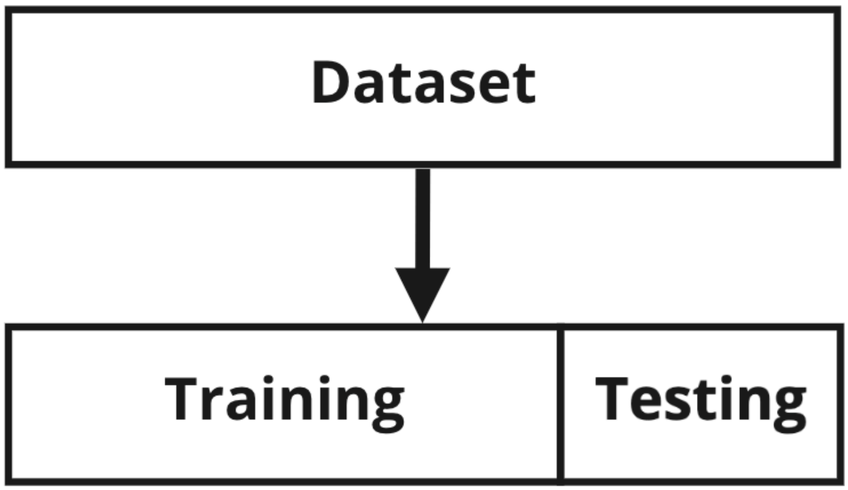
4. Generalization

The performance of a Machine Learning model is evaluated on its ability to **generalize** when predicting **unseen data**.

The Holdout Method

The Holdout Method is used to evaluate a model's ability to generalize. It consists of splitting the dataset into two sets:

- **Training set** (~70%)
- **Testing set** (~30%)



Example

Imagine our dataset has 9 observations

	GrLivArea	SalePrice
0	1710	208500
1	1262	181500
2	1786	223500
3	1717	140000
4	2198	250000
5	1362	143000
6	1694	307000
7	2090	200000
8	1774	129900

Train

Test

💻 `train_test_split`

Let's model the `SalePrice` (`y`) according to the `GrLivArea` (`X`) whilst keeping generalization in mind.

In [ ]:
```
livecode_data.head()
```

Out[ ]:

	GrLivArea	SalePrice
1557	1142	134000
1004	1504	181000
1518	2234	501837
348	1626	154000
794	1895	194500

Splitting

In [ ]:
```
from sklearn.model_selection import train_test_split

# split the data into train and test
train_data, test_data = train_test_split(livecode_data, test_size=0.3)

# Ready X's and y's
X_train = train_data[['GrLivArea']]
y_train = train_data['SalePrice']

X_test = test_data[['GrLivArea']]
y_test = test_data['SalePrice']
```

You could also directly pass `X` and `y` to `train_test_split` .

In [ ]:
```
# Ready X and y
X = livecode_data[['GrLivArea']]
y = livecode_data['SalePrice']

# Split into Train/Test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
3)
```

Training and scoring

In [ ]:
```
# Instantiate the model
model = LinearRegression()

# Train the model on the Training data
model.fit(X_train, y_train)

# Score the model on the Test data
model.score(X_test,y_test)
```

Out[ ]:
```
0.48189443216474215
```

❓ Can you think about any limitations of the Holdout Method?

Data split is random

Different random splits will create different results

In [ ]:
```
### RUN THIS CELL MULTIPLE TIMES TO SEE DIFFERENT SCORES

# Split into Train/Test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
3)

# Instantiate the model
model = LinearRegression()

# Train the model on the Training data
model.fit(X_train, y_train)

# Score the model on the Test data
model.score(X_test,y_test)
```

Out[ ]:
```
0.5246872071350943
```

We can use the `random_state` option, but we can be easily tempted to pick one with the best score 😈

