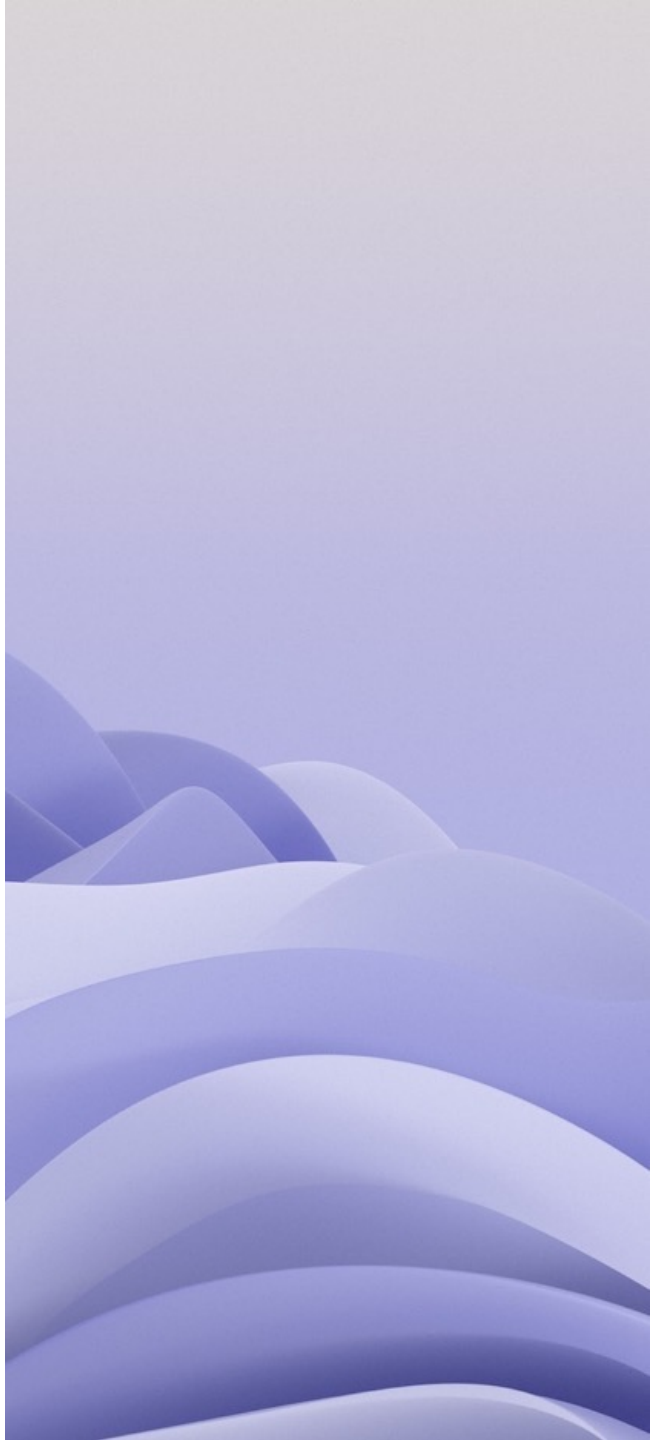
Friday 8th December 2023

Big Data

Data Mining, warehouse, viz, nosql

applied to health data

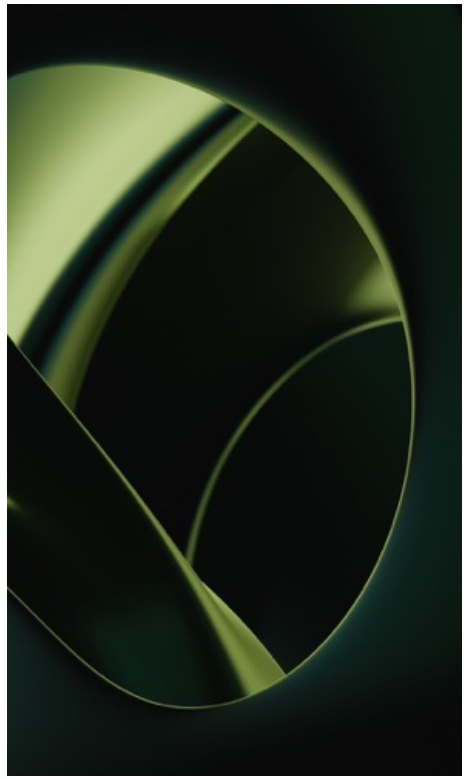


Agenda

Data streamin introduction

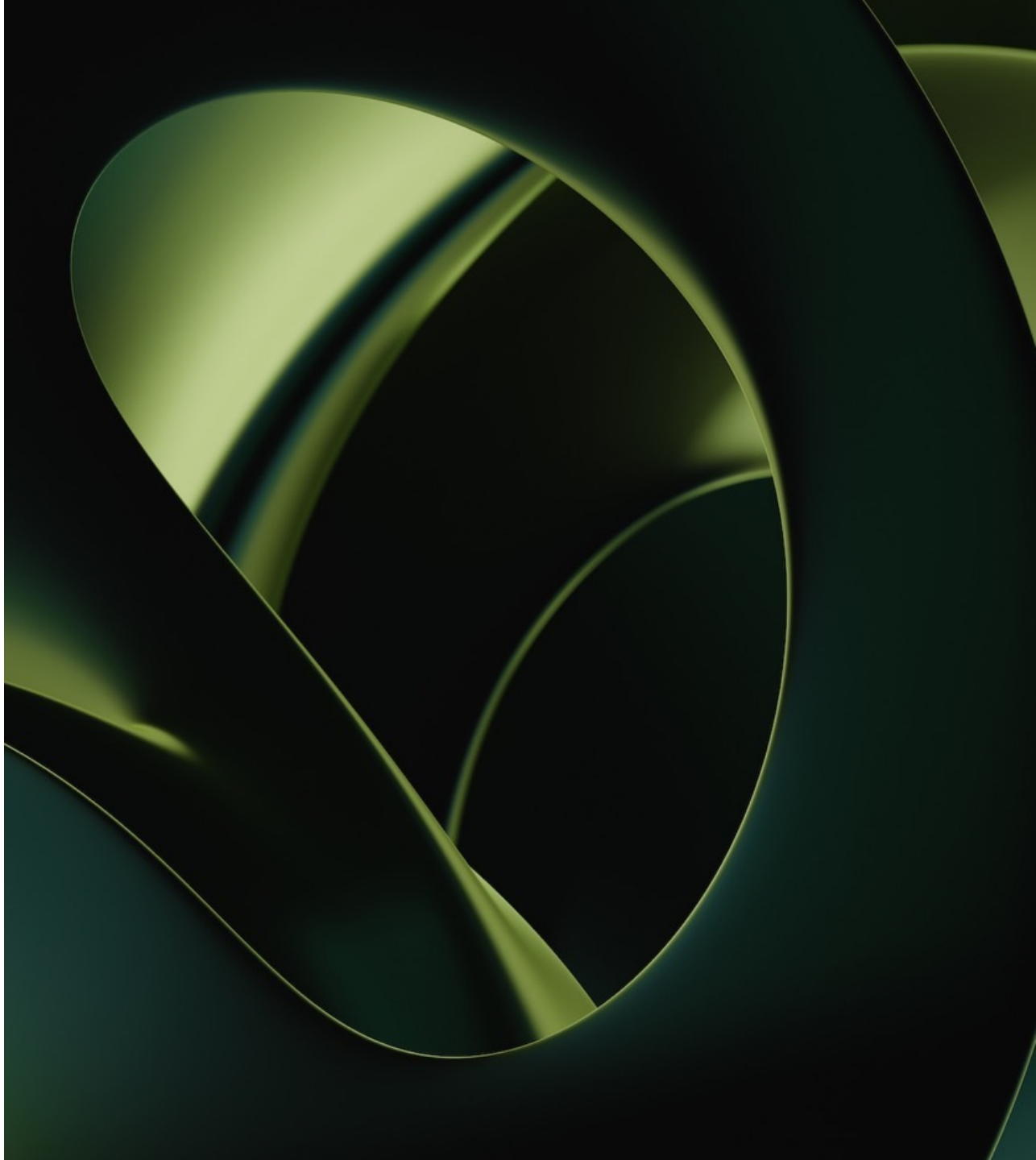
Hands-On

Pandas crash course



