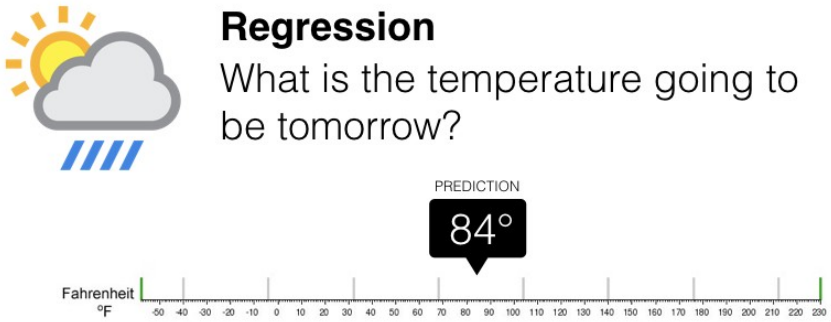
Data Preparation

Recap of `Fundamentals of Machine Learning`

Regression vs. Classification

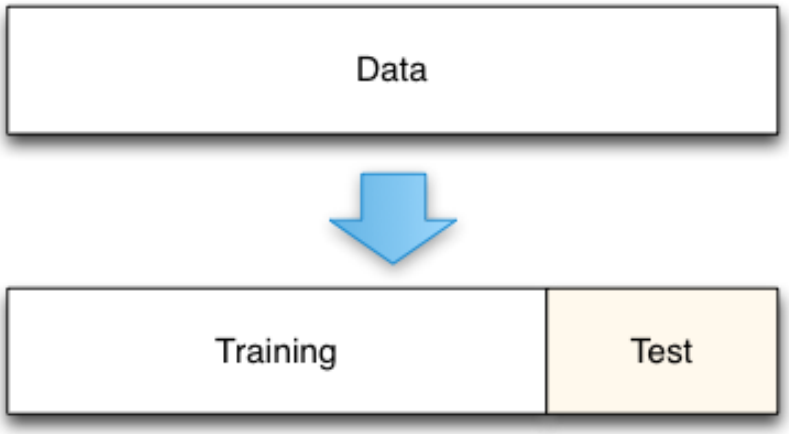


The Sklearn modelling workflow

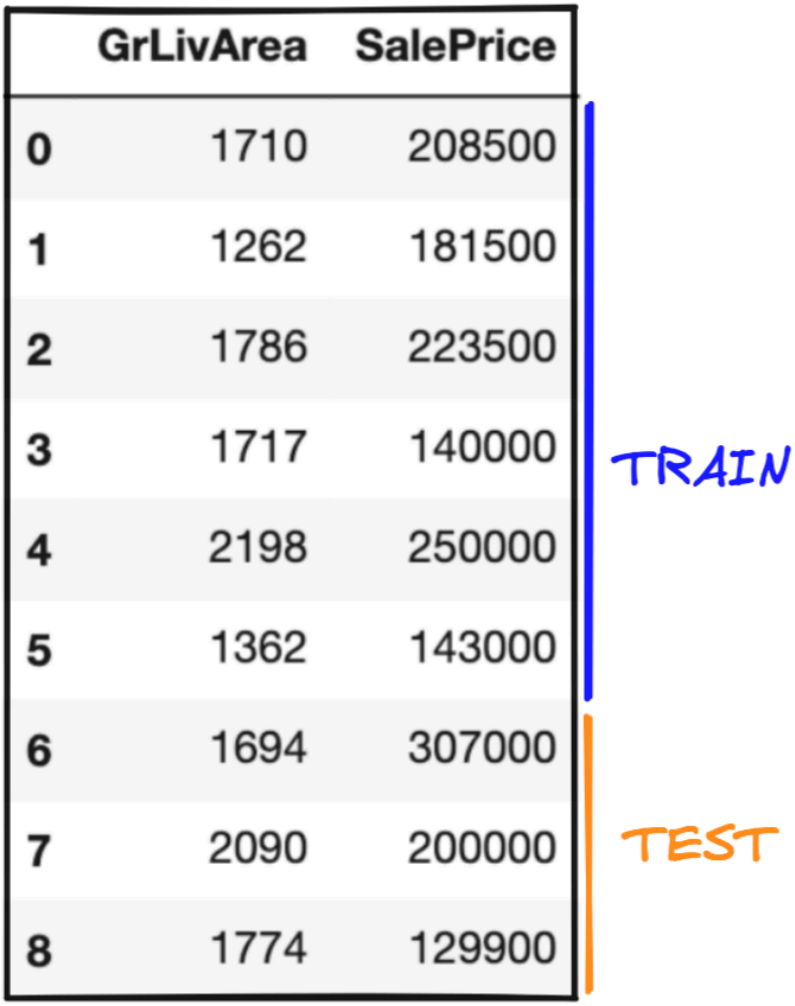
```
from sklearn.some_module import SomeModel

model = SomeModel()
model.fit(X_train,y_train)
model.score(X_test,y_test)
model.predict(X_new)
```

The Holdout Method



The Holdout Method - Dataframe view

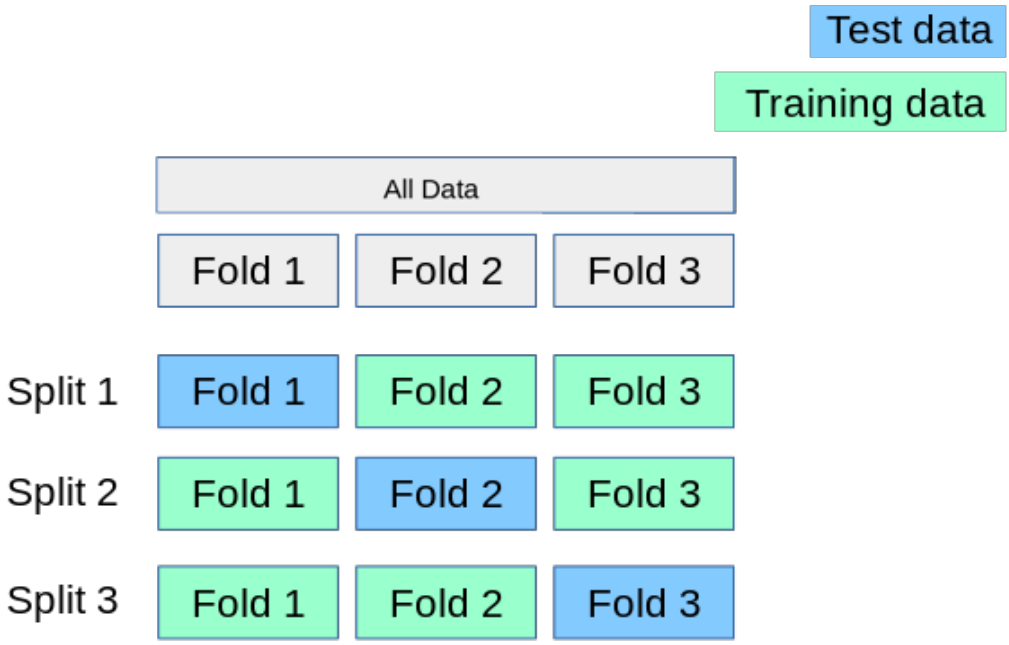


The Holdout Method in `Scikit-Learn`

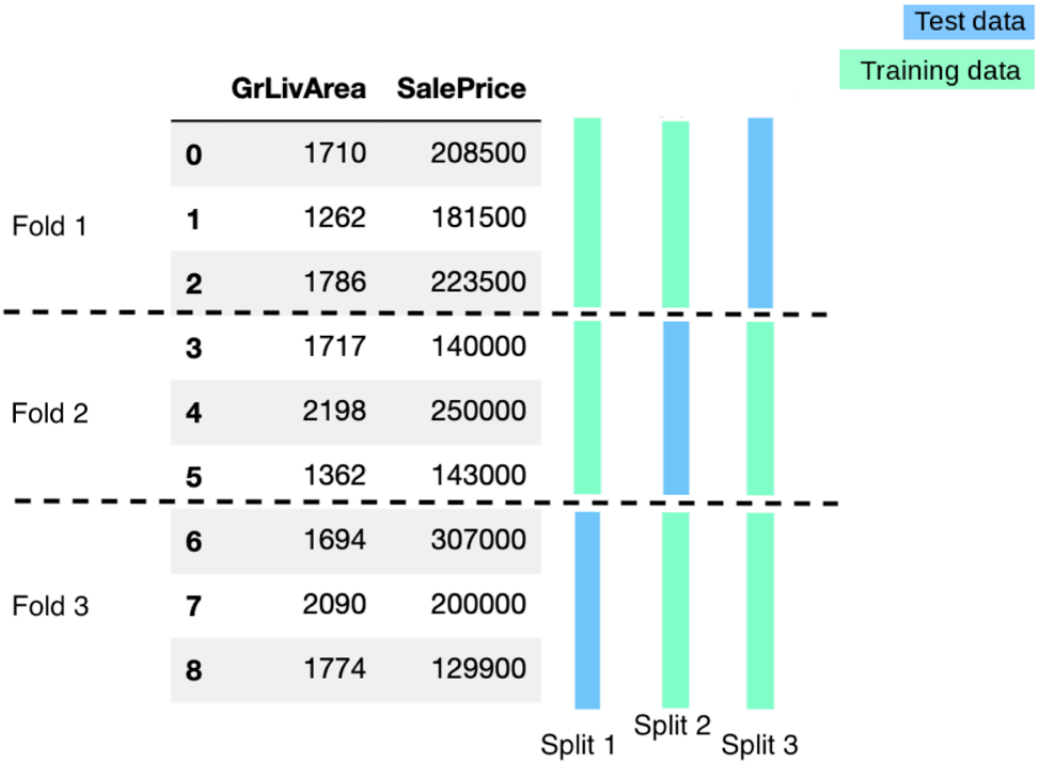
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=88)
```

K-Fold Cross validation



Cross validation - Dataframe view

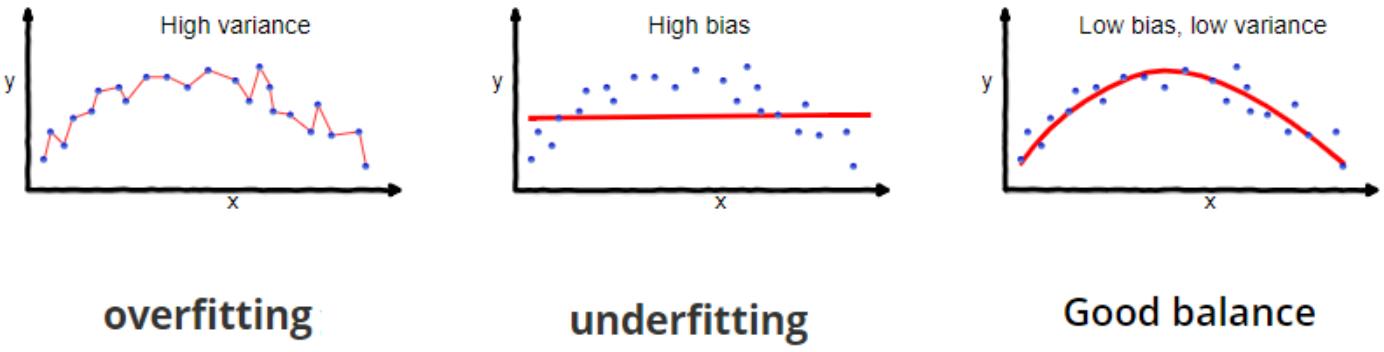


Cross validation in `Scikit-Learn`

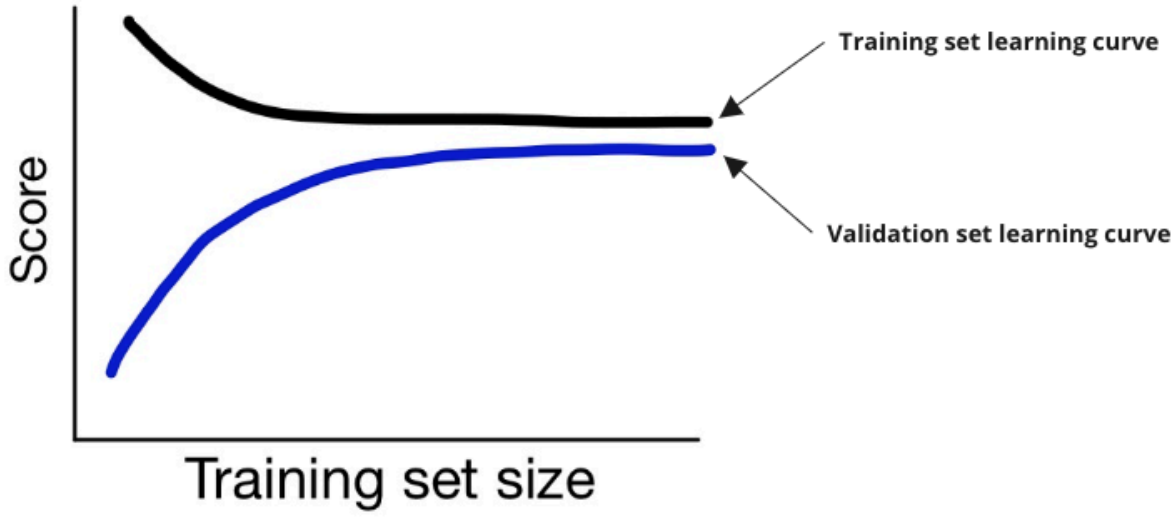
```
from sklearn.model_selection import cross_validate