Loss of information

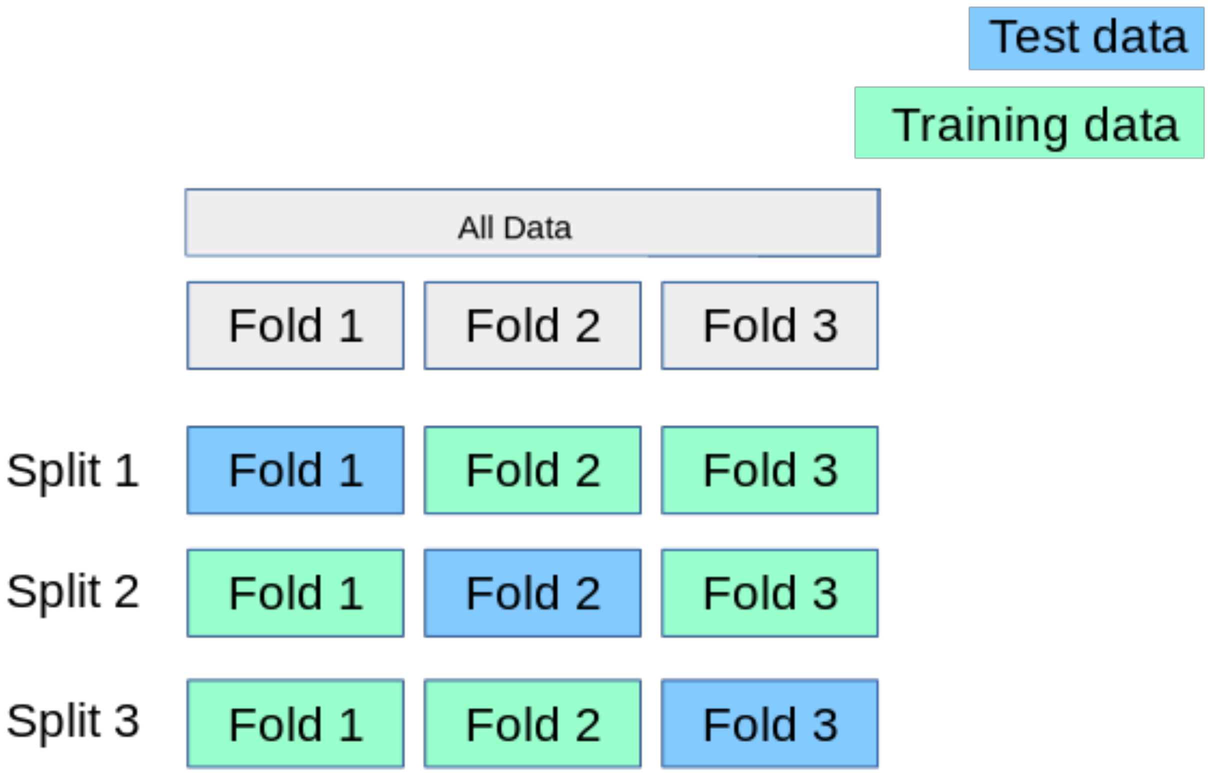
The data in the Test set is not used to train the model. If you have a small dataset, that loss could be significant!

❓ How would you solve that issue?