Introduction to data mining

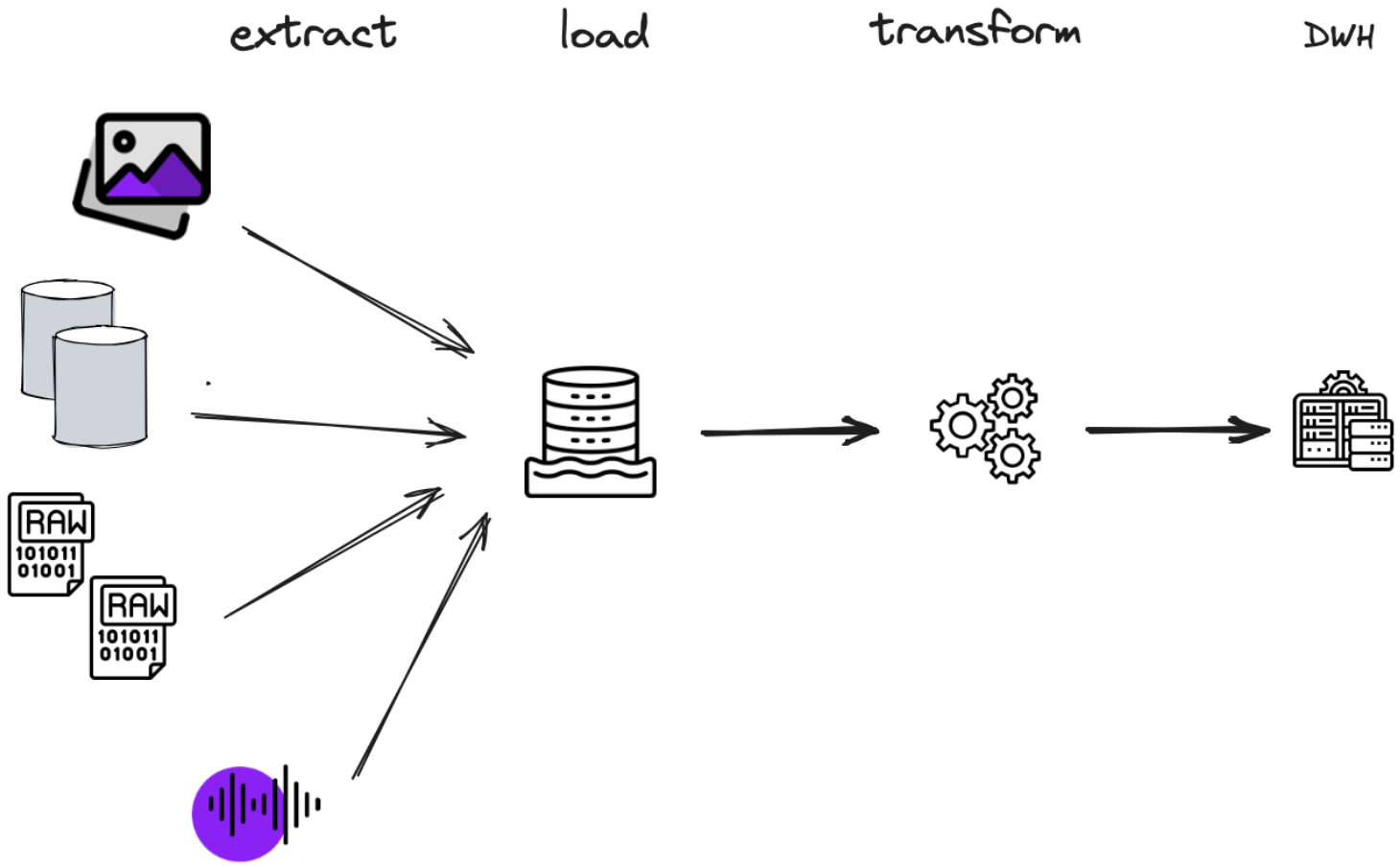
Data Mining introduction



saegus
SMART SHIFT · FOR REAL

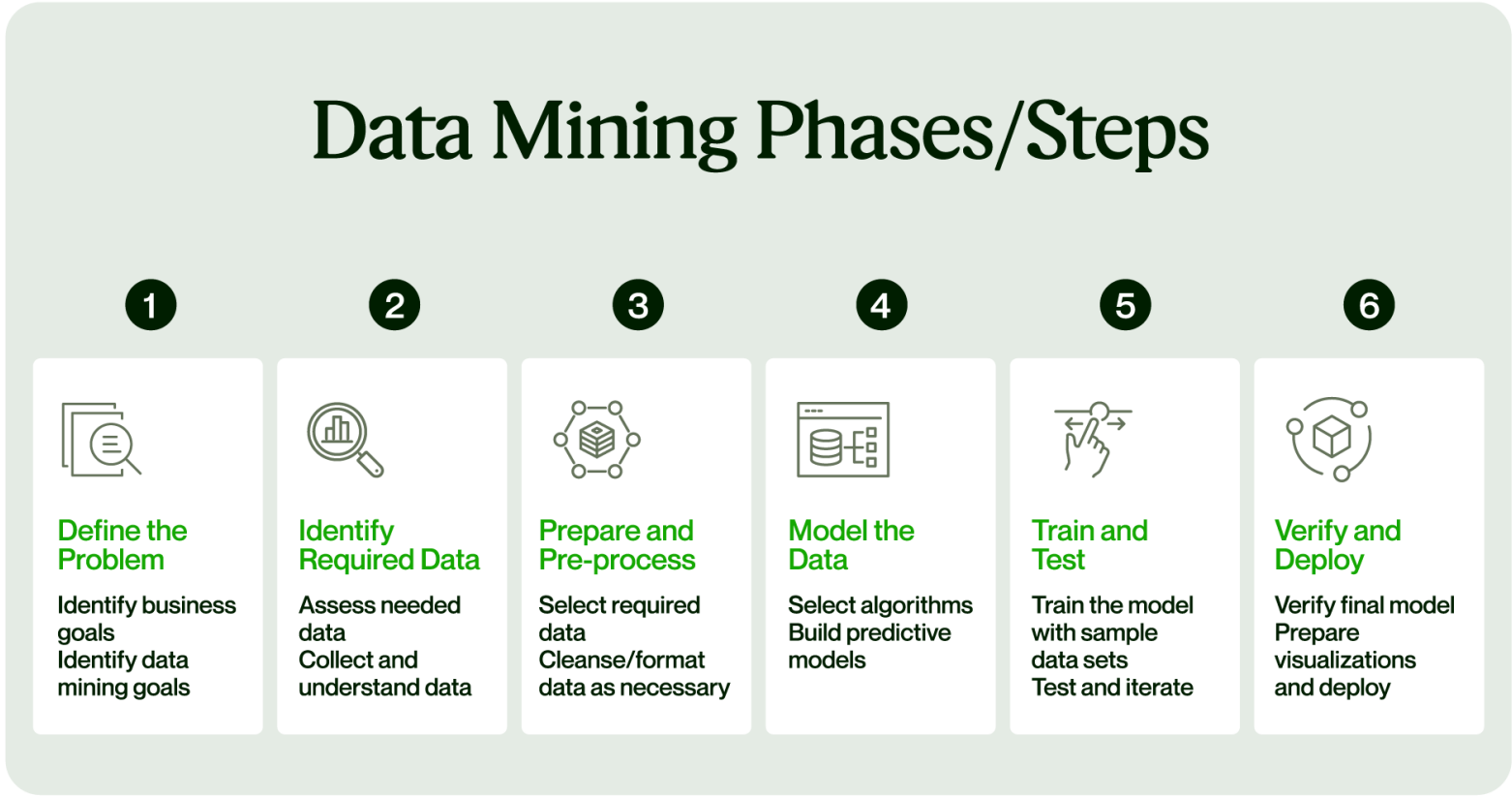
Data mining Introduction

Now what ?



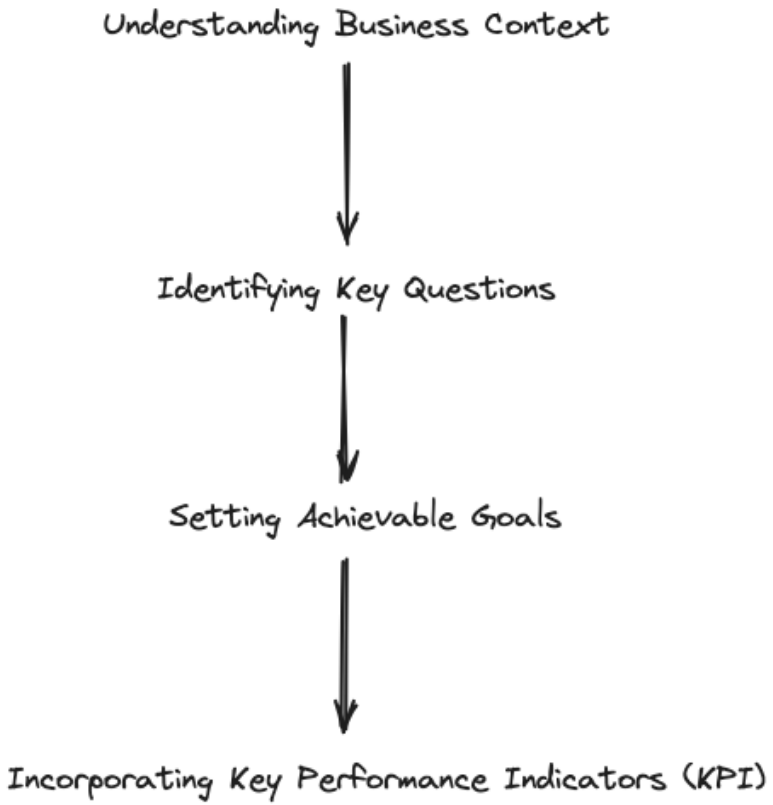
Data mining Introduction

Now what ?