cross_validate(model, X, y, cv = 5) # returns test_score, fit_time and sco
re_time
```

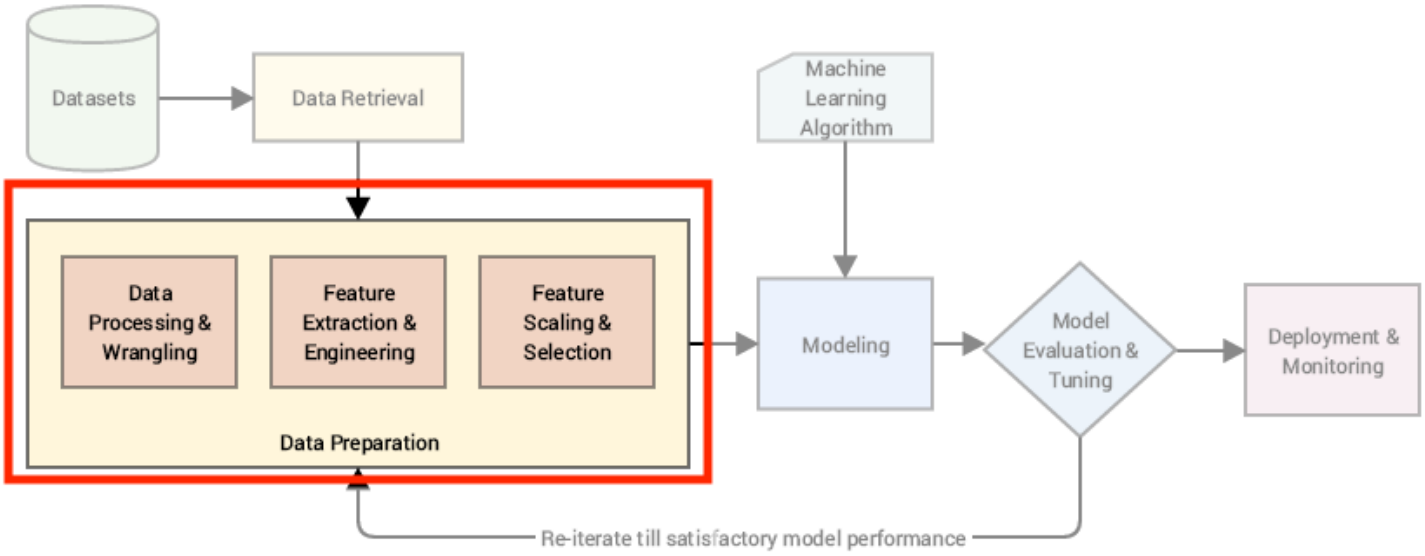
Bias/Variance tradeoff



Learning Curves



Plan of the lecture



General prerequisites

(1) 👥 Duplicates

(2) 🔮 Missing data

(3) 🐳 Outliers

Numerical columns

(4) 🔢 Scaling

Balanced datasets

(5) ⚖️ Balancing

Categorical columns

(6) 🔠 Encoding

(7) 🟨 Discretizing

Generating new features

(8) 🟨 Feature creation

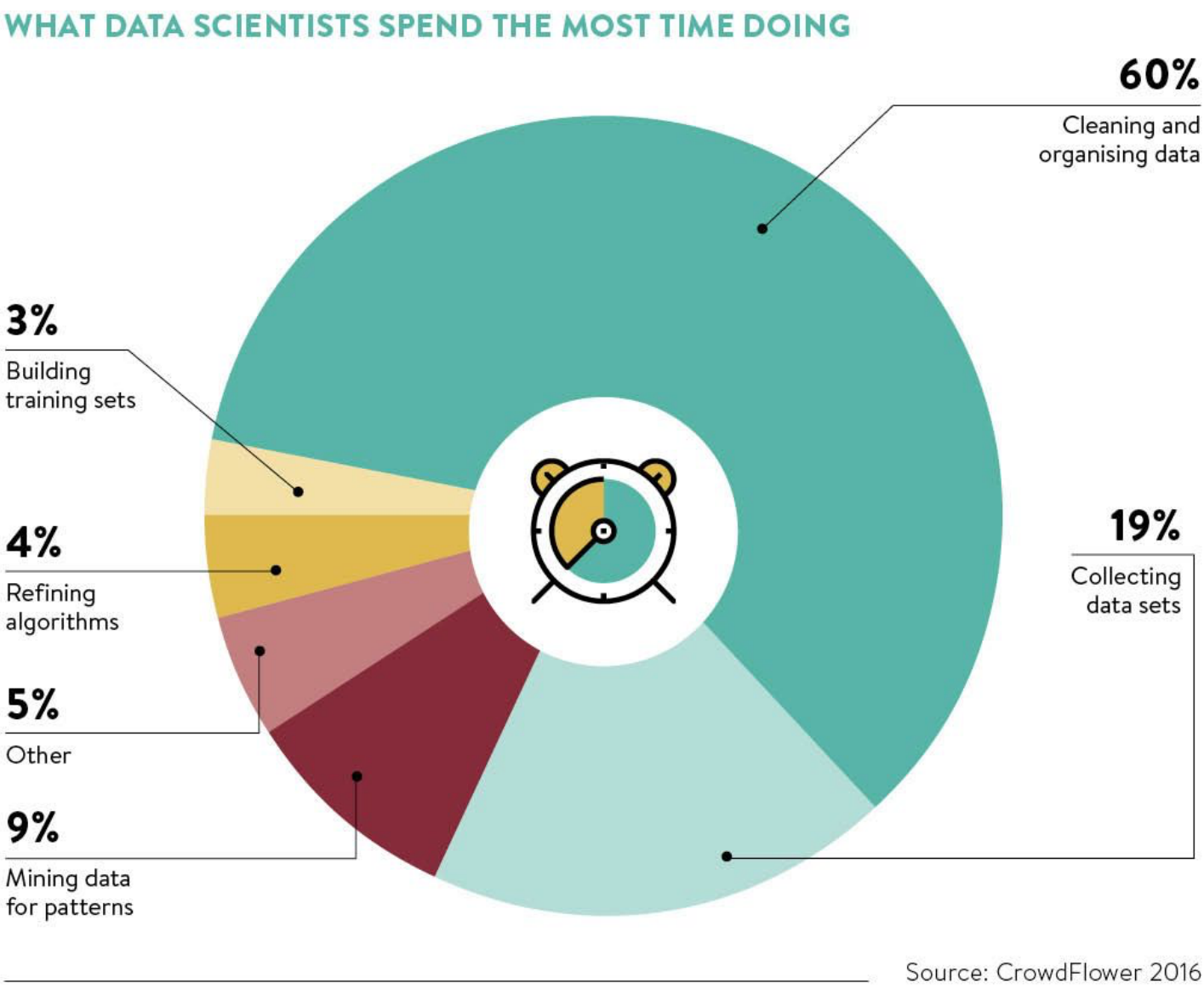
Using the most relevant features

(9) 🤖 Feature selection, Modelling and Feature Permutation

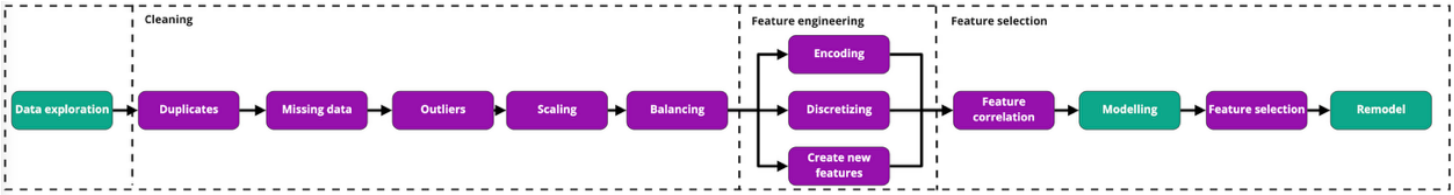
🛠️ Why preprocessing?

- Raw data is dirty and noisy
- Machine learning algorithms have certain constraints regarding input data
- Transformations can improve the model performance

🔗 Source (https://xkcd.com/1838/)



🔗 Source (hhttps://visit.figure-eight.com/rs/416-ZBE-142/images/CrowdFlower_DataScienceReport_2016.pdf)



(https://github.com/lewagon/data-images/raw/master/ML/preparation_steps.pdf)

🏠 Let's consider the `Houses dataset` we used in the previous lecture `01 - Fundamentals of ML`

👉 Now, we are going to include some new features in our modelling: `Alley`, `Street`, `WallMat`, `Pesos`, and `MoSold`.

- 💾 Download the dataset here (https://wagon-public-datasets.s3.amazonaws.com/Machine%20Learning%20Datasets/ML_Houses_dataset.csv)
- ℹ️ Have a look at the full dataset description here (https://wagon-public-datasets.s3.amazonaws.com/Machine%20Learning%20Datasets/ML_Houses_dataset_description.txt).

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

Out[ ]:

	GrLivArea	BedroomAbvGr	KitchenAbvGr	OverallCond	Pesos	Alley	Street	WallMat
0	1710	3	1	5	4170000.0	NaN	Pave	Concrete
1	1262	3	1	8	3630000.0	NaN	Pave	Wood
2	1786	3	1	5	4470000.0	NaN	Pave	Wood
3	1717	3	1	5	2800000.0	NaN	Pave	Concrete
4	2198	4	1	5	5000000.0	NaN	Pave	Concrete

(1) 👥 Duplicates

❓ Duplicated observations can discredit the performance evaluation of a model. Why ❓

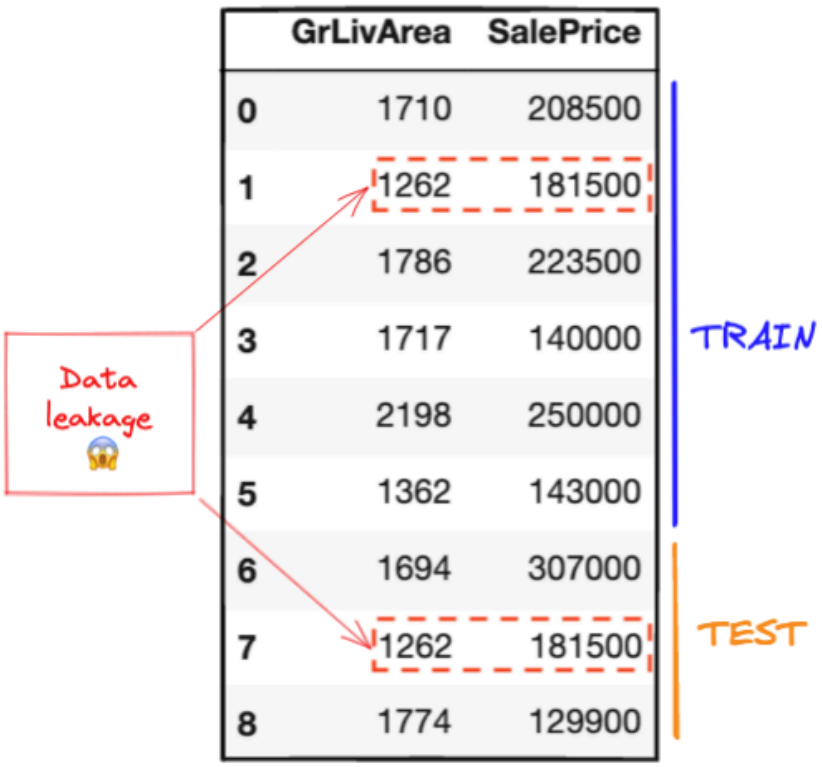
In [ ]:
```
data[["GrLivArea","SalePrice"]].head(10)
```

Out[ ]:

	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
9	1077	118000

⚠️ Data Leakage

- In order to evaluate a model's ability to generalize, **the data in the test set should remain unseen by the algorithm during the training phase**.
  - If there are **duplicated rows** present in ***both*** the **training set** and the **test** set, this can cause **unreliable scores**.



💻 drop_duplicates

In [ ]:
```
len(data) # Check number of rows before removing duplicates
```

Out[ ]:
```
1760
```

In [ ]:
```
data.duplicated() # Check whether a row is a duplicated version of a previous row
```

Out[ ]:
```
0       False
1       False
2       False
3       False
4       False
        ...
1755     True
1756     True
1757     True
1758     True
1759     True
Length: 1760, dtype: bool
```

In [ ]:
```
data.duplicated().sum() # Compute the number of duplicated rows
```

Out[ ]: 303

In [ ]:
```
data = data.drop_duplicates() # Remove duplicates
len(data)# Check new number of rows
```

Out[ ]: 1457

(2) 🔮 Missing Data

Common reasons for missing data

- 👨‍💻 Programming error
- 😐 Failure of measurement (e.g. a patient in a clinical study misses a scheduled visit)
- 🎲 Random events (e.g. meteorological data collection device runs out of power)
- ❌ Incorrect text entries
- etc...

Common representations for missing data

- `NaN` (*not a number*)
- `999` (*or any suspiciously large negative number...*)
- `?`
- $\pm\infty$ (infinite values)
- `""` (*Empty string*)

💻 Detecting missing data

In [ ]:
```
# Counting the number of NaN for each column
data.isnull().sum().sort_values(ascending=False)
```

Out[ ]:
```
WallMat          1452
Alley            1367
Pesos              10
GrLivArea           0
BedroomAbvGr        0
KitchenAbvGr        0
OverallCond         0
Street              0
SalePrice           0
dtype: int64
```

In [ ]:
```
# Counting the percentage of NaN for each column
data.isnull().sum().sort_values(ascending=False) / len(data) #NaN perc
entage for each column
```

Out[ ]:
```
WallMat          0.996568
Alley            0.938229
Pesos            0.006863
GrLivArea        0.000000
BedroomAbvGr     0.000000
KitchenAbvGr     0.000000
OverallCond      0.000000
Street           0.000000
SalePrice        0.000000
dtype: float64
```

Handling missing data

How you handle missing values will differ from field to field and dataset to dataset.

- What might have caused the missing values?
- Do the missing values represent a particular story or event?
- Can I replace them by another value?
- Can I afford to lose any data?

🚨 Some of these questions require domain knowledge. Ensure you are aware of what each column truly represents before starting any machine learning task!

👉 You can download the description of the dataset that we will be playing with here (https://wagon-public-datasets.s3.amazonaws.com/Machine%20Learning%20Datasets/ML_Houses_dataset_description.txt).

`WallMat`

In [ ]:
```
# Percentage of missing values in WallMat
(data.WallMat.isnull().sum() / len(data))
```

Out[ ]:
```
0.9965682910089224
```

In [ ]:
```
# 99% is way too high, let's drop this feature
data = data.drop(columns='WallMat') # Drop WallMat column
data.head()
```

Out[ ]:

	GrLivArea	BedroomAbvGr	KitchenAbvGr	OverallCond	Pesos	Alley	Street	SalePrice
0	1710	3	1	5	4170000.0	NaN	Pave	208500
1	1262	3	1	8	3630000.0	NaN	Pave	181500
2	1786	3	1	5	4470000.0	NaN	Pave	223500
3	1717	3	1	5	2800000.0	NaN	Pave	140000
4	2198	4	1	5	5000000.0	NaN	Pave	250000

`Alley`

In [ ]:
```
#Percentage of missing values in Alley
(data.Alley.isnull().sum() / len(data))
```

Out[ ]:
```
0.938229238160604
```

🚨 Missing data does not necessarily mean a lack of information!

👉 Here, you have to be careful. A *NaN* simply means that the house doesn't have an Alley.

In [ ]:
```
import numpy as np

data.Alley = data.Alley.replace(np.nan, "NoAlley") # Replace NaN by "NoAlley"
data.Alley.value_counts() # Check count of each category
```

Out[ ]:
```
NoAlley    1367
Grvl         50
Pave         40
Name: Alley, dtype: int64
```

Pesos

In [ ]:
```
# Percentage of missing values in Pesos
(data.Pesos.isnull().sum() / len(data))
```

Out[ ]:
```
0.0068634179821551134
```

In [ ]:
```
# Option 1: Drop rows where Pesos value is missing
data.dropna(subset=['Pesos'])