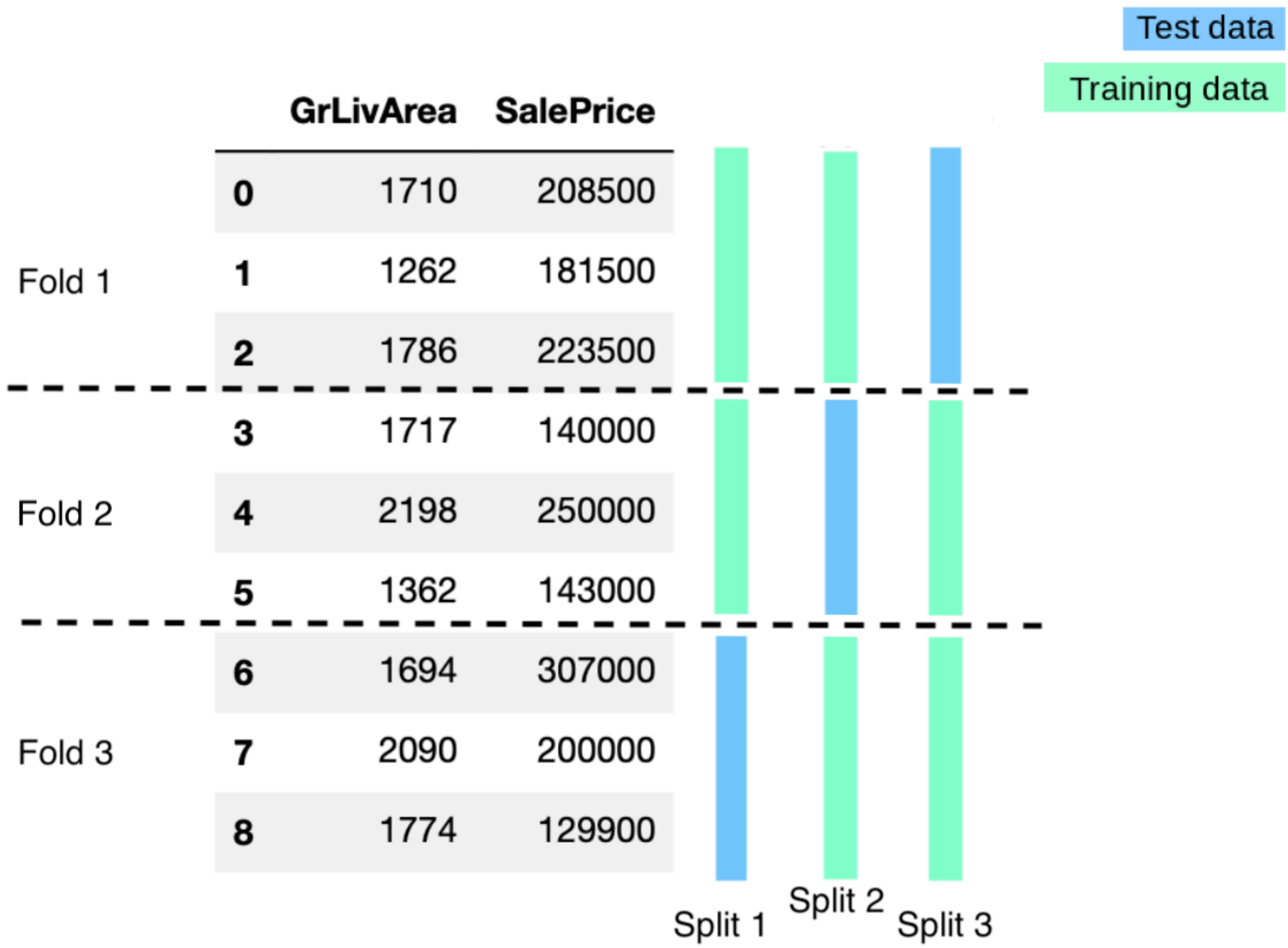
👉 Average the scores of multiple holdout splits.

K-Fold Cross Validation

1. The dataset is split into number of folds K
2. For each split, a sub-model is trained and scored
3. The average score of all sub-models is the **cross-validated** score of the model



Dataframe view



💻 `cross_validate`

In [ ]:
```python
from sklearn.model_selection import cross_validate

# Instantiate model
model = LinearRegression()

# 5-Fold Cross validate model
cv_results = cross_validate(model, X, y, cv=5)

# Scores
print(cv_results['test_score'])

# Mean of scores
cv_results['test_score'].mean()
```

```
[0.29966592 0.43383776 0.552854   0.59169285 0.51451021]
```

Out[ ]:
```
0.47851215043151624
```

Choosing K

- Choosing K is a tradeoff between trustworthy performance evaluation and computational expense
- More K-folds --> more sub-models to average scores from --> more representative score --> more computational time