Define the problem

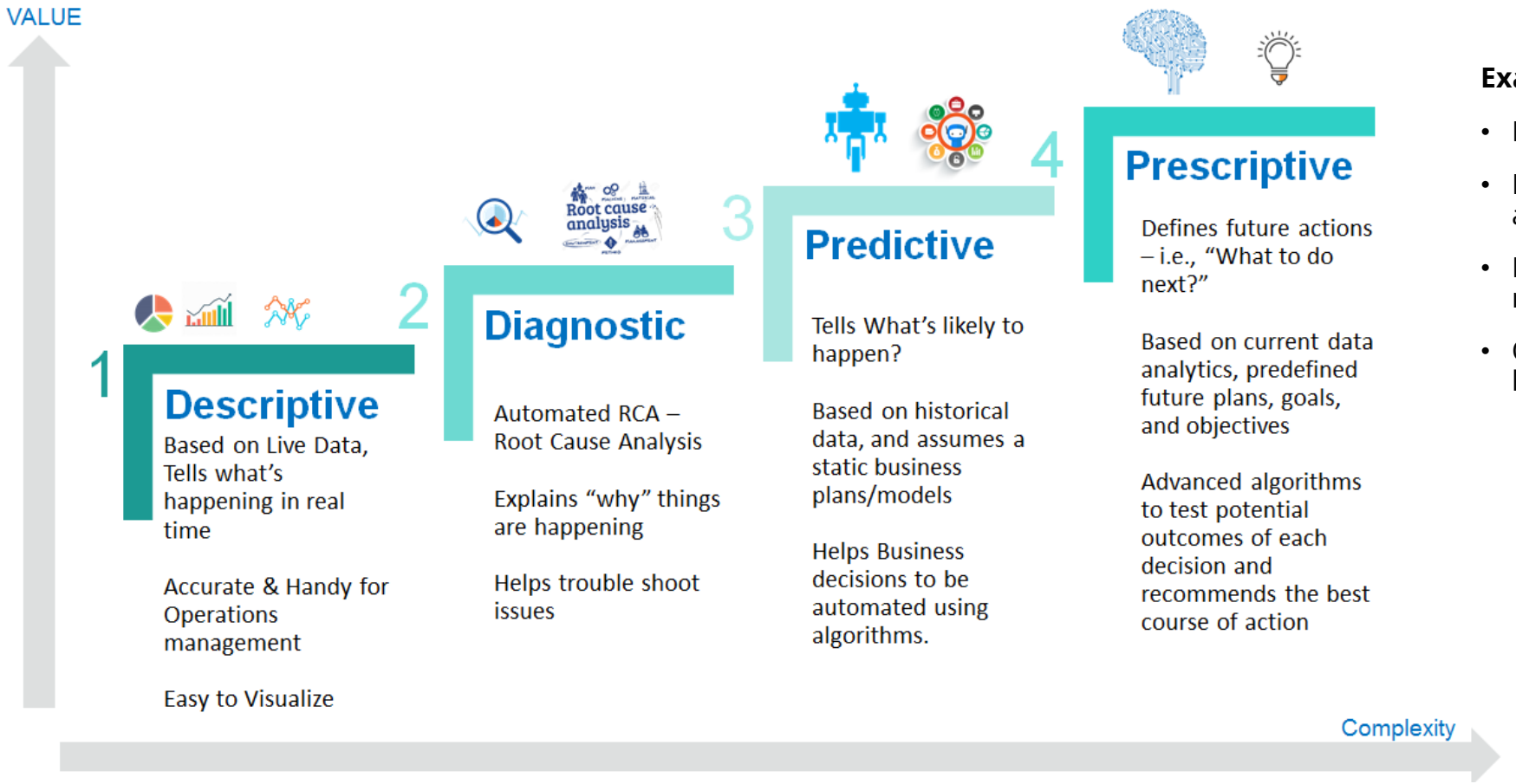
Stage 1 : define business objective



Examples :

- Analyze sales trends from past year data.
- Identify factors to improve customer satisfaction.
- Predict next quarter's revenue based on current trends.
- Group patients based on treatment response in healthcare data.

Types of Questions Answered by Data Mining



Examples :

- Predict patient readmission risks.
- Identify customer segments for targeted advertising
- Forecast stock price trends based on market data.
- Optimize supply chain logistics using historical data

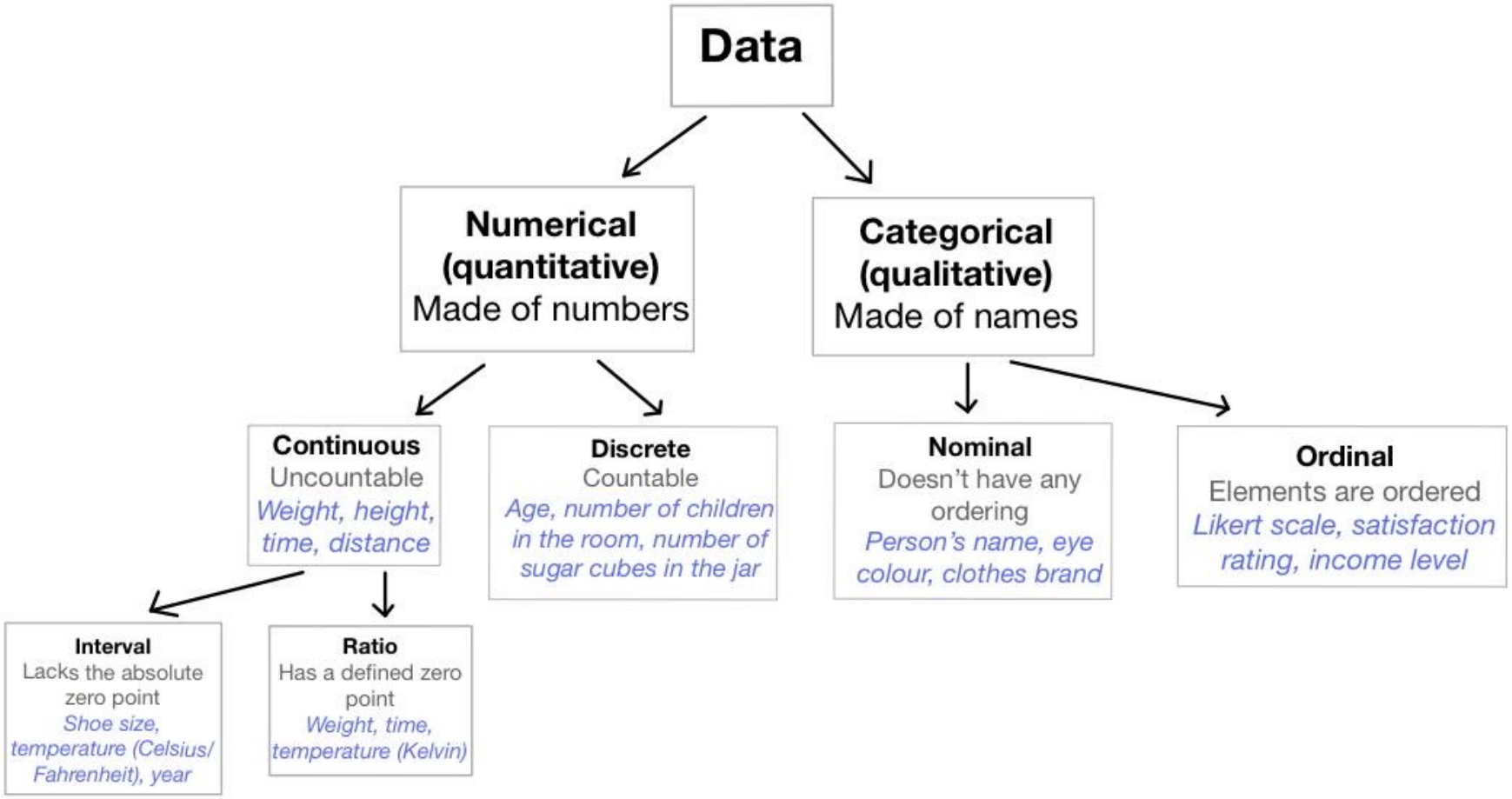
Data Mining data sources

Data Definition Framework

Data Format

Data Source	Structured	Unstructured
Internal	**Human-Generated** • Survey ratings • Aptitude testing **Machine-Generated** • Web metrics from Web logs • Product purchase from sales Records • Process control measures	**Human-Generated** • Emails, letters, text messages • Audio transcripts • Customer comments • Voicemails • Corporate video/communications • Pictures, illustrations • Employee reviews
External	**Human-Generated** • Number of Retweets, Facebook likes, Google Plus +1s • Ratings on Yelp • Patient ratings ratings **Machine-Generated** • GPS for tweets • Time of tweet/updates/postings	**Human-Generated** • Content of social media updates • Comments in online forums • Comments on Yelp • Video reviews • Pinterest images • Surveillance video

What kinds of column are there ?