# Option 2: Replace missing Pesos values with mean
data.Pesos.replace(np.nan, data.Pesos.mean())
```

Out[ ]:
```
0       4170000.0
1       3630000.0
2       4470000.0
3       2800000.0
4       5000000.0
           ...
1455    3500000.0
1456    4200000.0
1457    5330000.0
1458    2842500.0
1459    2950000.0
Name: Pesos, Length: 1457, dtype: float64
```

💡 Suggestions:

- **More than 30%** of missing values
  →
  🚮 Potentially drop the feature or the row
- **Less than 30%** of missing values
  →
  💡 Consider an *imputer* with a strategy that makes sense (cf. next slides)

🚨 Keep in mind that imputing a missing value is an approximation. This can generate potential **noise** and/or **bias** for your models.

🖥️ Sklearn's `SimpleImputer`

With this tool called *SimpleImputer*, you can replace missing values with a strategy of your choice (e.g. median, mean, mode, most frequent, ...)

📚 **sklearn.impute.SimpleImputer** (https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html)

In [ ]:
```
from sklearn.impute import SimpleImputer

# Instantiate a SimpleImputer object with your strategy of choice
imputer = SimpleImputer(strategy="mean")

# Call the "fit" method on the object
imputer.fit(data[['Pesos']])

# Call the "transform" method on the object
data['Pesos'] = imputer.transform(data[['Pesos']])

# The mean is stored in the transformer's memory
imputer.statistics_
```

Out[ ]:
```
array([3608796.22667588])
```

How did the `SimpleImputer` work?

`imputer.fit()`

1. `imputer` computes the strategy for the feature(s) it is being fitted on
2. stores the "strategic" value as an attribute

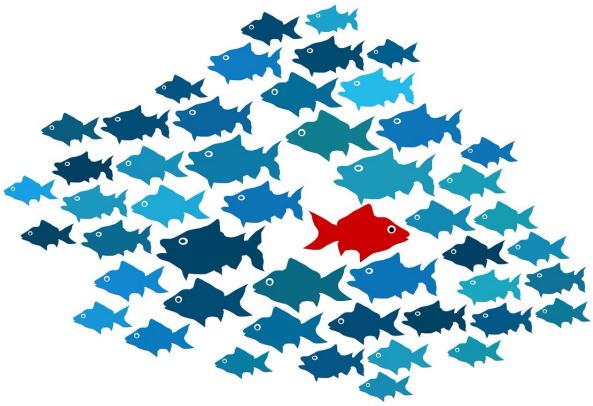
`imputer.transform()`

1. identifies missing values
2. replaces missing values with the strategic value calculated in the `fit` step

 In Scikit Learn, there are a couple of tools designed to help you prepare a dataset before feeding a Machine Learning model with the preprocessed data. They are called **scikit-learn transformers**

- **`.fit()`** : learns and stores constants as attributes of the transformer
- **`.transform()`** : uses these attributes to transform features of your choice from the original dataset

(3) 🐋 Outliers



> ***Outliers are data points which deviate from the rest of the data.***

Common reasons for outliers

- ⌨️ Data entry errors
- 📐 Measurement errors
- 👨‍🔬 Data manipulation and preprocessing errors
- 🆕 Novelties (not errors)

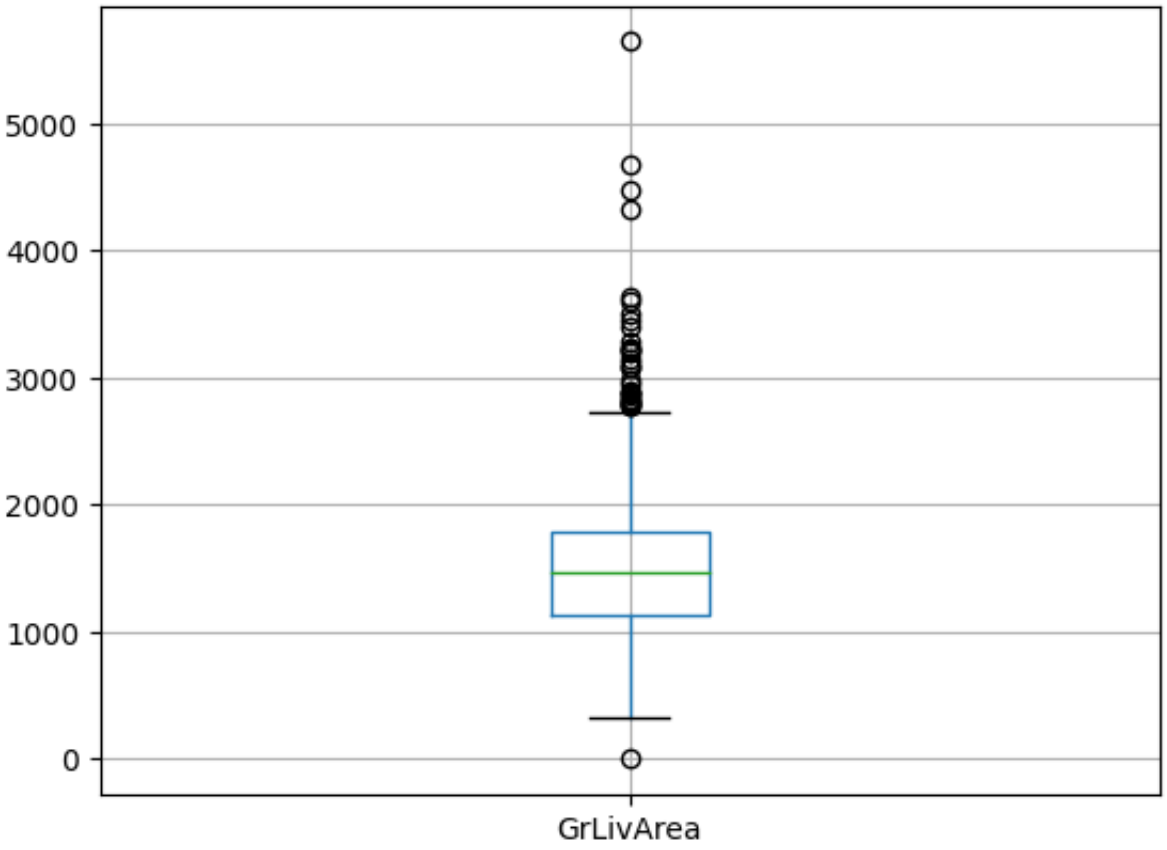
Outliers affect:

- 🕵️ Dataset distributions and patterns
- 🕵️ Central tendency metrics such as the *mean* of a feature
- 🕵️ Dispersion metrics such as *standard deviation*
- 🤖 Performance of a Machine Learning model

Detecting Outliers - Boxplot

We can use ***boxplots*** to visualize outliers within a dataset.

In [ ]:
```
data[['GrLivArea']].boxplot();
```



- 📚 pandas.DataFrame.boxplot (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.boxplot.html)
- 📚 matplotlib.pyplot.boxplot (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.boxplot.html)
- 📚 seaborn.boxplot (https://seaborn.pydata.org/generated/seaborn.boxplot.html)

Are all the outliers "real outliers" ❓

In [ ]:
```
data['GrLivArea'].min()
```

Out[ ]:
```
-1
```

Handling Outliers

- Is the outlier evidently false?
- Could it be a novelty?
- Could it be used as a feature?

🚨 Outliers can be an opinion. We must fully comprehend what an outlier is before removing it from the dataset.

Dropping Outliers

If data is evidently false: a house cannot have a living area of -1 ft^2.

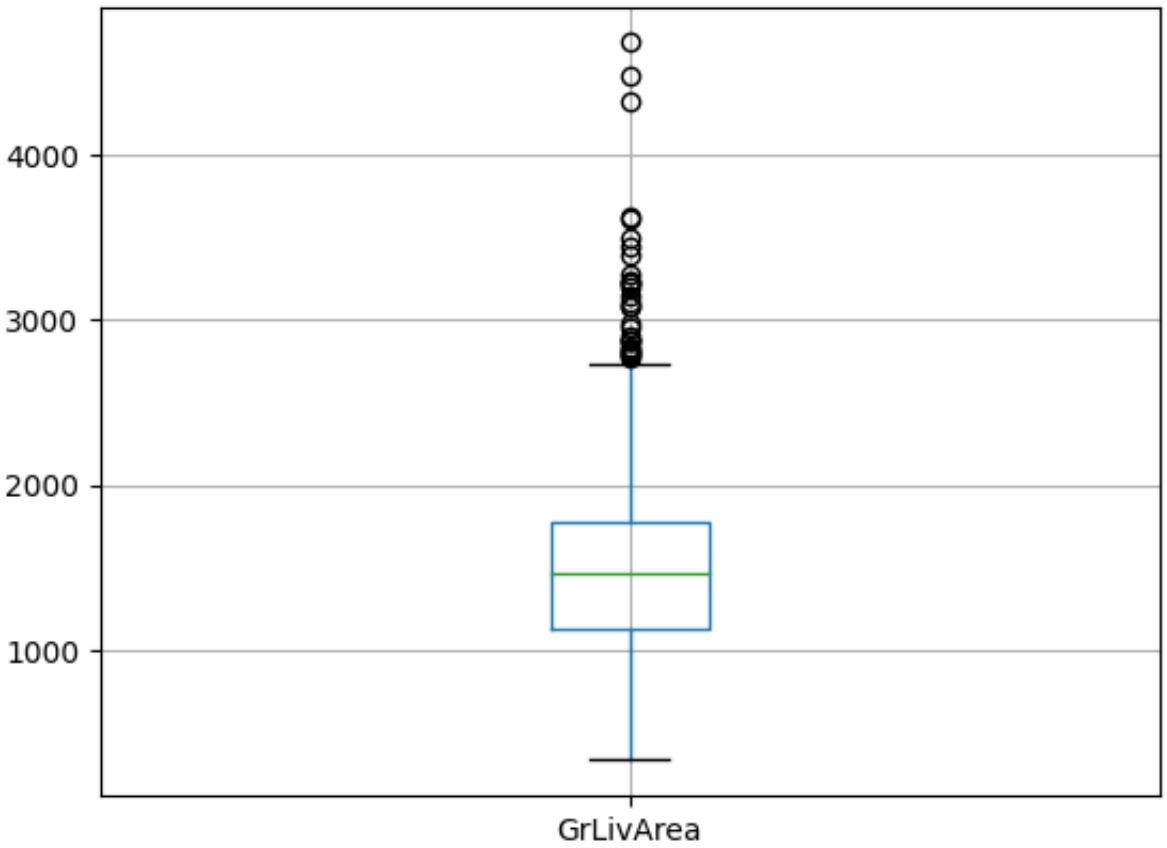
```
In [ ]: data['GrLivArea'] == -1
```

```
Out[ ]: 0       False
        1       False
        2       False
        3       False
        4       False
                ...
        1455    False
        1456    False
        1457    False
        1458    False
        1459    False
        Name: GrLivArea, Length: 1457, dtype: bool
```

In [ ]:
```
# Save the indexes corresponding to rows
# without the absurd -1 value
# and without large mansions (>5000 ft)
boolean_mask = (data['GrLivArea']>0) & (data['GrLivArea']<5000)