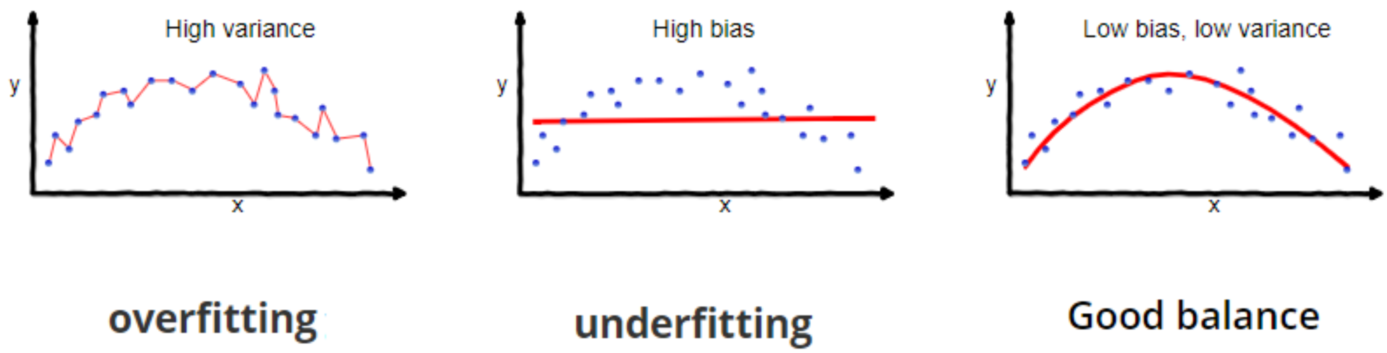
ℹ️ Rule of thumb:
$K = 5$
or
10

🚨 Cross-validation does not output a trained model, it only scores a hypothetical model trained on the entire dataset.

The Bias/Variance tradeoff

For a model to generalize, there will be a tradeoff between **bias** and **variance**.

- **Bias (Underfitting)**: The inability for an algorithm to learn the patterns within a dataset.
- **Variance (Overfitting)**: The algorithm generates an overly complex relationship when modeling patterns within a dataset.



[Source (https://medium.com/towards-data-science/understanding-the-bias-variance-tradeoff-165e6942b229)](https://medium.com/towards-data-science/understanding-the-bias-variance-tradeoff-165e6942b229)

No Free Lunch Theorem

Some models **oversimplify**, while others **overcomplicate** a relationship between the features and the target.

It's up to us as data scientists to make **assumptions** about the data and evaluate reasonable models accordingly.

There is no one-size-fits-all model, this is known as the **[No Free Lunch Theorem](https://en.wikipedia.org/wiki/No_free_lunch_theorem)** (https://en.wikipedia.org/wiki/No_free_lunch_theorem).