What kinds of column are there ?

Example of how quantitative and qualitative data can be gathered from the same data unit

Data unit	Numeric variable = Quantitative data		Categorical variable = Qualitative data	
A person	"How many children do you have?"	4 children	"In which country were your children born?"	Australia
	"How much do you earn?"	$60,000 p.a.	"What is your occupation?"	Photographer
	"How many hours do you work?"	38 hours per week	"Do you work full-time or part-time?"	Full-time
A house	"How many square metres is the house?"	200 square metres	"In which city or town is the house located?"	Brisbane
A business	"How many workers are currently employed?"	264 employees	"What is the industry of the business?"	Retail
A farm	"How many milk cows are located on the farm?"	36 cows	"What is the main activity of the farm?"	Dairy

"Numbers are Your Friends": In data science, quantitative variables are the key to unlocking the full potential of machine learning algorithms.

Direct Algorithm Compatibility: Most machine learning models inherently require numerical input; quantitative variables fit directly into these algorithms.

Precise Measurement: Numerical data provides a scale of measurement, allowing for more precise and nuanced analysis compared to categorical data.

Enhanced Mathematical Operations: Quantitative variables allow for a wider range of mathematical and statistical operations, leading to deeper insights.

Improved Predictive Power: Numbers can capture more granular differences and patterns in data, enhancing the predictive power of models.

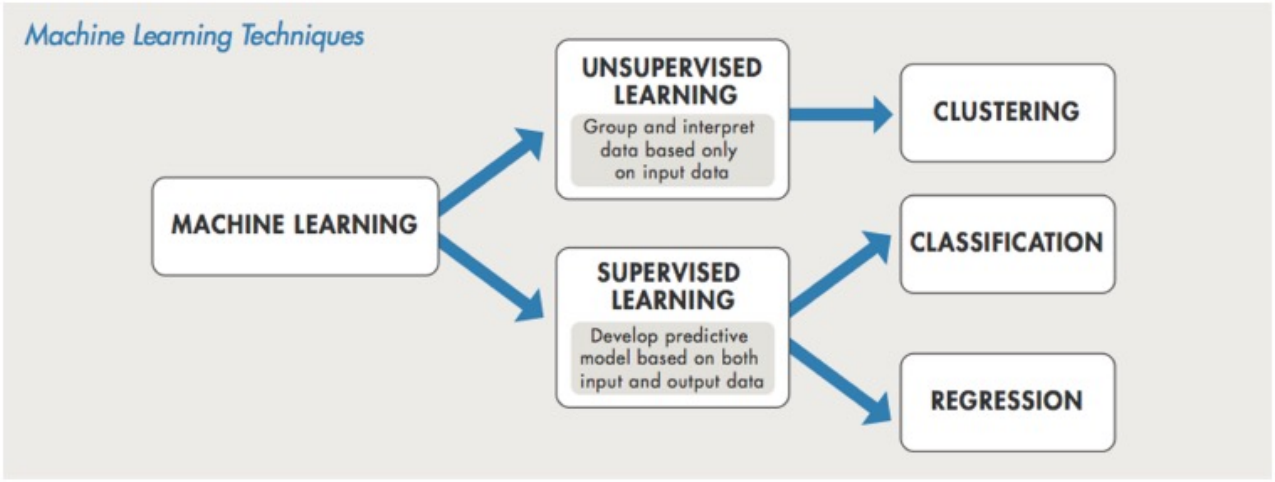
Facilitates Complex Analysis: Quantitative variables enable complex analytical techniques like regression analysis, factor analysis, and many others.

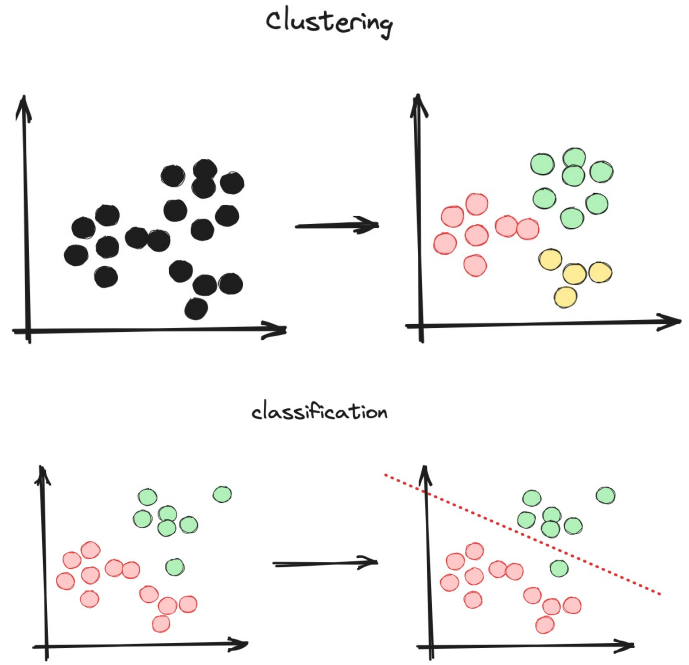
Reduces Ambiguity: Numbers are precise and leave less room for ambiguity compared to categorical data, which can be subjective or inconsistent.

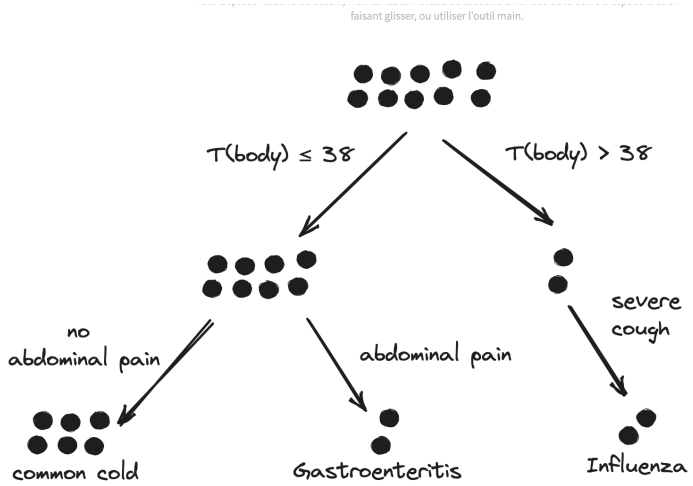
Data preprocessing

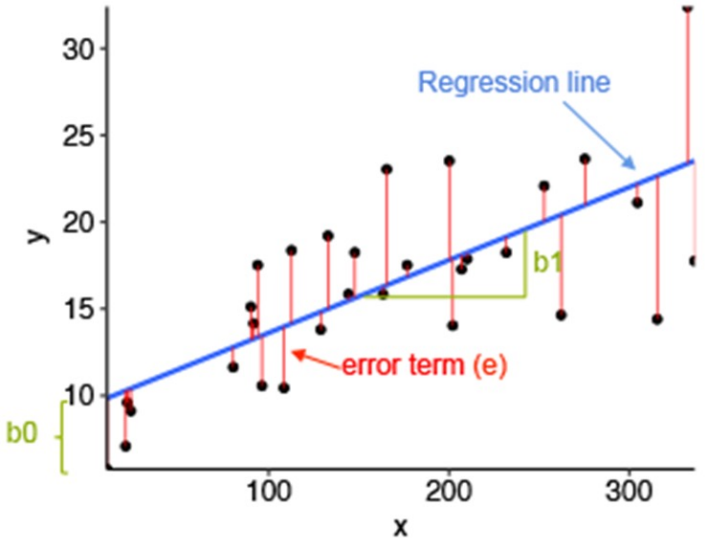
- **Data cleaning:** Eliminate errors, missing data and outliers.
- **Normalization:** Standardize variable scales for analysis.
- **Data Transformation:** Convert data into a usable format.
- **Missing Value Imputation:** Fill in missing data intelligently.
- **Category Coding:** Transform categorical variables into numerical values.
- **Dimensionality Reduction:** Reduce the number of variables, keep the essentials.
- **Feature Engineering:** Create new variables to improve models.
- **Data Balancing:** Adjust proportions to avoid bias.
- **Deduplication:** Remove duplicates to ensure uniqueness.
- **Segmentation:** Divide data into relevant subsets. Data
- **Aggregation:** Combine data from multiple sources.
- **Binning / Discretization:** Convert continuous variables into categories.
- **Aberrant Value Detection and Handling:** Identify and manage anomalies.
- **Text Cleaning (for textual data):** Clean and structure textual data.
- **Feature Scaling:** Scale variables for modeling.