# Apply the boolean filtering
data = data[boolean_mask].reset_index(drop=True)

# Visualize the boxplot again
data[['GrLivArea']].boxplot();
```



(4) 🔢 Feature Scaling

> 🪜 ***Feature scaling is the process of transforming numerical features into a common smaller range*** ±

Why scaling?

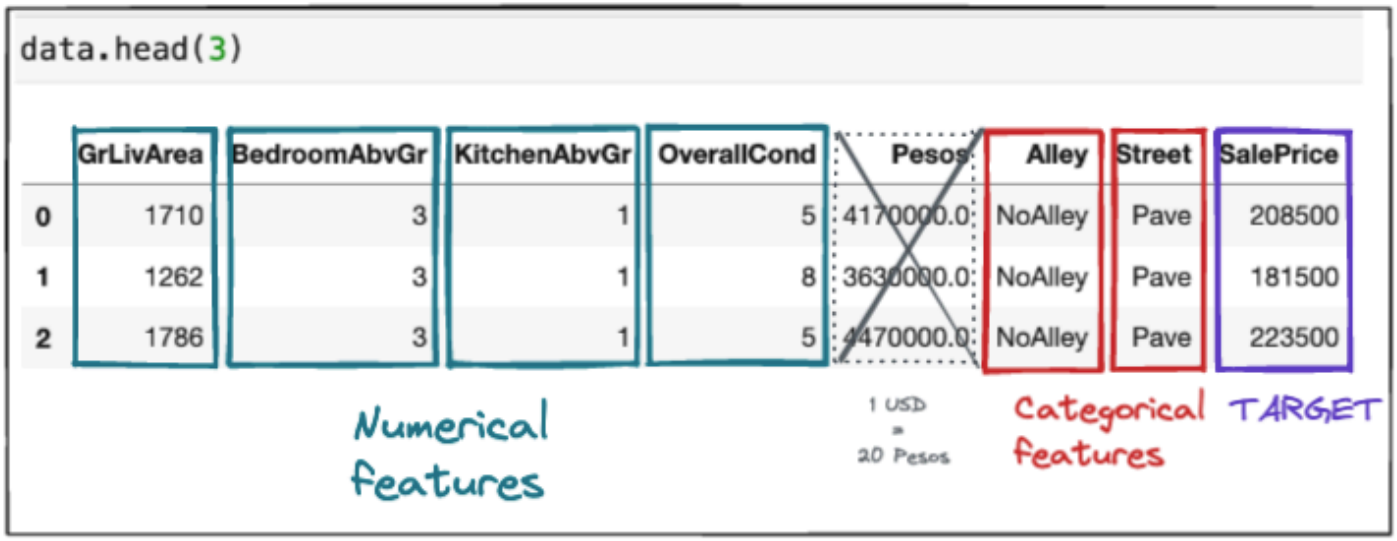
- ❗ Features with large magnitudes can incorrectly outweigh features of small magnitudes
- ⚡ Scaling to smaller magnitudes improves computational efficiency
- 🕵️ Increases interpretability about the impact of each feature in a Machine Learning model

👇 Look at our dataset, which of these features are 🔢 numerical features?

In [ ]:
```
data.head(3)
```

Out[ ]:

	GrLivArea	BedroomAbvGr	KitchenAbvGr	OverallCond	Pesos	Alley	Street	SalePric
0	1710	3	1	5	4170000.0	NoAlley	Pave	20850
1	1262	3	1	8	3630000.0	NoAlley	Pave	18150
2	1786	3	1	5	4470000.0	NoAlley	Pave	22350



Note: this conversion rate can evolve, let's use this one in the lecture.

The most famous scalers

1. **`StandardScaler`** ("Standardizing")
2. **`MinMaxScaler`** ("Normalizing")
3. **`RobustScaler`**

(4.1) Standardizing

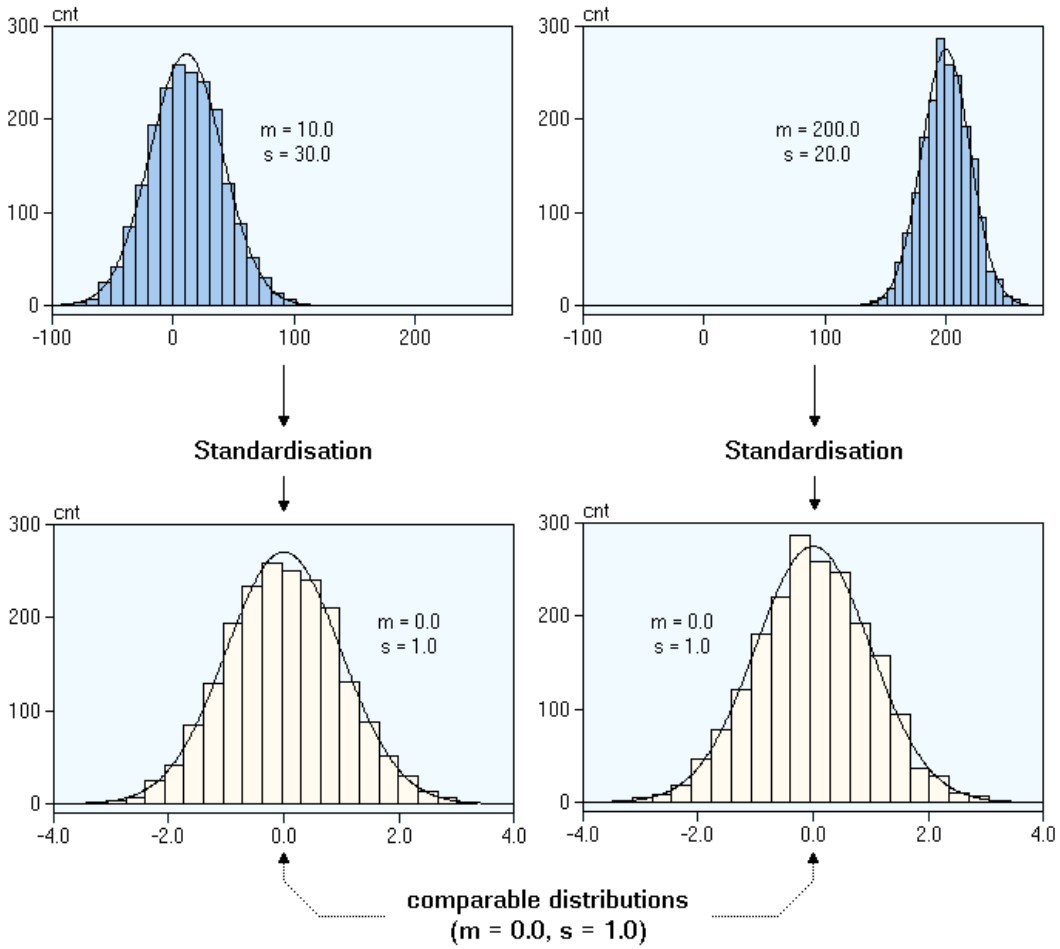
[Math Processing Error]

This operation will transform a feature so that its distribution is:

- centered around 0 ($\mu = 0$)
- with a standard deviation equal to ($\sigma = 1$)

📚 **sklearn.preprocessing.StandardScaler** (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)

The effect of standardization



Standardization: Pros & Cons

✅ Most efficient when a feature is normally distributed

🆗 Does not ensure an exact common range for two different features...

🆗 ...but overall, 99% of the values for a Gaussian-distributed feature are located within the interval $[\mu - 3\sigma, \mu + 3\sigma] = [-3, +3]$
(after standardizing).

❗ Sensitive to outliers...

❗ Can distort relative distances between feature values...

(4.2) Normalizing

[Math Processing Error]

👉 The feature values are compressed in a fixed range [0,1].

📚 **sklearn.preprocessing.MinMaxScaler** (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html)

MinMax Scaling effects and use cases:

✅ Ensures a fixed range in
$[0, 1]$
for all the values of a given feature

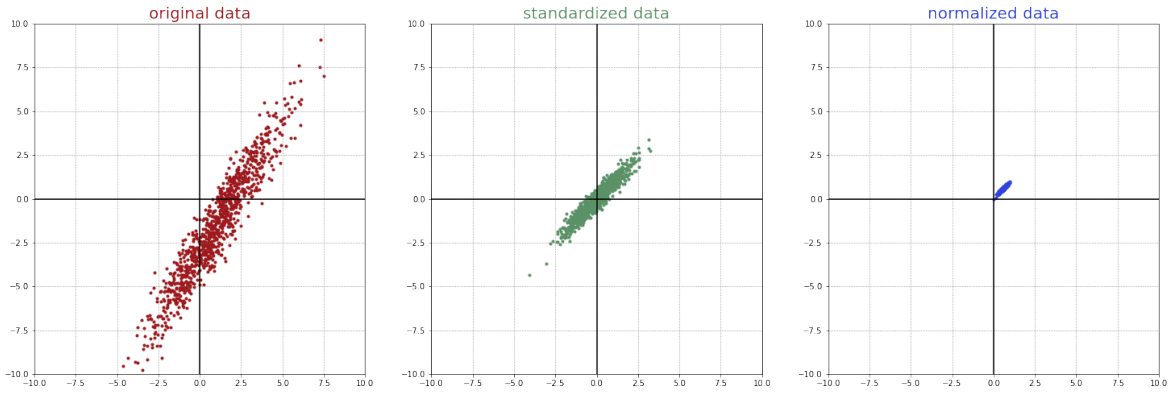
ℹ️ It neither reduces the effect of outliers nor changes skewness.

👉 Preserves matrix sparsity! (*a 0 remains 0 for a positive matrix*)

🚀 Go-to scaling for:

- Ordinal features (*e.g. Olist review score*)
- Positive features or sparse matrix (*for instance: pixel luminosity*)
- The KNearestNeighbors algorithm (*KNN*), a distance-based algorithm that we will learn during the `Performance Metrics` lecture (*why? (https://stats.stackexchange.com/questions/363889/which-type-of-data-normalizing-should-be-used-with-knn#:~:text=Standardization%2C%20on%20the%20other%20hand%2C%20does%20have%20many%20*

Scaling effects
$\rightarrow$
StandardScaler vs. *MinMaxScaler*



🧐 What if you're concerned with outliers?

(4.3) Robust Scaling

✅ Robust Scaling uses:

- the median as central tendency metric
- the interquartile range
  $IQR = Q_3 - Q_1$
  as dispersion metric.

💪 Both of them are less sensitive to outliers than the mean and the standard deviation!
[Math Processing Error]

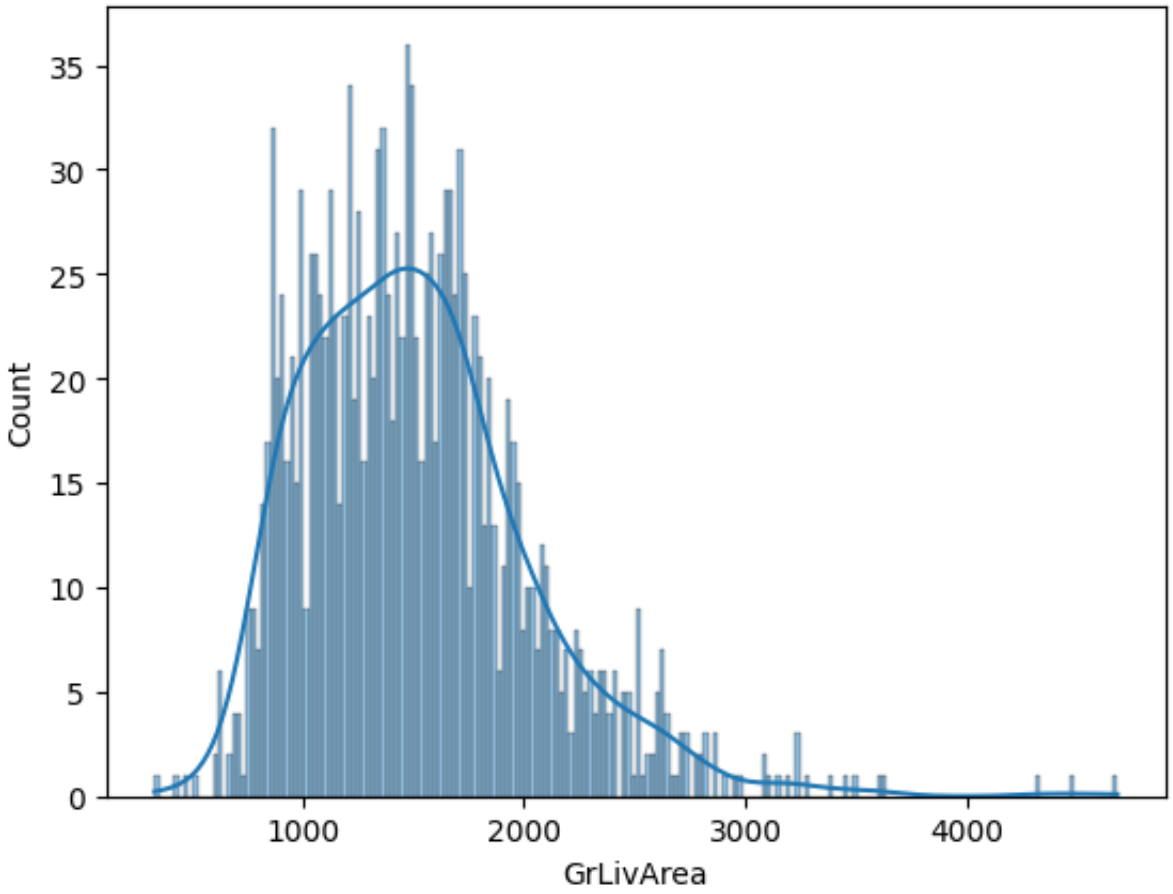
📚 **sklearn.preprocessing.RobustScaler** (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html)

(4.4) 👨🏻‍💻 Scaling in practice with Scikit Learn

👉 Let's scale the `GrLivArea` as an example.

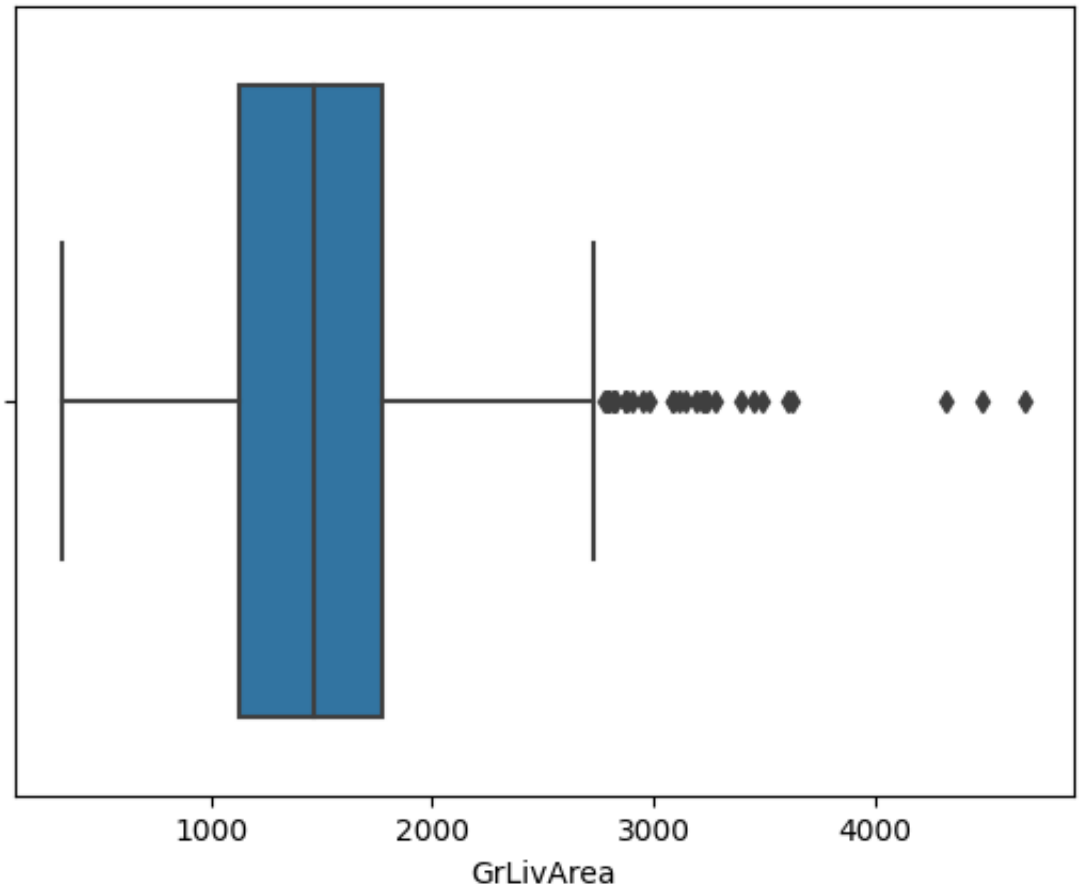
What is the **distribution** of the `GrLivArea` feature ❓

In [ ]:
```
import seaborn as sns
sns.histplot(data['GrLivArea'], bins=200,kde=True);
```



Does `GrLivArea` have outliers ❓

In [ ]:
```
sns.boxplot(data=data, x='GrLivArea');
```



Which scaler should we apply to the `GrLiveArea` feature ❓

💻 **RobustScaler** applied to the `GrLivArea`

In [ ]:
```
from sklearn.preprocessing import RobustScaler