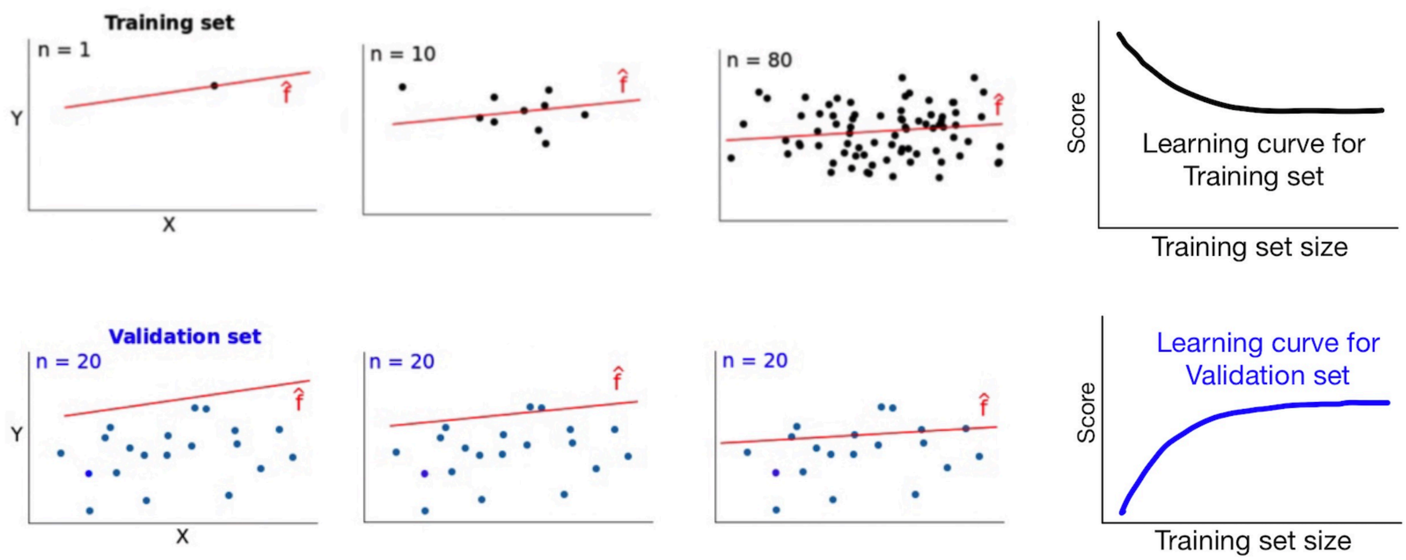
The Learning Curves

Learning curves are used to diagnose three aspects of model behaviour on the dataset:

- Underfitting
- Overfitting
- Whether the model has sufficient data to learn the patterns of the dataset

Concept

Increasing the size of the training set can affect the training and validation scores.



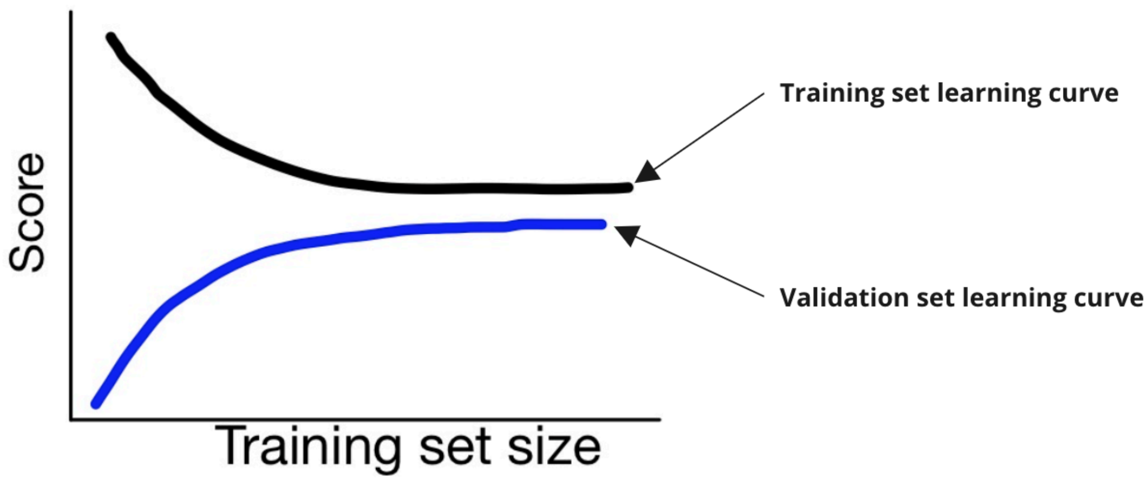
[Source (https://www.dataquest.io/blog/learning-curves-machine-learning/)](https://www.dataquest.io/blog/learning-curves-machine-learning/)

Reading the curves

The two curves are plotted together on the same graph.