Data analysis







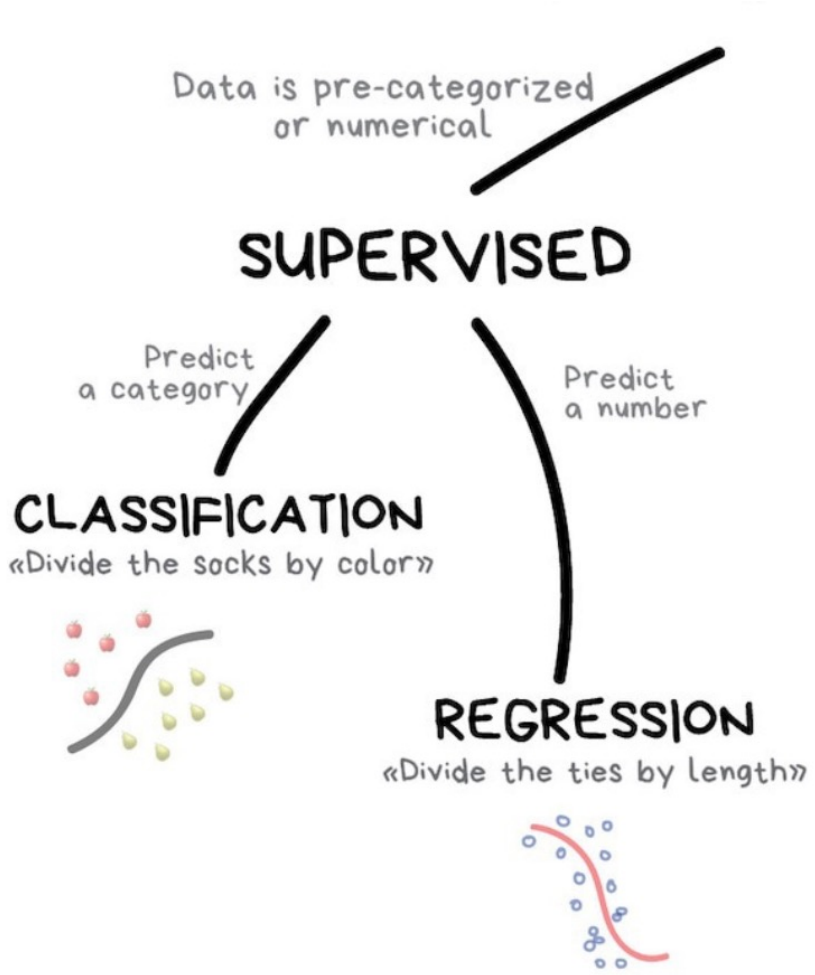


Data analysis

Types of data science

Supervised machine learning is when you have an input x and an output Y and you use an algorithm to learn an approximation of the function that maps the input to the output: Y= f(x)

A **classification problem** occurs when the output variable is a category such as in our use case “type of XXX”, “spam or not spams”. The algorithm learns a function that tries to separate the classes as best as possible.



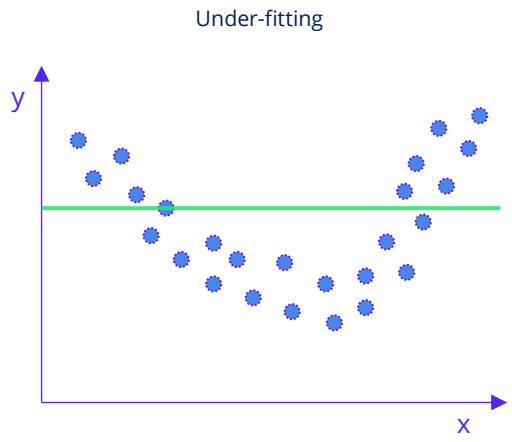
https://vas3k.com/blog/machine_learning/Machine

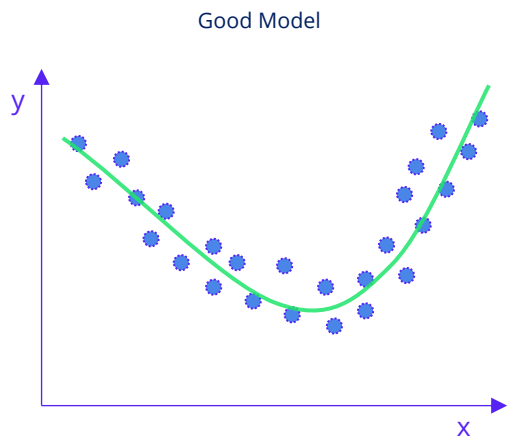
Data analysis

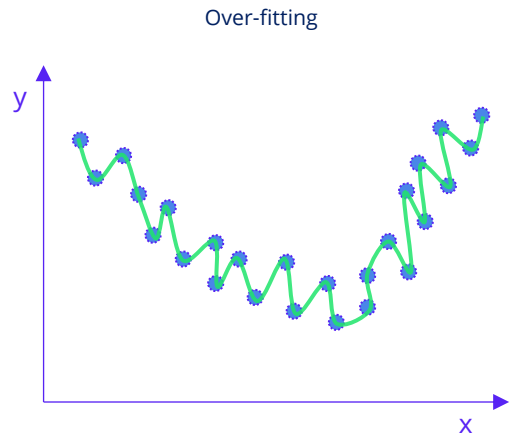
Overfitting vs underfitting

- The concept of **generalization** in data science is very important.
- **Overfitting** and **underfitting** are two terms used to talk about how well a model can be generalized.

- **Overfitting** refers to a model that learns too well on training data.
- **Underfitting** refers to a model that can neither perform well on the training data nor on unseen dat







Missing numerical values

https://insightsoftware.com/blog/how-to-handle-missing-data-values-while-data-cleaning/

Row no	State	Salary	Yrs of Experience
1	NY	57400	Mid
2	TX		Entry
3	NJ	90000	High
4	VT	36900	Entry
5	TX		Mid
6	CA	76600	High
7	NY	85000	High
8	CA		Entry
9	CT	45000	Entry

Missing values

Row no	State	Salary	Yrs of Experience
1	NY	57400	Mid
2	TX	65150	Entry
3	NJ	90000	High
4	VT	36900	Entry
5	TX	65150	Mid
6	CA	76600	High
7	NY	85000	High
8	CA	65150	Entry
9	CT	45000	Entry

Replaced with the mean salary

Replaced with mean entry level salary in TX

Row no	State	Salary	Yrs of Experience
1	NY	57400	Mid
2	TX	35000	Entry
3	NJ	90000	High
4	VT	36900	Entry
5	TX	48000	Mid
6	CA	76600	High
7	NY	85000	High
8	CA	43000	Entry
9	CT	45000	Entry

Replaced with mean Mid level salary in TX

Replaced with mean Entry level salary in CA

Numerical example :

- Replace with a constant value
- Replace with the mean or median
- Replace with value using the other column
- Predict the missing value with algorithm

Missing categorical values

https://insightsoftware.com/blog/how-to-handle-missing-data-values-while-data-cleaning/

Row no	State	Education
1	NY	High School
2	TX	
3	NJ	High School
4	VT	High School
5	TX	
6	CA	College
7	NY	High School
8	CA	
9	CT	College

Missing values

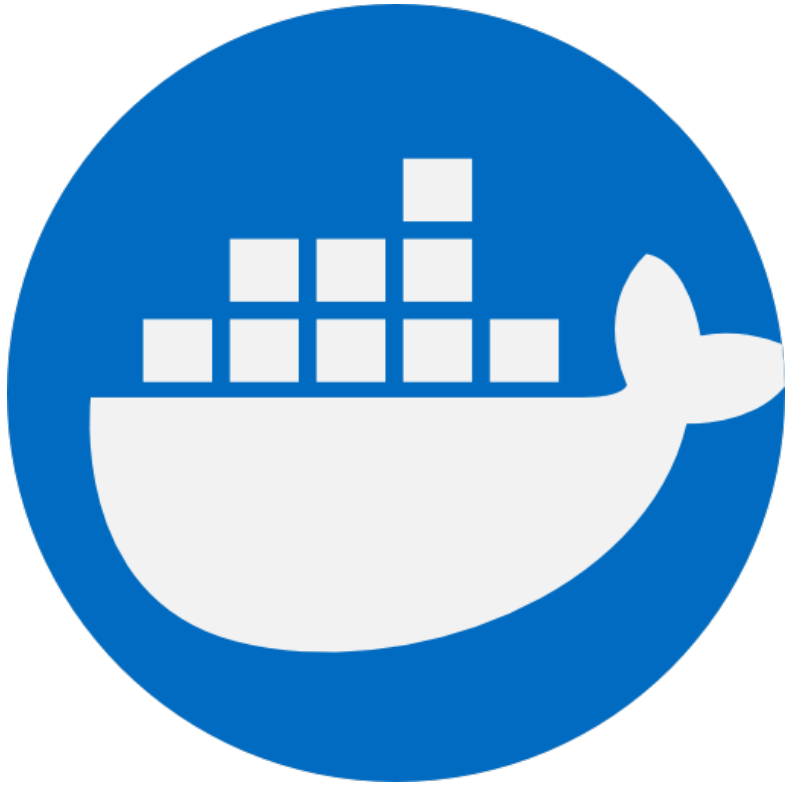
Row no	State	Education
1	NY	High School
2	TX	Unknown
3	NJ	High School
4	VT	High School
5	TX	Unknown
6	CA	College
7	NY	High School
8	CA	Unknown
9	CT	College

Missing values as a new category

Numerical example :

- Replace with a constant value
- Replace with the mean or median
- Replace with value using the other column
- Predict the missing value with algorithm

Before going further : Docker



What is it ?