# Step 0 - Instantiate Robust Scaler

rb_scaler = RobustScaler()

# Step 1 - Fit the scaler to the `GrLiveArea`
# to "learn" the median value and the IQR

rb_scaler.fit(data[['GrLivArea']])

# Step 2 - Scale / Transform
# to apply the transformation (value - median) / IQR for every house

data['GrLivArea'] = rb_scaler.transform(data[['GrLivArea']])

data.head()
```

Out[ ]:

	GrLivArea	BedroomAbvGr	KitchenAbvGr	OverallCond	Pesos	Alley	Street	SalePric
0	0.380216	3	1	5	4170000.0	NoAlley	Pave	20850
1	-0.312210	3	1	8	3630000.0	NoAlley	Pave	18150
2	0.497682	3	1	5	4470000.0	NoAlley	Pave	22350
3	0.391036	3	1	5	2800000.0	NoAlley	Pave	14000
4	1.134467	4	1	5	5000000.0	NoAlley	Pave	25000

(4.5) Rules of thumb when scaling 👍

`Feature Transformation/Engineering`

- If your feature is extremely skewed
  →
  consider ***Feature Engineering*** first (*e.g. log(feature)*)

`Robust Scaler`

- If your feature transformation doesn't work
  →
  consider ***Robust Scaling*** this feature
- If your feature is still heavily skewed with outliers that (you assume to be) irrelevant
  →
  consider ***Robust Scaling***

`Standard Scaler`

- Otherwise, ***Standard Scaling*** is a safe bet.
  - Before doing a fine-grained selection of which scalers to use for each feature, you can run a model using only Standard Scaling to run the model quickly.
  - Models like Linear Regression and Neural Networks work quite well with zero-centered features.

`MinMax Scaler`

- If your features form a positive or sparse matrix (*e.g. RGB values in a picture between 0 and 255*)
  →
  consider ***MinMax Scaling***
- If you think that outliers (if any) are full part of the dataset and shouldn't be removed
  →
  consider ***MinMax Scaling***

> All rules of thumb are subject to exceptions 🙃

(5) ⚖️ Dataset balancing

- In a dataset, we also have to deal with categorical columns.
  - Depending on your task, they can be a 🔠 feature or a 🎯 target.
  - They need to be converted into numbers for the Machine Learning algorithm to understand them.

- Before turning classes into numbers, we need to ***check for ⚖️ class imbalance***.
  - 🦠 *`Disease prevalence`*
    - In epidemiology, prevalence is the proportion of a particular population found to be affected by a medical condition (typically a disease or a risk factor such as smoking or seatbelt use)
  - 🙋🏿 *`Race`*
    - Ethnic groups being overrepresented/underrepresented in some contexts
    - Read A. King & E. Puyol-Antón - AI Models can be racially biased (https://www.kcl.ac.uk/news/ai-models-can-be-racially-biased-when-trained-on-unbalanced-data-sets-researchers-find)
  - 🐓 *`Gender`*
    - Cf. Focus2030 - Overview of data resources on gender equality cross the world (https://focus2030.org/Overview-of-data-resources-on-gender-equality-across-the-world)
  - 🛍️ *`E-commerce - Conversion rate`*:
    - Most users ignore the ads and only a small fraction will click on them.
  - 💳 *`Banking - Credit card fraud detection`*:
    - A vast majority of transactions are legitimate and only a small fraction are fraudulent.

Why balancing?

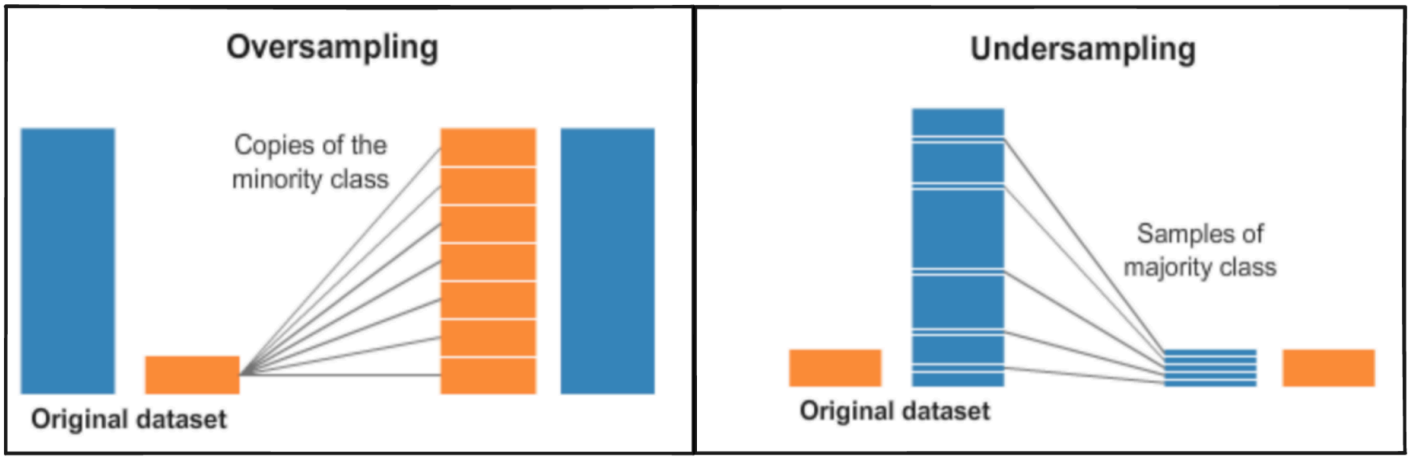
- 🤝 Machine Learning algorithms learn by example
  - 😓 If a class is underrepresented enough, the model will tend to predict the under-represented class poorly
  - ⚖️ A 70/30 ratio (class A / class B) split for binary classification can be considered imbalanced

Balancing strategies

- ***Oversampling*** of minority class
  - Alternatively, ***Computation of new instances*** for the minority class
- ***Undersampling*** of majority class

Oversampling or Undersampling

- ***Oversampling*** = duplicating instances of the minority class
- ***Undersampling*** = sampling down the majority class



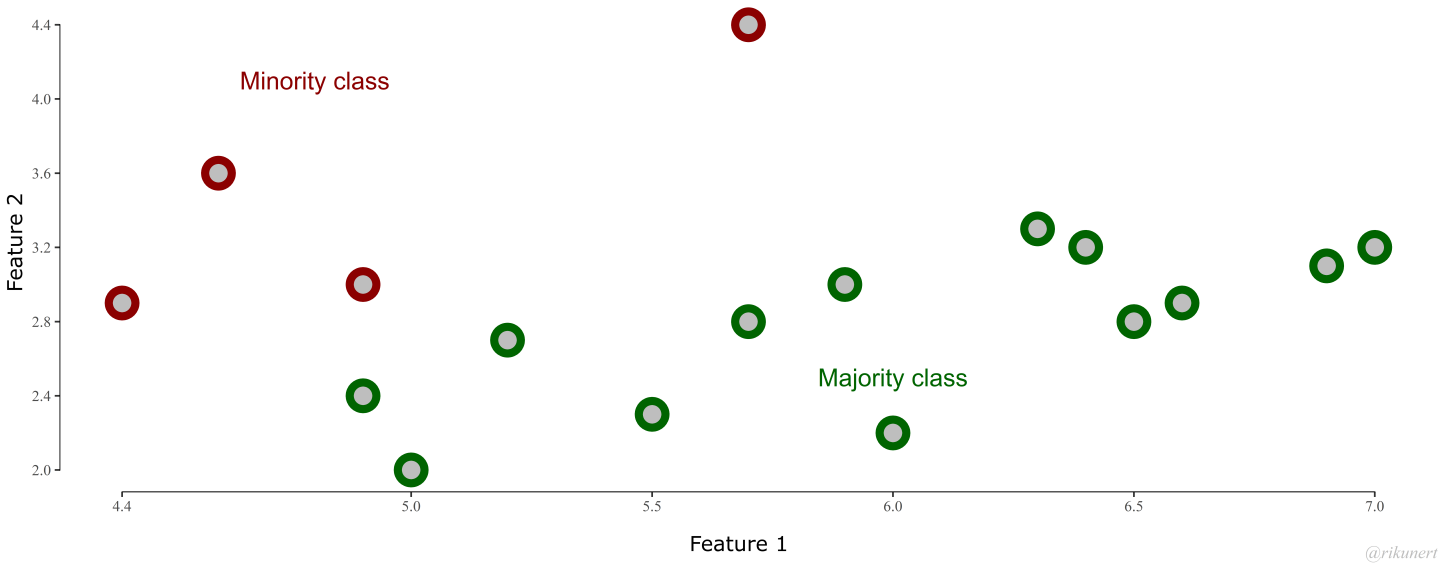
🔗 Source (https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets#t1)

❗ ***Warning about the oversampling method*** ❗

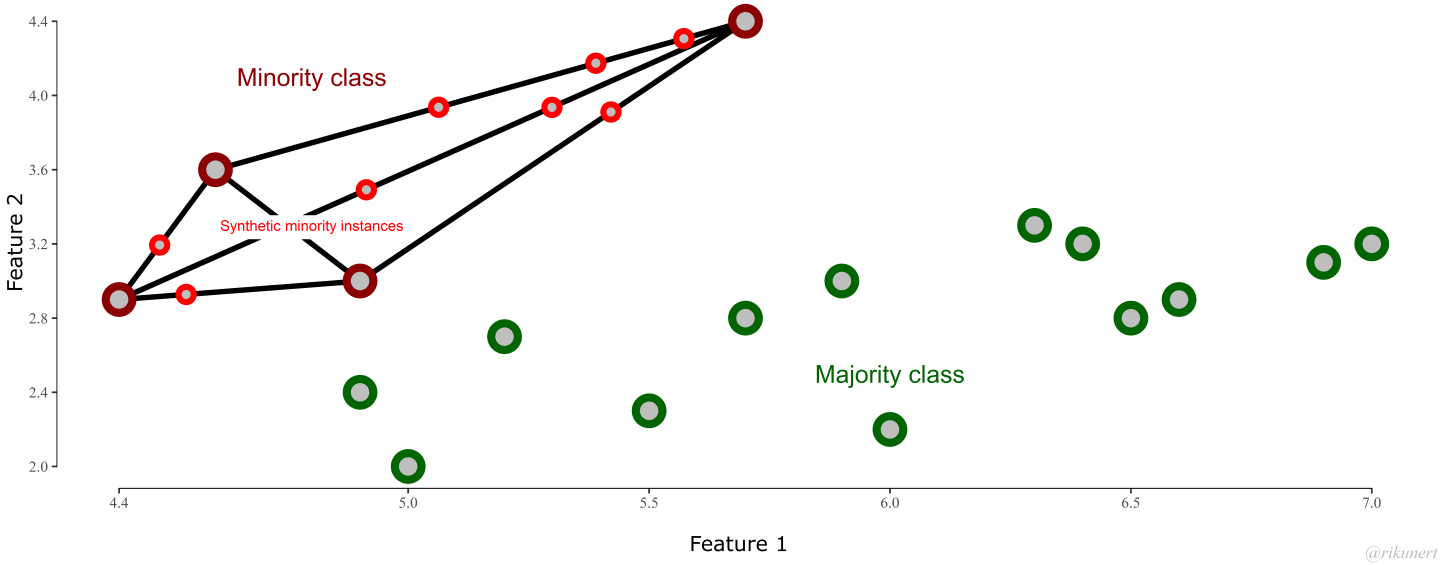
1. **Train**-**test** split your dataset *before* oversampling
2. **Oversample only in the train set**
   →
   The model needs to learn about the minority class
3. **Evaluate on the test set without oversampling**
   →
   We want the model to be evaluated in real conditions

Synthetic Minority Oversampling TEchnique (SMOTE)

> ***SMOTE is an oversampling algorithm that generates new minority instances from existing minority instances - based on linear combinations of existing points.***



🔗 Source (https://rikunert.com/SMOTE_explained)



📚 SMOTE documentation (https://imbalanced-learn.org/stable/over_sampling.html#over-sampling)

ℹ️ Notice that you have to `pip install imbalanced-learn`

👉 Imbalanced-learn (imported as `imblearn`) is an open source, MIT-licensed library relying on `scikit-learn` (imported as `sklearn`) and provides tools when dealing with classification with imbalanced classes.

🚨 ***Be careful with balancing techniques*** 🚨

- Use balancing techniques only on the training set to help the model learn about the minority class.
- The test set should remain representative of the real world.

(6) 🔡 Encoding

> 🔡 **Encoding consists in transforming non-numerical data into an equivalent numerical form.**

Why encoding?

- 🔡 Data may be represented as words, letters, or symbols
- 🤖 Most Machine Learning algorithms only process numerical data

(6.1) Feature Encoding with *OrdinalEncoder*

> ***OrdinalEncoder assigns a number to each category.***

📚 **sklearn.preprocessing.OrdinalEncoder** (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html)

👇 Look at the following feature "*classes*".

In [ ]:
```
example = pd.DataFrame({"classes":["bad", "average", "average", "goo
d", "good", "bad", "good"]})
example
```

Out[ ]:

	classes
0	bad
1	average
2	average
3	good
4	good
5	bad
6	good

🤔 How would a Machine Learning algorithm understand these classes?

🥷 Let's convert this column into numbers!

In [ ]:
```
from sklearn.preprocessing import OrdinalEncoder