As the training size increases:

- The training score will decrease
- The test score will increase
- The curves typically (but not always!) demonstrate convergence

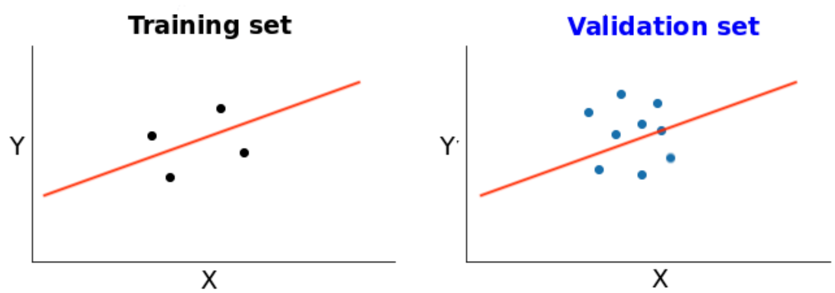


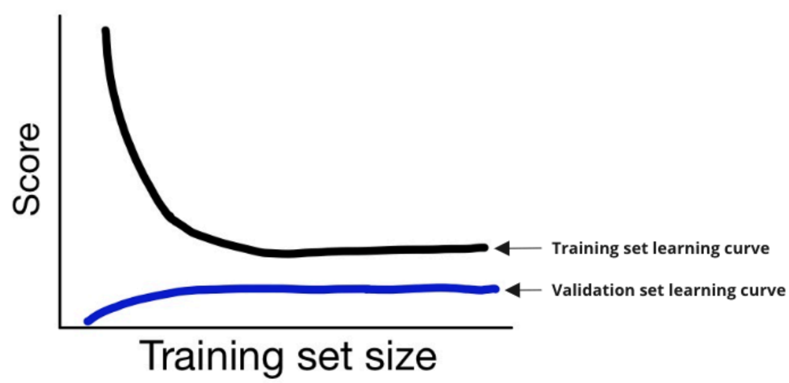
High bias (Underfitting)

Low scores in **both** training and test sets.

If the model cannot determine a relationship in the training set, we cannot expect the model to score highly in the test set.

Training and testing scores converge and **plateau at a low score**. No matter how much data is used for training, the model cannot determine a meaningful relationship.

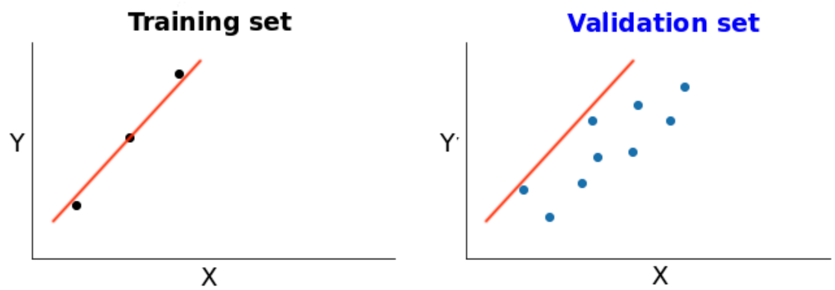


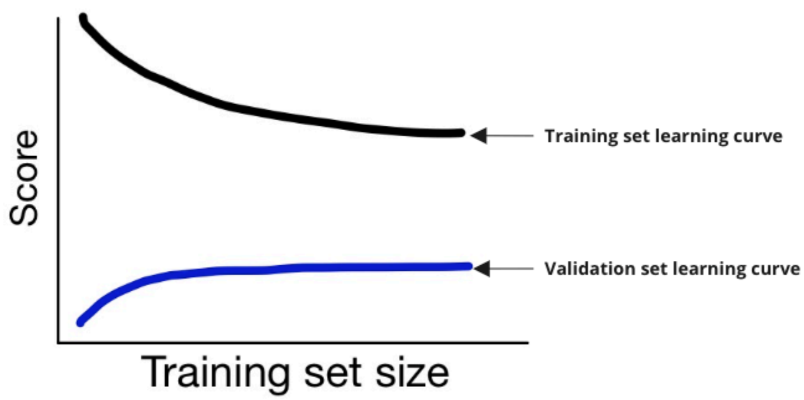


High variance (Overfitting)

The model has paid **too much attention** to both signal and noise in the training data, leading to **high training scores**.

Reliance on noise does not generalize well on unseen data, resulting in **low test scores**. More training data *might* help in this case.