- Container platform to isolate and deploy apps

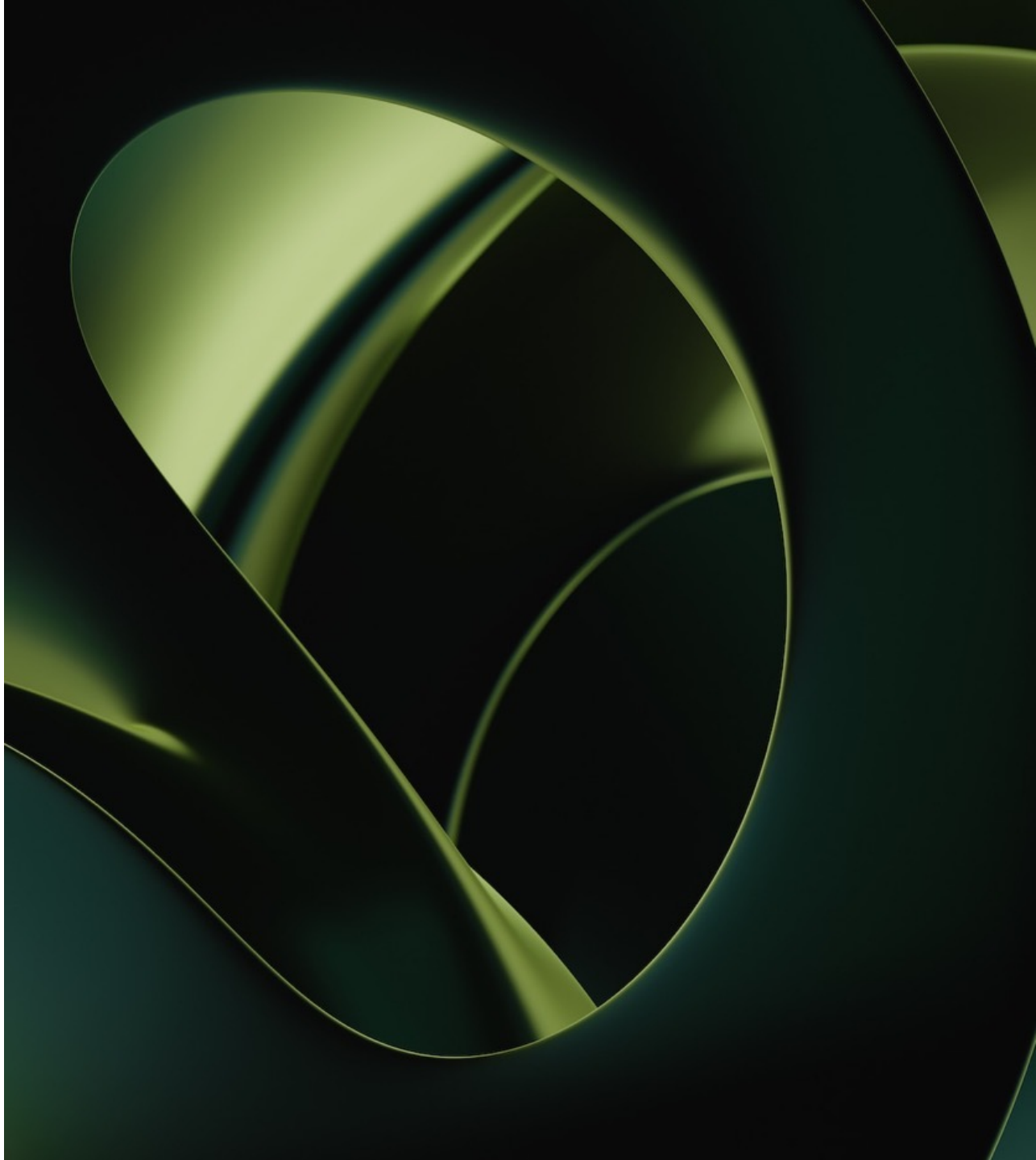
Advantages

- Consistency
- Portability
- Isolation
- Conflits reductions

Docker-compose

- Apps with multiple containers

Hands-on



saegus
SMART SHIFT · FOR REAL

House pricing prediction

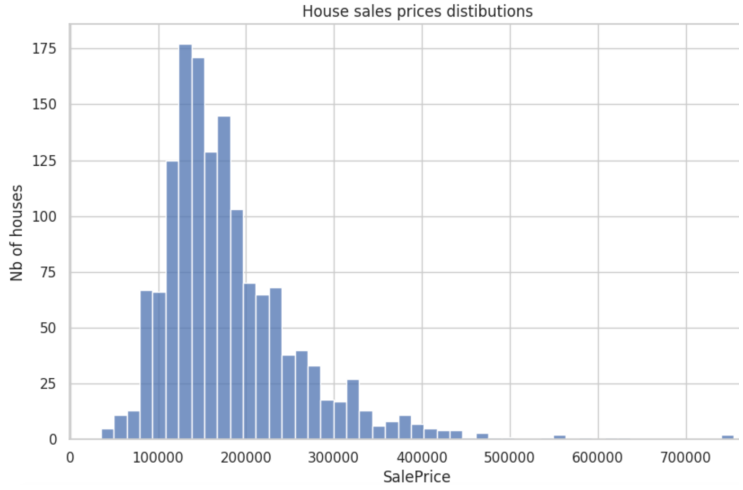
https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques
https://www.kaggle.com/couletsimon/episen-course/

What is it ?

- Dataset with everything to describe residential homes in Ames

Why ?

- 79 explenatory variables
- Small dataset (1460 rows)



Define the problem

Identify data

Features Overview:

- Sale Details:
  - *MoSold, YrSold, SaleType, SaleCondition*
- Property Attributes:
  - *MSSubClass, MSZoning, LotFrontage, LotArea*
  - *Street, Alley, LotShape, LandContour, Utilities*
- Location and Proximity:
  - *Neighborhood, Condition1, Condition2*
- Building and Quality:
  - *BldgType, HouseStyle, OverallQual, OverallCond, YearBuilt, YearRemodAdd*
- Basement and Interior:
  - *Foundation, BsmtQual, BsmtCond, BsmtExposure*
  - *BsmtFinType1, BsmtFinSF1, BsmtFinType2, BsmtFinSF2*
  - BsmtUnfSF, TotalBsmtS
- Exterior and Materials:
  - *RoofStyle, RoofMatl, Exterior1st, Exterior2nd*
  - *MasVnrType, MasVnrArea, ExterQual, ExterCond*
- Functional and Recreational:
  - *Functional, Fireplaces, FireplaceQu*
  - *GarageType, GarageYrBlt, GarageFinish, GarageCars*
  - *GarageArea, GarageQual, GarageCond*
- Bathrooms and Bedrooms:
  - *BsmtFullBath, BsmtHalfBath, FullBath, HalfBath, Bedroom, Kitchen, KitchenQual, TotRmsAbvGrd*
- Outdoor Features:
  - *PavedDrive, WoodDeckSF, OpenPorchSF, EnclosedPorch*
  - *3SsnPorch, ScreenPorch, PoolArea, PoolQC*
  - *Fence, MiscFeature, MiscVal*
- Utilities and Living Space:
  - *Heating, HeatingQC, CentralAir, Electrical*
  - *1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea*

Target Variable:

- SalePrice: Sale price of the property (in dollars)

Analysis Goal:

- We'd like to know what makes a house

Python Crash Course - Pandas

Before going into further anylisis, let's do a quick python crash course

- Pandas is the most used python library that provides data structures and data analysis tools
- It provides methods to read data files in different formats and to process the data in objects called DataFrames.
- A DataFrame is an object similar to a table that contains named columns.
- It is similar to the spark's Dataframe structure, but it isn't distributed
- Be careful, it won't support more than Gb of data in memory (we won't use more than <10Mb in the course)
- The crash course will give you a quick tour of functionalities offer by Pandas.


```
1 import pyspark
2 import pandas as pd
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 import numpy as np
```

A common convention is to import pandas as "pd" and seaborn as "sns"

Python Crash Course - Pandas

Before going into further anylisis, let's do a quick python crash course

- ***data.head(X) :*** To display the **first X lines** of the DataFrame

```
data.head(3)
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	Sale
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	20
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	18
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	22

3 rows × 81 columns

- ***data.shape:*** To get the (nb_rows, nb_col) of the DataFram

```
print("Data shape: {}".format(data.shape))
```

```
Data shape: (1460, 81)
```

- ***data.describe():*** To get the general statistics of the numerical columns of the DataFrame

```
data.describe()
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	...	WoodDeckSF	OpenPorchSF	EnclosedPorch	3
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	...	1460.000000	1460.000000	1460.000000	146
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	...	94.244521	46.660274	21.954110	
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.098091	...	125.338794	66.256028	61.119149	
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	...	0.000000	25.000000	0.000000	
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.250000	...	168.000000	68.000000	0.000000	
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	...	857.000000	547.000000	552.000000	50