# Instantiate the Ordinal Encoder
ordinal_encoder = OrdinalEncoder()

# Fit it
ordinal_encoder.fit(example[["classes"]])

# Display the learned categories
display(ordinal_encoder.categories_)

# Transform categories into ordered numbers
example["encoded_classes"] = ordinal_encoder.transform(example[["class
es"]])

# Show the transformed classes
example
```

```
[array(['average', 'bad', 'good'], dtype=object)]
```

Out[ ]:

	classes	encoded_classes
0	bad	1.0
1	average	0.0
2	average	0.0
3	good	2.0
4	good	2.0
5	bad	1.0
6	good	2.0

👆 We would expect something such as `bad`
↔
0, `average`
↔
1 and `good`
↔
2, right ?

💡 You can specify it in the Ordinal Encoder with `categories`

In [ ]:
```
from sklearn.preprocessing import OrdinalEncoder

# Instantiate the Ordinal Encoder
ordinal_encoder = OrdinalEncoder(categories=[["bad","average","goo
d"]])

# Fit it
ordinal_encoder.fit(example[["classes"]])

# Display the learned categories
display(ordinal_encoder.categories_)

# Transform categories into ordered numbers
example["encoded_classes"] = ordinal_encoder.transform(example[["class
es"]])

# Show the transformed classes
example
```

```
[array(['bad', 'average', 'good'], dtype=object)]
```

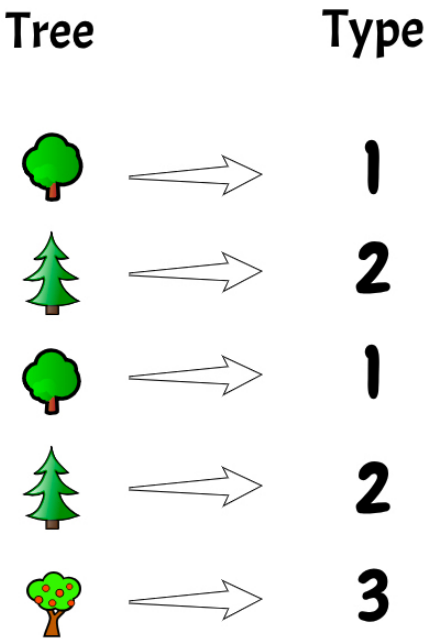
Out[ ]:

	classes	encoded_classes
0	bad	0.0
1	average	1.0
2	average	1.0
3	good	2.0
4	good	2.0
5	bad	0.0
6	good	2.0

🤔 But what if we cannot rank the different categories?

🌲 Have a look at the following illustration:

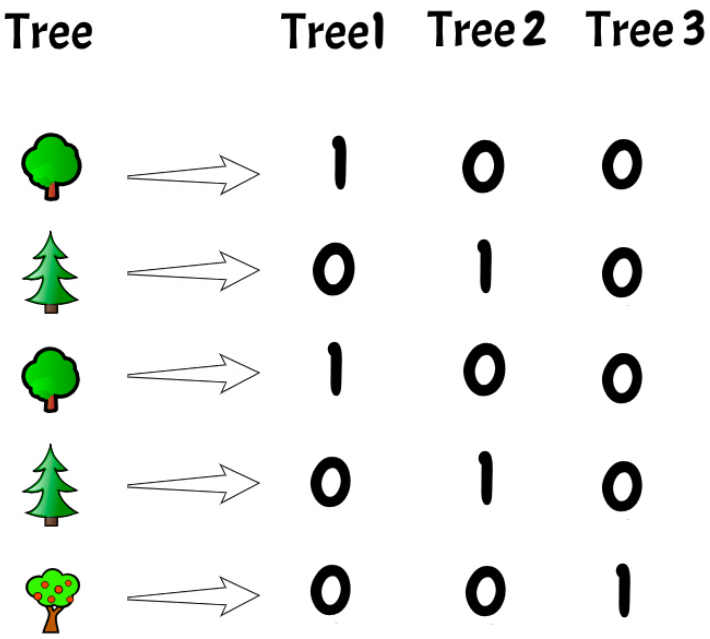
- If we were to use the *`OrdinalEncoder`*, we would create a false relationship between the different types of trees.
- How do we overcome this problem?



(6.2) Feature Encoding with *OneHotEncoder*

👉 Create a binary column for each possible category. This is also known as **One Hot Encoding**.

📚 Sklearn `OneHotEncoder()` documentation (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html)



💻 One-hot-Encoding `Alley`

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder
import numpy as np

# Check unique values for streets (3)
print(f"The unique values for 'Alley' are {data.Alley.unique()}")

# Instantiate the OneHotEncoder
ohe = OneHotEncoder(sparse_output=False)

# Fit encoder
ohe.fit(data[['Alley']])

# Display the detected categories
print(f"The categories detected by the OneHotEncoder are {ohe.categori
es_}")
```

```
The unique values for 'Alley' are ['NoAlley' 'Grvl' 'Pave']
The categories detected by the OneHotEncoder are [array(['Grvl', 'No
Alley', 'Pave'], dtype=object)]
```

In [ ]:
```
# Display the generated names
print(f"The column names for the encoded values are {ohe.get_feature_n
ames_out()}")

# Transform the current "Alley" column
data[ohe.get_feature_names_out()] = ohe.transform(data[['Alley']])

# Drop the column "Alley" which has been encoded
data = data.drop(columns=["Alley"])

# Show the dataset
data.head(3)
```

```
The column names for the encoded values are ['Alley_Grvl' 'Alley_NoA
lley' 'Alley_Pave']
```

Out[ ]:

	GrLivArea	BedroomAbvGr	KitchenAbvGr	OverallCond	Pesos	Street	SalePrice	Alley_(
0	0.380216	3	1	5	4170000.0	Pave	208500	
1	-0.312210	3	1	8	3630000.0	Pave	181500	
2	0.497682	3	1	5	4470000.0	Pave	223500	

💻 One-hot Encoding `Street`

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder

# Check unique values for streets (2)
print(f"The unique values for 'Street' are {data.Street.unique()}")

# Instantiate the OneHotEncoder
ohe_binary = OneHotEncoder(sparse_output=False, drop="if_binary")

# Fit encoder
ohe_binary.fit(data[['Street']])

# Display the detected categories
print(f"The categories detected by the OneHotEncoder are {ohe_binary.c
ategories_}")
```

```
The unique values for 'Street' are ['Pave' 'Grvl']
The categories detected by the OneHotEncoder are [array(['Grvl', 'Pa
ve'], dtype=object)]
```

In [ ]:
```
# Display the generated names
print(f"The column names for the encoded values are {ohe_binary.get_fe
ature_names_out()}")

# Transform the current "Street" column
data[ohe_binary.get_feature_names_out()] = ohe_binary.transform(dat
a[['Street']])

# Drop the column "Street" which has been encoded
data = data.drop(columns=["Street"])

# Show the dataset
data.head(3)
```

```
The column names for the encoded values are ['Street_Pave']
```

Out[ ]:

	GrLivArea	BedroomAbvGr	KitchenAbvGr	OverallCond	Pesos	SalePrice	Alley_Grvl	All
0	0.380216	3	1	5	4170000.0	208500	0.0	
1	-0.312210	3	1	8	3630000.0	181500	0.0	
2	0.497682	3	1	5	4470000.0	223500	0.0	

(6.3) *LabelEncoder*

We talked about how to encode categorical features. But now...

🎯 What about a classification task where you would need to ***encode categorical targets***?

🐠 Let's say that we want to predict the species of penguins.

In [ ]:
```
import seaborn as sns

penguins = sns.load_dataset("penguins")
penguins.head()
```

Out[ ]:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female

In [ ]:
```
target = penguins["species"]
target.value_counts()
```

Out[ ]:
```
Adelie       152
Gentoo       124
Chinstrap     68
Name: species, dtype: int64
```

📚 **sklearn.preprocessing.LabelEncoder** (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html)

In [ ]:
```
from sklearn.preprocessing import LabelEncoder

# Instantiate the LabelEncoder
label_encoder = LabelEncoder()

# Fit it to the target
label_encoder.fit(target)

# Find the encoded classes
print(f"The Label Encoder has encoded the penguin classes into {label_
encoder.classes_}")

# Transform the targets
encoded_target = label_encoder.transform(target)
```

```
The Label Encoder has encoded the penguin classes into ['Adelie' 'Ch
instrap' 'Gentoo']
```

In [ ]:
```
# Showing the target and the encoded target side by side
pd.DataFrame({"target": target, "encoded_target": encoded_target}).sam
ple(10)
```

Out[ ]:

	target	encoded_target
128	Adelie	0
275	Gentoo	2
215	Chinstrap	1
141	Adelie	0
131	Adelie	0
161	Chinstrap	1
265	Gentoo	2
50	Adelie	0
25	Adelie	0
314	Gentoo	2

💡 You can revert back to the original target:

In [ ]:
```
original_target = label_encoder.inverse_transform(encoded_target)
```

In [ ]:
```
# Showing the encoded target and the original target side by side
pd.DataFrame({"encoded_target": encoded_target, "original_target": ori
ginal_target, "target": target}).sample(10)
```

Out[ ]:

	encoded_target	original_target	target
302	2	Gentoo	Gentoo
154	1	Chinstrap	Chinstrap
279	2	Gentoo	Gentoo
134	0	Adelie	Adelie
332	2	Gentoo	Gentoo
85	0	Adelie	Adelie
99	0	Adelie	Adelie
293	2	Gentoo	Gentoo
261	2	Gentoo	Gentoo
95	0	Adelie	Adelie

Now, that was quite a tedious process 😩

💡 **Fortunately in most cases you don't have to encode and decode the targets!**

Most models in Scikit Learn can handle **targets** that have not been encoded.

These models handle the label encoding inside the model for us.

So, before you start encoding your targets (and decoding them back),
check if encoding is really needed!

First try your Scikit Learn model without encoding the targets, it will probably work.

📆 So where are we now ?

- ✅ (1) 👥 Duplicates
- ✅ (2) ⁉️ Missing data
- ✅ (3) 🐋 Outliers
- ✅ (4) 🔢 Feature scaling
- ✅ (5) 🔠 Encoding
- ✅ (6) ⚖️ Dataset balancing

👉 We need to talk about:

- 🟨 (7) Discretizing
- 🟨 (8) Feature creation

🔥 And the final and crucial section of this lecture:

- 🤖 (9) Feature selection, Modelling and Feature permutation

(7) 🟨 Discretizing

> **Discretizing is the process of turning continuous data into discrete data using bins.**

- 🪄 Performs feature engineering
- 🎯 Turns a regression task into a classification task

📚 **Pandas `cut()` documentation** (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.cut.html)

🖥️ Discretizing Sale Price

If we want to qualify a house as *Cheap* or *Expensive* for example, instead of predicting its price, we can prepare this dataset for a classification task.

Let's separate houses into either *Cheap* or *Expensive*, according to the average price of houses.

In [ ]:
```
data['SalePriceBinary'] = pd.cut(x=data['SalePrice'],
                                 bins=[data['SalePrice'].min()-1,
                                       data['SalePrice'].mean(),
                                       data['SalePrice'].max()+1],
                                 labels=['cheap', 'expensive'])