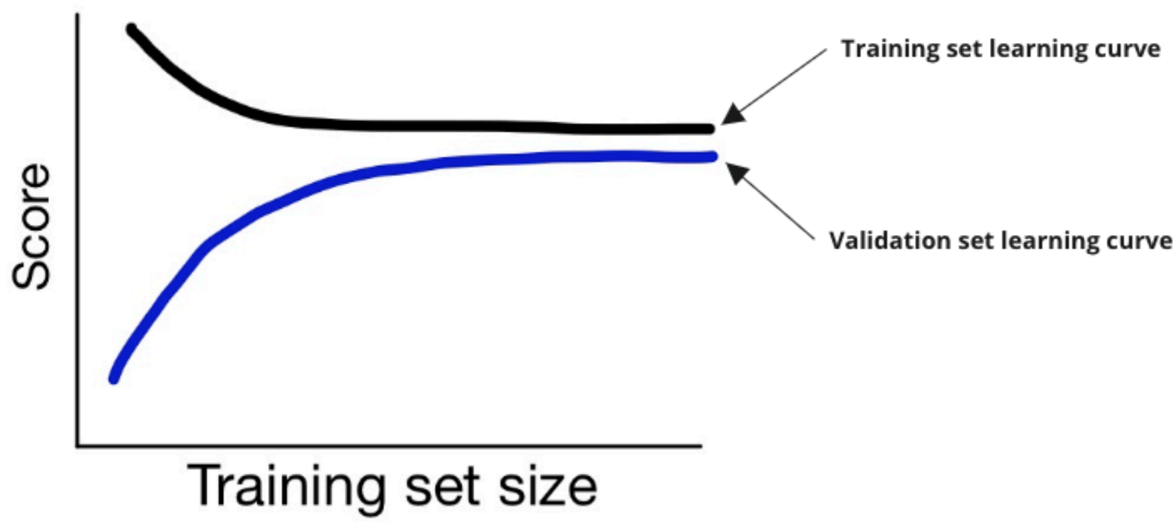




We're going to learn how to combat bias and variance in future lectures ⚔️

Ideal curves

- High score on training set
- High score on test set
- Converged curves



🖥️ Learning Curves with Sklearn

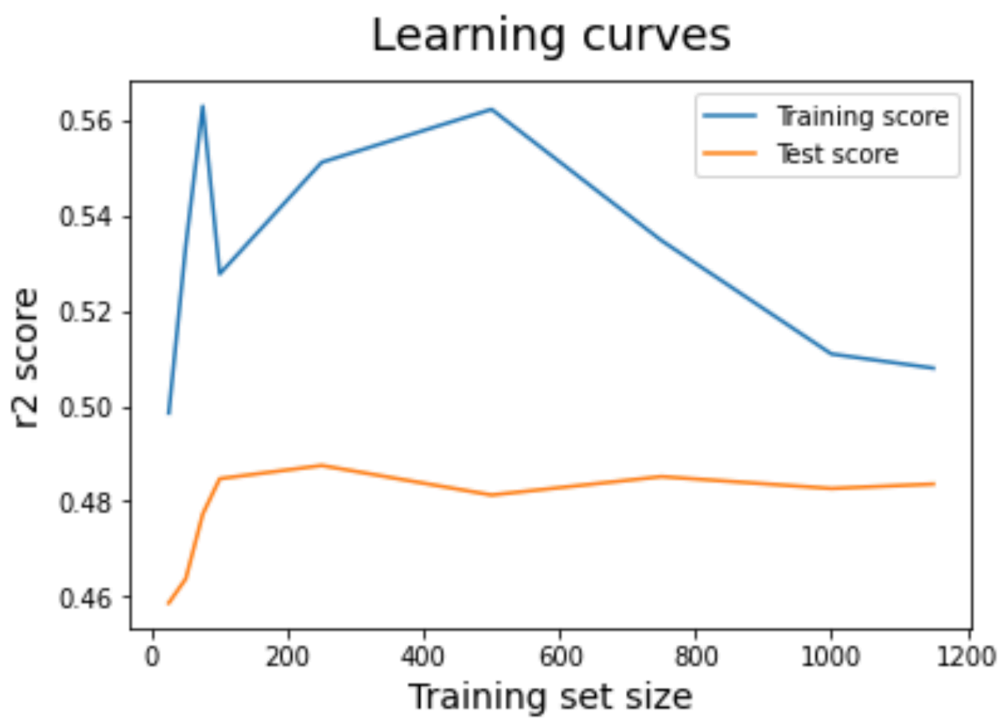
Let's plot and read the learning curves for our Regression model on house prices.