Python Crash Course - Pandas

Before going into further anylisis, let's do a quick python crash course

- ***data.info():*** To show general information, column names, # of non-null, type of col

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
```

- ***data['colname']:*** *to select a specific column*

```
[30]: data['GrLivArea']
```

```
[30… 0       1710
     1       1262
     2       1786
     3       1717
     4       2198
             ...
     1455    1647
     1456    2073
     1457    2340
     1458    1078
     1459    1256
     Name: GrLivArea, Length: 1460, dtype: int64
```

- ***Data[['col1', 'col2', … 'colN']] :*** to create a sub-dataframe from an array of column

```
[31]: data[['GrLivArea', 'SalePrice']]
```

[31…

	GrLivArea	SalePrice
0	1710	208500
1	1262	181500
2	1786	223500
3	1717	140000
4	2198	250000
…	…	…

- ***data.filter(like='XxX').columns:*** to get returns an array containing columns that contains 'XxX' in their names

```
bsmt_columns = data.filter(like='Bsmt').columns
data[bsmt_columns].head()
```

	BsmtQual	BsmtCond	BsmtExposure	BsmtFinType1	BsmtFinSF1	Bs
0	Gd	TA	No	GLQ	706	
1	Gd	TA	Gd	ALQ	978	
2	Gd	TA	Mn	GLQ	486	
3	TA	Gd	No	ALQ	216	
4	Gd	TA	Av	GLQ	655	

Python Crash Course - Pandas

Before going into further anylisis, let's do a quick python crash course

- ***data['col'].dtype :*** *to return the type of the col*
- ***Data['col'].astype(type) :*** *to cast to specified type*

```
print(data['GrLivArea'].dtype)

data['GrLivArea'] = data['GrLivArea'].astype(str)
print(data['GrLivArea'].dtype)

data['GrLivArea'] = data['GrLivArea'].astype(int)
print(data['GrLivArea'].dtype)
```

```
int64
object
int64
```

- ***Filters the columns***

```
data[(data['GrLivArea']>4000) & (data['SalePrice']<300000)].head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition
523	524	60	RL	130.0	40094	Pave	NaN	IR1	Bnk	AllPub	...	0	NaN	NaN	NaN	0	10	2007	New	Partial
1298	1299	60	RL	313.0	63887	Pave	NaN	IR3	Bnk	AllPub	...	480	Gd	NaN	NaN	0	01	2008	New	Partial

2 rows × 81 columns

Python Crash Course - Pandas

First example of data-cleaning : Getting a data out of two columns (1/3)

- We have two columns for the date ; *MoSold* and *YrSold*

=> We'd like to get the full date of sale as a column *instead of 2 distrincts information*
=> We'd like to have sth like « 2010-12 » as a date => concatenation

- What does those columns look like ?

```
year_col_names = ["MoSold","YrSold"]
data[year_col_names].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   MoSold  1460 non-null   int64
 1   YrSold  1460 non-null   int64
```

```
data[year_col_names].head()
```

	MoSold	YrSold
0	2	2008
1	5	2007
2	9	2008
3	2	2006
4	12	2008

=> The columns are both integers
=> We'd like to concatenate both
=> We can't concatenate integers as is
=> MoSold < 10 ?

Python Crash Course - Pandas

First example of data-cleaning : Getting a data out of two columns

- We have two columns for the date ; *MoSold* and *YrSold*

=> We'd like to get the full date of sale as a column *instead of 2 distrincts information*
=> We'd like to have sth like « 2010-12 » as a date => concatenation

- We'd like to convert MoSold 6 -> 06

those are optional but good practice

```python
def add_0_to_month(month: int) -> str:
    if month < 10:
        return "0"+str(month)
    else:
        return str(month)

assert(add_0_to_month(6)=='06')
assert(add_0_to_month(12)=='12')
```

those are "unit tests"

- And apply it to the whole column

```python
data['MoSold'] = data['MoSold'].apply(lambda month: add_0_to_month(month))
data['MoSold']
```

```
0    02
1    05
2    09
3    02
4    12
     ..
```

Python Crash Course - Pandas

First example of data-cleaning : Getting a data out of two columns

- We have two columns for the date ; *MoSold* and *YrSold*

=> We'd like to get the full date of sale as a column *instead of 2 distrincts information*
=> We'd like to have sth like « 2010-12 » as a date => concatenation

- Construct a new column from the 2 others

```
data['SaleDateStr'] = data['YrSold'].astype(str) + '-' + data['MoSold']
data['SaleDateStr'].head()
```

```
0    2008-02
1    2007-05
2    2008-09
3    2006-02
4    2008-12
Name: SaleDateStr, dtype: object
```

- Convert the new column to date

```
data['SaleDate'] = pd.to_datetime(data['SaleDateStr'])
data['SaleDate'].head()
```

```
0   2008-02-01
1   2007-05-01
2   2008-09-01
3   2006-02-01
4   2008-12-01
Name: SaleDate, dtype: datetime64[ns]
```

- Drop the intermediate columns (*inplace* arg to drop in the original dataframe)

```
data.drop(columns=['MoSold', 'YrSold', 'SaleDateStr'], inplace=True)
```

Python Crash Course - Pandas

First example of data-cleaning : Getting a data out of two columns

- We have two columns for the date ; *MoSold* and *YrSold*

=> We'd like to get the full date of sale as a column *instead of 2 distrincts information*
=> We'd like to have sth like « 2010-12 » as a date => concatenation

- Construct a new column from the 2 others

```
data['SaleDateStr'] = data['YrSold'].astype(str) + '-' + data['MoSold']
data['SaleDateStr'].head()
```

```
0    2008-02
1    2007-05
2    2008-09
3    2006-02
4    2008-12
Name: SaleDateStr, dtype: object
```

- Convert the new column to date

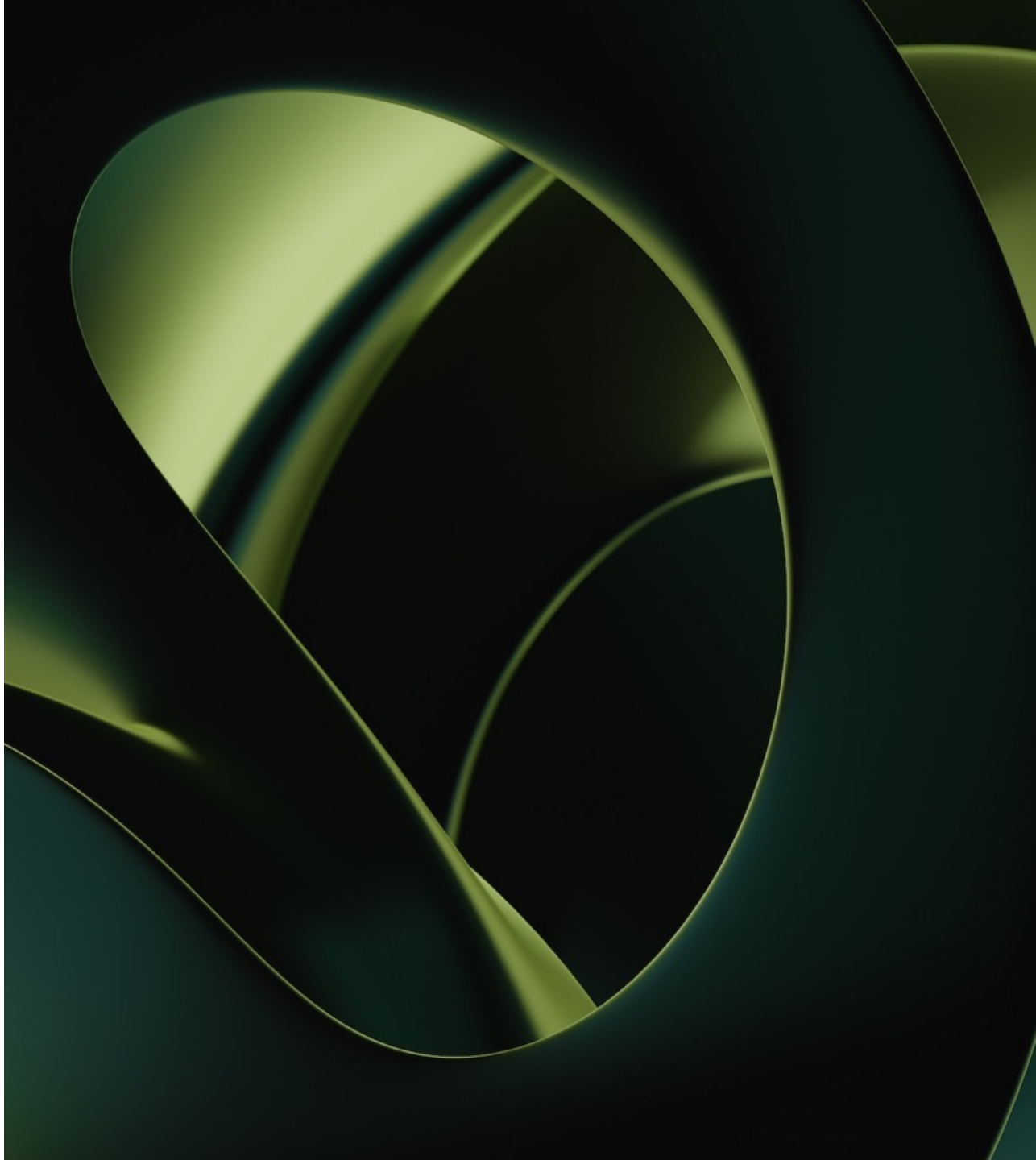
```
data['SaleDate'] = pd.to_datetime(data['SaleDateStr'])
data['SaleDate'].head()
```

```
0   2008-02-01
1   2007-05-01
2   2008-09-01
3   2006-02-01
4   2008-12-01
Name: SaleDate, dtype: datetime64[ns]
```