data.head()
```

Out[ ]:

	GrLivArea	BedroomAbvGr	KitchenAbvGr	OverallCond	Pesos	SalePrice	Alley_Grvl	All
0	0.380216	3	1	5	4170000.0	208500	0.0	
1	-0.312210	3	1	8	3630000.0	181500	0.0	
2	0.497682	3	1	5	4470000.0	223500	0.0	
3	0.391036	3	1	5	2800000.0	140000	0.0	
4	1.134467	4	1	5	5000000.0	250000	0.0	

(8) 🟨 Feature creation

🥷 We can introduce some domain knowledge into a dataset in order to drive more signals for our models to learn!

Why create new features?

- ➕ Create additional information
- 📈 Potentially improve model performance

Examples of creating new features

- 🛌 $\frac{bedroom}{totalroom}$ ratio
- 🏥 $BodyMassIndex = \frac{height}{weight^2}$
- 🛍️ `delivered_date - dispatch_date` for lag time between events
- 📆 Categorize `date` as either `weekday` or `weekend`

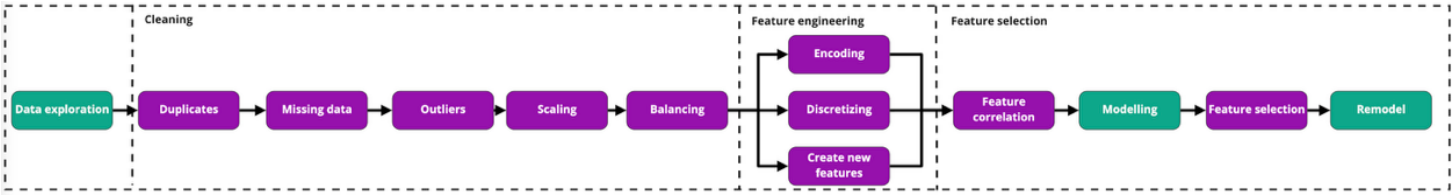
🤦 Encoding, discretizing and creating new features fall under a category of preprocessing known as **feature engineering**.

👉 Scikit Learn has many more processing tools!

📚 **sklearn.preprocessing** (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing)

(9) 🤖 Feature selection, Modelling and Feature Permutation

Prepare the dataset - All the steps



🔎 Click to zoom in (https://github.com/lewagon/data-images/raw/master/ML/preparation_steps.pdf)

> **Feature selection is the process of eliminating "non-informative" features.**

👇 There are 2 main types of statistical feature selection:

- Univariate feature selection
  →
  `Feature Correlation` (Univariate)
- Multivariate
  →
  `Features' multicollinearity` (Multivariate)

(9.1) Why feature selection?

- 🚮 Garbage in
  →
  garbage out
- 💥 The curse of dimensionality
- 🤯 Reducing complexity

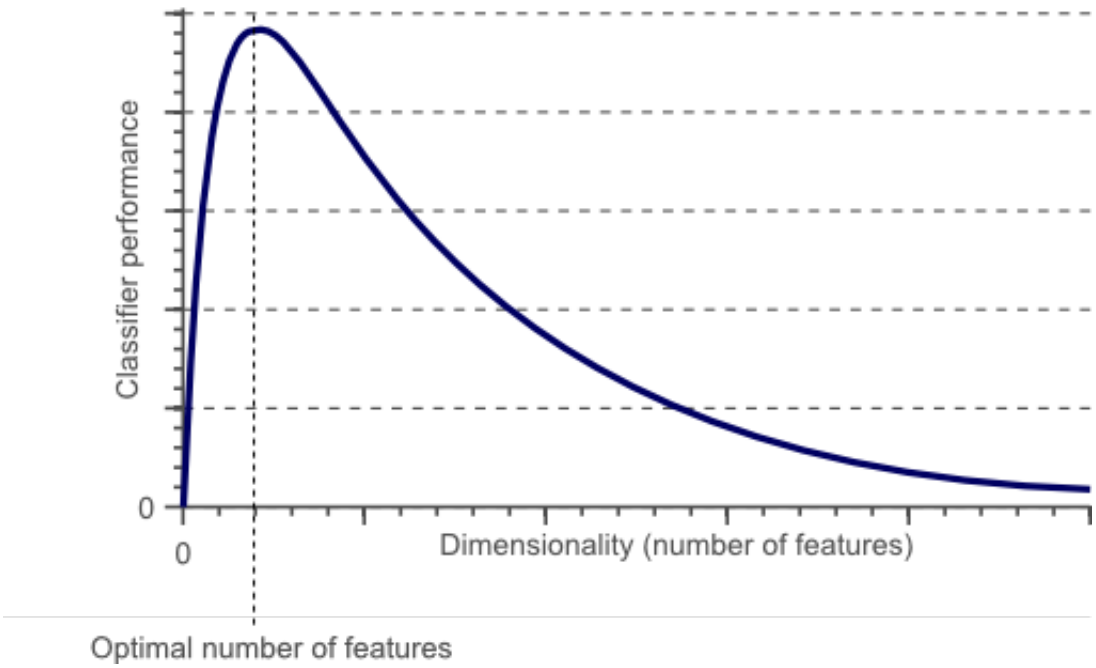
🚮 Garbage in
→
garbage out

Poor quality input will:

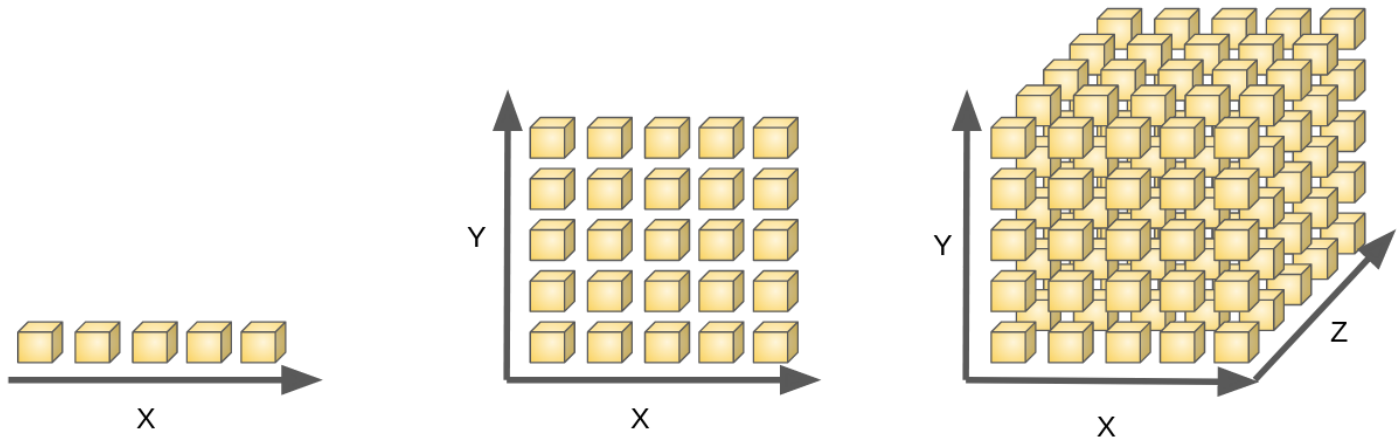
1. Induce noise
2. Destabilize the model
3. Generate unusable output

💥 The curse of dimensionality

Not observing enough data to support a meaningful relationship.



🔗Source (https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/)



As the number of features or dimensions grows, the amount of data we need to generalize a model accurately grows **exponentially** e.g.
5^1
,
5^2
,
5^3
, ...,
5^n
.

🔗More detail (https://livebook.manning.com/concept/r/dimensionality)

🚨 ***Warning about OneHotEncoder: be careful about which features you encode!***

- High variations within a categorical feature will generate more binary columns...
  - Consequently, you would need more data points
  - *Spoiler* : this is called "Curse of dimensionality" (https://en.wikipedia.org/wiki/Curse_of_dimensionality)

(9.2) Feature correlation

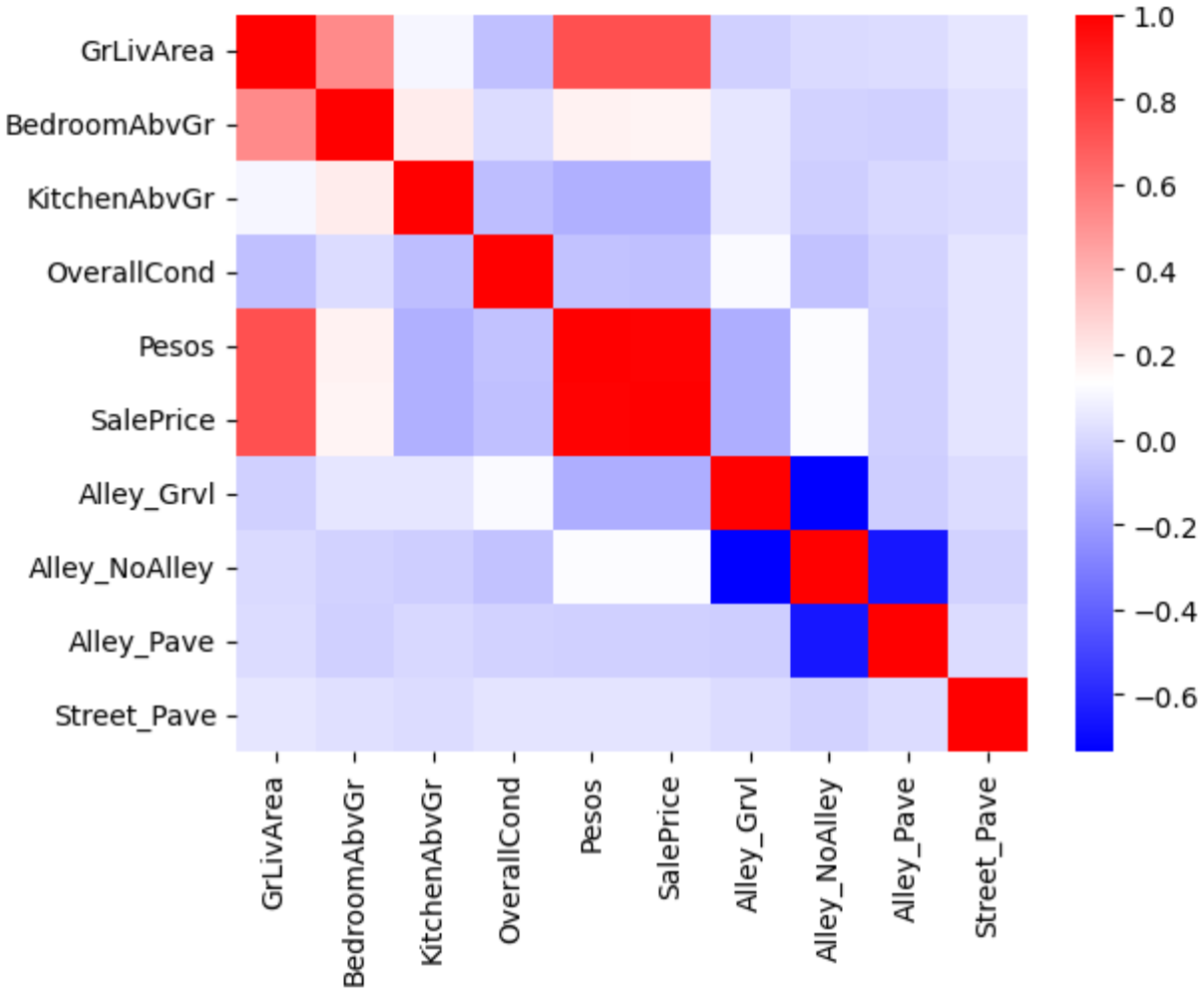
🤷 One of the feature selection techniques is to remove one of two features that are highly correlated to each other.

❗ High correlation between feature A and feature B $\rightarrow$ redundant information.

🖥️ Pearson Correlation

In [ ]:
```
import seaborn as sns

# Heatmap of pairwise correlations
correlation_matrix = data.select_dtypes('number').corr()
column_names = correlation_matrix.columns
sns.heatmap(correlation_matrix, xticklabels=column_names, yticklabels=
column_names,cmap= "bwr");
```



📚 **matplotlib > colors > colormaps** (https://matplotlib.org/stable/tutorials/colors/colormaps.html)

Let's turn the correlation matrix into a DataFrame.

In [ ]:
```
# Convert the correlation matrix into a DataFrame
corr_df = correlation_matrix.stack().reset_index()

# Rename the columns
corr_df.columns = ['feature_1','feature_2', 'correlation']

# Remove "self correlations"
no_self_correlation = (corr_df['feature_1'] != corr_df['feature_2'])
corr_df = corr_df[no_self_correlation]
```

Let's see which pairs of features are the most correlated (both positively and negatively)

In [ ]:
```
# Compute the absolute correlation
corr_df['absolute_correlation'] = np.abs(corr_df['correlation'])

# Show the top 5 most correlated pairs of feature
corr_df.sort_values(by="absolute_correlation", ascending=False).head(
5*2)
```

Out[ ]:

	feature_1	feature_2	correlation	absolute_correlation
54	SalePrice	Pesos	0.990353	0.990353
45	Pesos	SalePrice	0.990353	0.990353
76	Alley_NoAlley	Alley_Grvl	-0.734669	0.734669
67	Alley_Grvl	Alley_NoAlley	-0.734669	0.734669
50	SalePrice	GrLivArea	0.725634	0.725634
5	GrLivArea	SalePrice	0.725634	0.725634
40	Pesos	GrLivArea	0.724702	0.724702
4	GrLivArea	Pesos	0.724702	0.724702
87	Alley_Pave	Alley_NoAlley	-0.654782	0.654782
78	Alley_NoAlley	Alley_Pave	-0.654782	0.654782

💡 Hints:

- Remove as many of the "redundant" columns as you want, starting from those with the highest correlation.
- Keep doing it until your model's performance starts to drop significantly. At this point, you may have dropped too many features.

❓ Which pair of columns has the highest correlation ❓

The 🔢 feature `Pesos` is perfectly correlated to the 🎯 target `SalePrice`.

👀 What are we observing?

⚠️ Data Leakage

💵 `SalePrice` USD
~ 20
$\times$
`Pesos` (🔗 xe.converter/USD-to-MXN (https://www.xe.com/currencyconverter/convert/?Amount=1&From=USD&To=MXN))

🚮 We should drop the column `Pesos`

In [ ]:
```
data = data.drop(columns=['Pesos'])
```

(9.4) 🖥️ Modelling 🥳

🤷 Now that we have preprocessed our dataset, let's evaluate a classification model, the Logistic Regression (sklearn.linear_model.LogisticRegression).

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

Out[ ]:

	GrLivArea	BedroomAbvGr	KitchenAbvGr	OverallCond	SalePrice	Alley_Grvl	Alley_NoAlley
0	0.380216	3	1	5	208500	0.0	1.0
1	-0.312210	3	1	8	181500	0.0	1.0
2	0.497682	3	1	5	223500	0.0	1.0
3	0.391036	3	1	5	140000	0.0	1.0
4	1.134467	4	1	5	250000	0.0	1.0

In [ ]:
```
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