In [ ]:
```
import numpy as np
from sklearn.model_selection import learning_curve

train_sizes = [25,50,75,100,250,500,750,1000,1150]

# Get train scores (R2), train sizes, and validation scores using `lea
rning_curve`
train_sizes, train_scores, test_scores = learning_curve(
    estimator=LinearRegression(), X=X, y=y, train_sizes=train_sizes, c
v=5)

# Take the mean of cross-validated train scores and validation scores
train_scores_mean = np.mean(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)

# plt.plot(train_sizes, train_scores_mean, label = 'Training score')
# plt.plot(train_sizes, test_scores_mean, label = 'Test score')
# plt.ylabel('r2 score', fontsize = 14)
# plt.xlabel('Training set size', fontsize = 14)
# plt.title('Learning curves', fontsize = 18, y = 1.03)
# plt.legend()
```

Reading our curves 1/2

👉 The two curves have converged and plateaued:

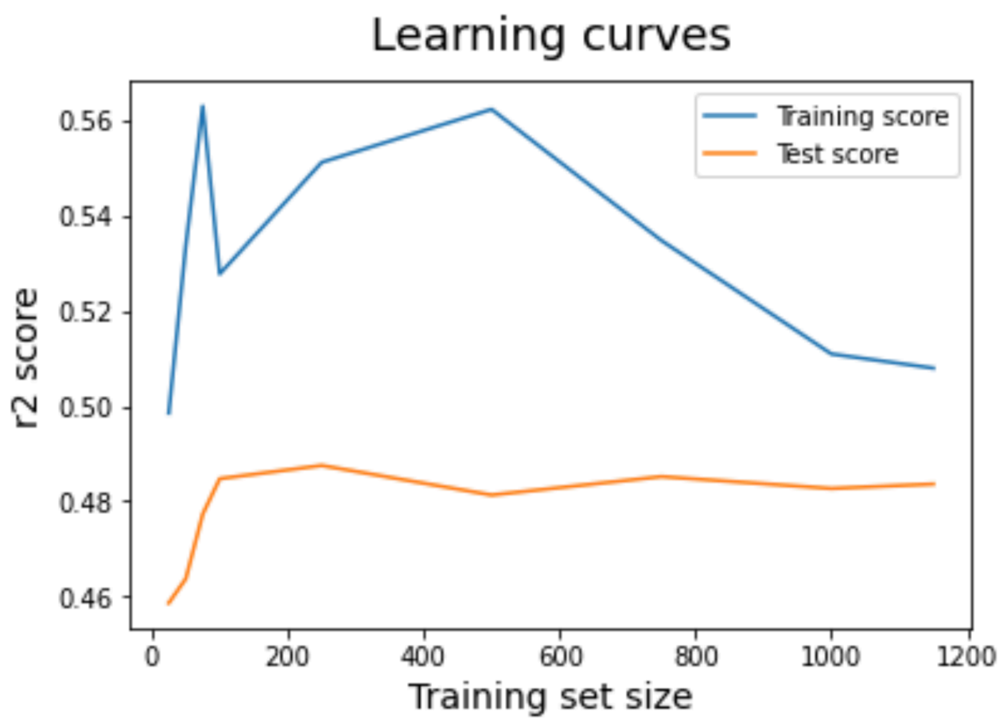
- The model is doing the best it can at finding the patterns within the data
- Adding more data will not improve its performance



Reading our curves 2/2

👉 The score is still relatively low (R2 around 0.5):

- It's up to us to try and improve this score while still maintaining generalization.
- You will learn how to do this in the exercises and in the next couple of lectures! 💪



Your turn! 🚀