# Encode the target
target_encoder = LabelEncoder().fit(data['SalePriceBinary'])
y = target_encoder.transform(data['SalePriceBinary'])

# Define the features
X = data.drop(columns=['SalePrice', 'SalePriceBinary'])

# Scale numerical features
# Notice that we already RobutScaled GrLivArea
minmax_scaler = MinMaxScaler()
X[["BedroomAbvGr","KitchenAbvGr","OverallCond"]] = minmax_scaler.fit_t
ransform(X[["BedroomAbvGr","KitchenAbvGr","OverallCond"]])

# Instantiate a model
log_reg = LogisticRegression(max_iter=1000)

# Score on multiple folds aka Cross Validation
scores = cross_val_score(log_reg, X, y, cv=10)
scores.mean()
```

Out[ ]:
```
0.8308455361360416
```

⚠️ Data Leakage

- $\geq$
  0.80 seems like a good accuracy score...
  $\rightarrow$
  If we have 100 houses, we are able to classify ~80 of them correctly.
- But actually, we have just committed the sin of data leakage, can you say which one and why?

🚨 One should never apply transformations on the entire dataset! 🚨

☠️ *Why should you scale the numerical features only on the training set?*

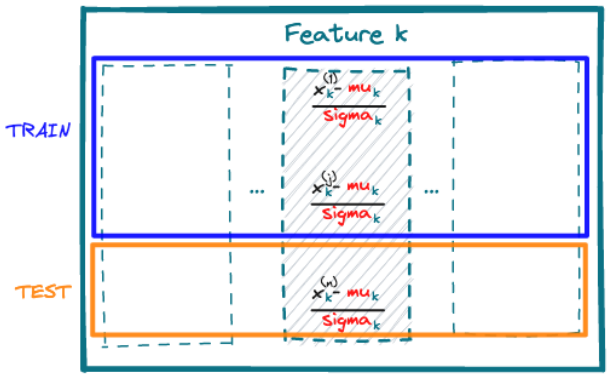
❌ Wrong Scenario - Data leakage

(1) Scaling Feature k

- **fit** ● Computing mu_k and $Sigma_k$
- **transform** ● Scaling all the values $x \rightarrow z = \frac{x_k - mu_k}{Sigma_k}$

(2) Train/Test split

- If you scale before the holdout method, both the train set and the test set will contain mu and sigma → **Data Leakage**



mu_k and $Sigma_k$ are LEAKED in the test set 😱 because they were calculated using the entire dataset 👎👎👎

❌ Wrong Scenario - Fitting a scaler twice

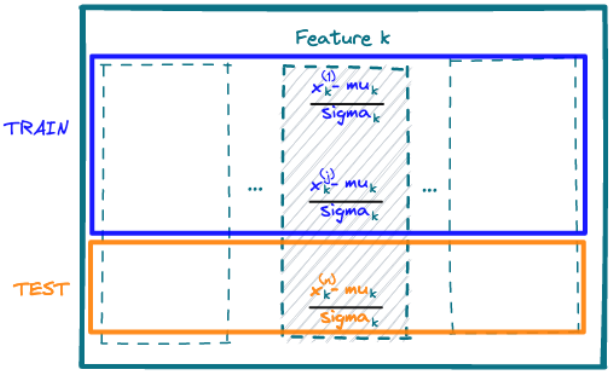
(1) Train/Test split

- Let's use the holdout method first to create the train set and the test set

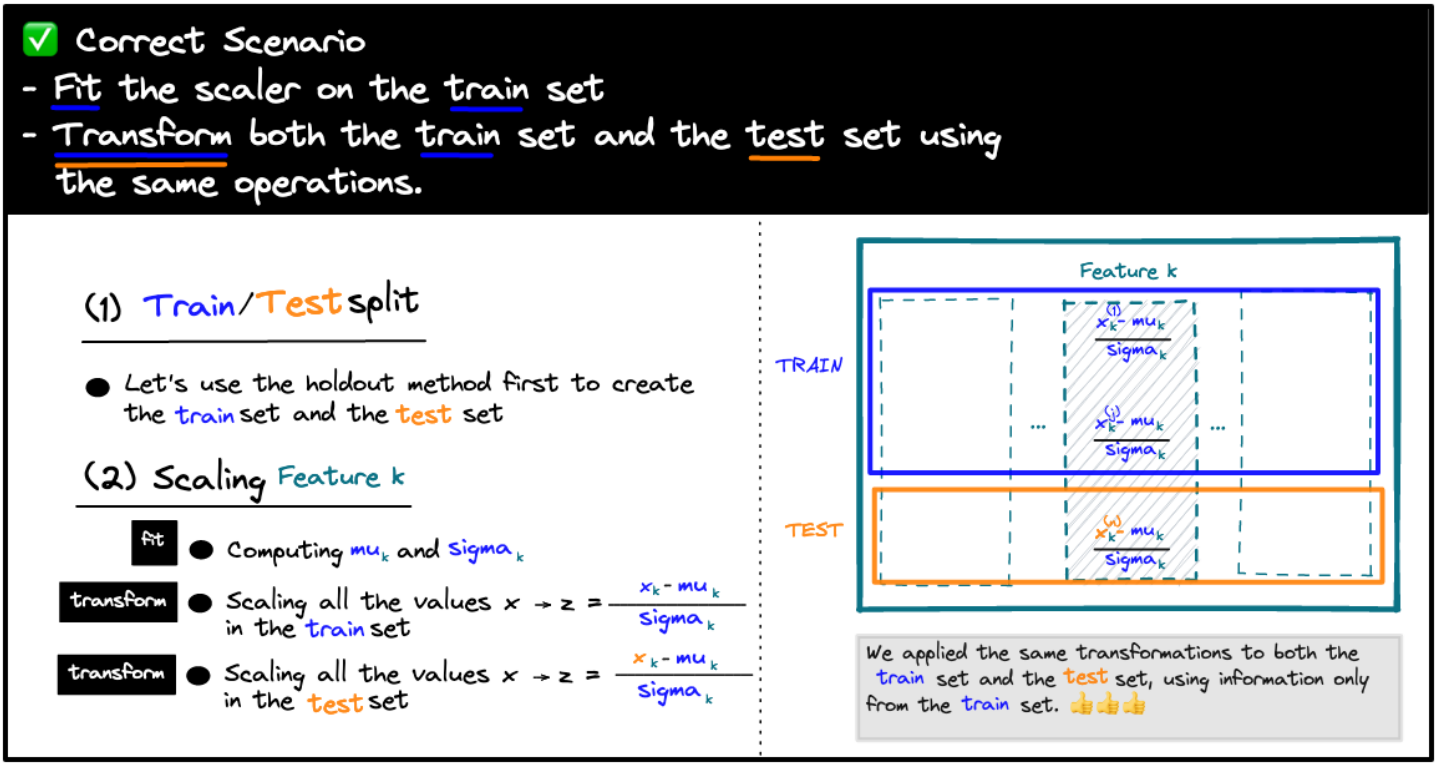
(2) Scaling Feature k

- **fit** ● Computing mu_k and $Sigma_k$
- **transform** ● Scaling all the values $x \rightarrow z = \frac{x_k - mu_k}{Sigma_k}$ in the train set
- **fit** ● Computing mu_k and $Sigma_k$
- **transform** ● Scaling all the values $x \rightarrow z = \frac{x_k - mu_k}{Sigma_k}$ in the test set

→ **Different transformations....**



⛽ The transformations on the train set and the test set should be the same ! 👎👎👎



🚨 *The problem is that we scaled before doing a train-test split for each of the 5 submodels of the cross validation...*

🤔 *How to prevent data leakage derived from scaling during CV ❓*

How to prevent data leakage derived from scaling during CV?

Option 1: Manual K-fold Cross Validation

1. Split your dataset into folds
2. Run a `for` loop with the following steps:

    - Define `X_train`, `X_val`, `y_train`, `y_val`
    - Fit your scaler on the train set
    - Transform both your train set and your validation set
    - Train your model on the train set
    - Evaluate it on the validation set

🤦 This manual K-fold Cross-Validation is tedious, boring, but easy to code.

🔗 See this gist (https://gist.github.com/1fd64f3dfb9bec3a9cea179e6642443a)

How to prevent data leakage derived from scaling during CV?

Option 2: Pipelines

- Cf. `Machine Learning > Unit 6 - Workflow`

(9.5) 🏇 Feature Permutation

> ***Feature Permutation is a feature selection "algorithm" which evaluates the importance of each feature in predicting the target.***

How does `Feature Permutation` work?

1️⃣ 🐴 **Trains** a base model containing all the features and **records the test score**.

2️⃣ 🎲 `Permutation` randomly **shuffles a feature** within the **test** set.

3️⃣ 🆕 **Records the new score on the test set** with the **shuffled feature**

4️⃣ 🕵️ Compares the new score to the original score. If the score dropped significantly, it means that this feature is important and that we shouldn't have shuffled it!

5️⃣ 🔁 Repeat steps 2-3-4 for each feature.

📚 **sklearn.inspection.permutation_importance** (https://scikit-learn.org/stable/modules/generated/sklearn.inspection.permutation_importance.html)

💻 Feature permutation in Sklearn

- Score without Permutation:

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

# Model to be cross-validated
log_model = LogisticRegression()

# Cross Validation
np.mean(cross_val_score(log_model, X, y , cv=5))
```

Out[ ]:
```
0.829553264604811
```

- Permutation:

In [ ]:
```
from sklearn.inspection import permutation_importance

# Fit model
log_model = LogisticRegression().fit(X, y)

# Perform the permutation
permutation_score = permutation_importance(log_model, X, y, n_repeats=
10)

# Unstack results showing the decrease in performance after shuffling
features
importance_df = pd.DataFrame(np.vstack((X.columns,
                                        permutation_score.importances_
mean())).T)
importance_df.columns=['feature','score decrease']

# Show the important features
importance_df.sort_values(by="score decrease", ascending=False)
```

Out[ ]:

	feature	score decrease
0	GrLivArea	0.299863
1	BedroomAbvGr	0.025842
2	KitchenAbvGr	0.013265
5	Alley_NoAlley	0.010241
4	Alley_Grvl	0.004742
3	OverallCond	0.003918
7	Street_Pave	0.000687
6	Alley_Pave	-0.000687

- Model with the strongest features:

In [ ]:
```
# Selecting the strongest features
strongest_features = X[["GrLivArea", "BedroomAbvGr"]]

# Re-instantiating a Logistic Regression
log_reg = LogisticRegression()

# Average accuracy of the cross-validated model
np.mean(cross_val_score(log_reg, strongest_features, y, cv=10))
```

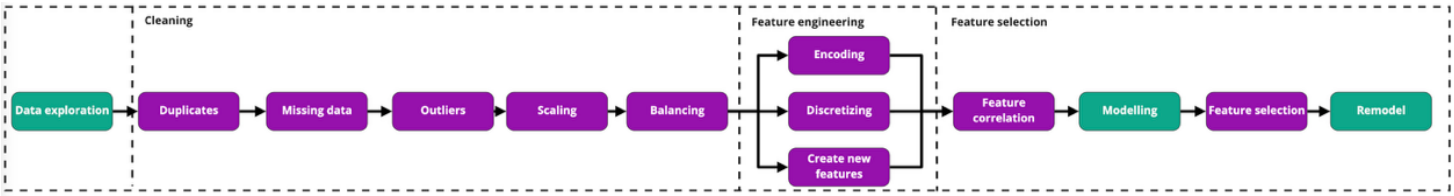
Out[ ]: 0.8136419461502126

(9.6) Reducing Complexity

- A few words about complexity:
  - A complex model (in terms of features and/or algorithms) is not always the best solution
  - Trust the 20/80 **Pareto Law** (https://en.wikipedia.org/wiki/Pareto_principle)

- Reducing the number of features makes the model:
  - More interpretable
  - Faster to train
  - Easier to implement and maintain in production

Prepare the dataset - Wrap up



🔎 Click to zoom in (https://github.com/lewagon/data-images/raw/master/ML/preparation_steps.pdf)

🚨 Whatever steps and strategies we choose to prepare the training set, we must apply exactly the same **transformations** to the test set, or to any new data point we want our model to predict.

🚨 Do not `fit` scalers/encoders on the test data! Use the learnings from fitting on the train set!

```
# Instantiate your Scaler or your Encoder
transformer = Transformer()

# Fit it on the training set (or on specific columns)
transformer.fit(X_train)

# Apply the same transformations to both the train set and the test set
X_train_transformed = transformer.transform(X_train)
X_test_transformed = transformer.transform(X_test)
```

🚨 When preparing a dataset, we **introduce human biases**.

- We preprocess it using strategies we believe to be the best possible strategies.
- This can be *harmful and non-inclusive* even if we didn't have bad intentions!

Your Turn! 🚀