- Drop the intermediate columns (*inplace* arg to drop in the original dataframe)

```
data.drop(columns=['MoSold', 'YrSold', 'SaleDateStr'], inplace=True)
```

Descriptive analysis



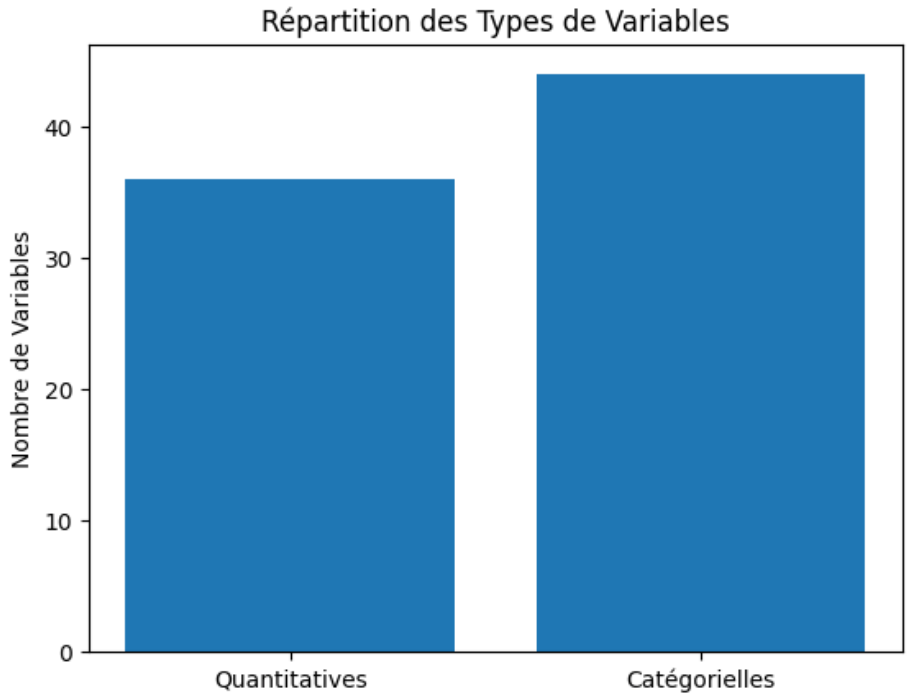
What are the factors that influence the price ?

- **Investment:**
  - "How can we maximize ROI by predicting future property valuations in the real estate market?"
- **Mortgage Lending:**
  - "What predictive model can we use to assess property values for setting loan-to-value ratios and mortgage conditions?"
- **Insurance:**
  - "How can insurance premiums be adjusted based on the predictive value of properties in different areas?"
- **Government and Urban Planning:**
  - "Which predictive insights can assist in urban development and tax revenue planning based on housing prices?"
- **Sellers and Real Estate Agents:**
  - "What pricing strategy should be used for listing properties to remain competitive in the current market?"
- **Buyers:**
  - "How do we identify undervalued properties to make strategic purchase decisions?"
- **Renovation and Development:**
  - "Which home improvements are predicted to add the most value to properties in our portfolio?"

Qualitative vs Quantitatives

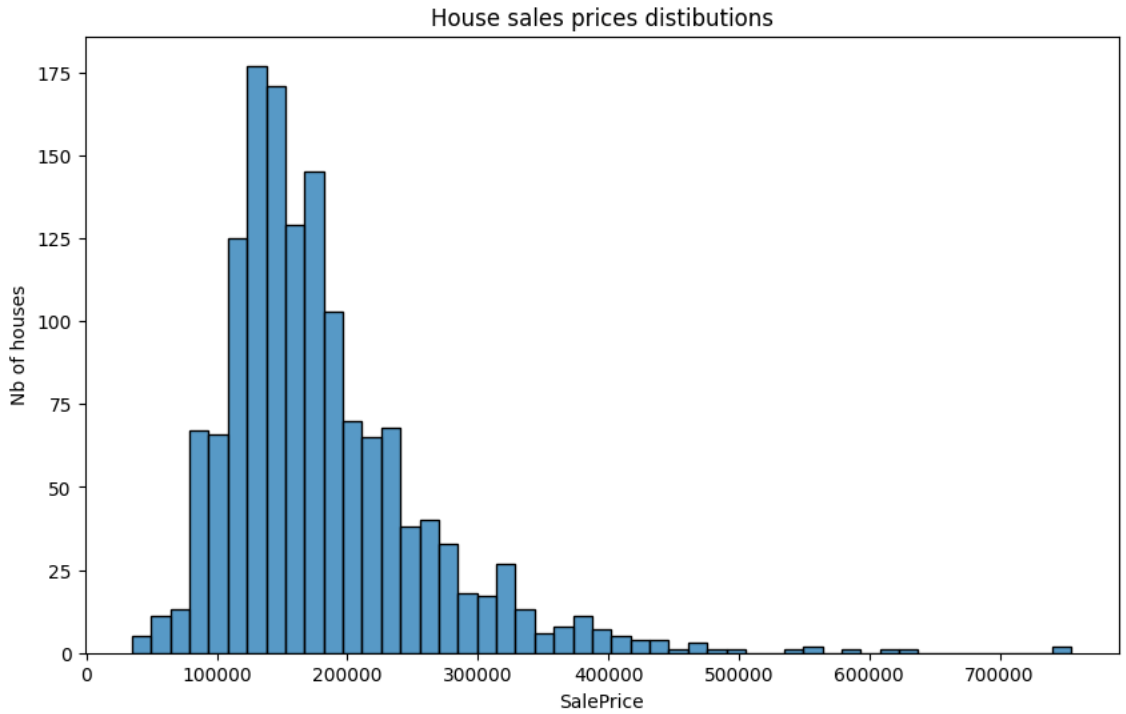
```
types_variables = {'Quantitatives': len(data.select_dtypes(include=['float', 'int']).columns),
                   'Catégorielles': len(data.select_dtypes(include=['object']).columns)}

plt.bar(types_variables.keys(), types_variables.values())
plt.title('Répartition des Types de Variables')
plt.ylabel('Nombre de Variables')
plt.show()
```



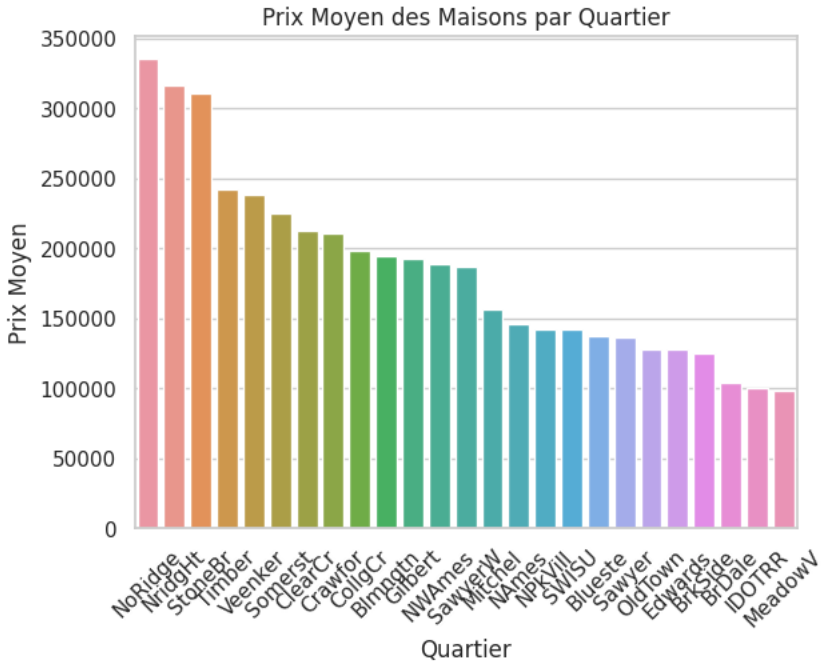
Target variable distribution

```
plt.figure(figsize=(10,6))
#plt.hist(data['SalePrice'], bins=30, edgecolor='black')
sns.histplot(data['SalePrice'])
plt.title('House sales prices distibutions')
plt.xlabel('SalePrice')
plt.ylabel('Nb of houses')
plt.show()
```

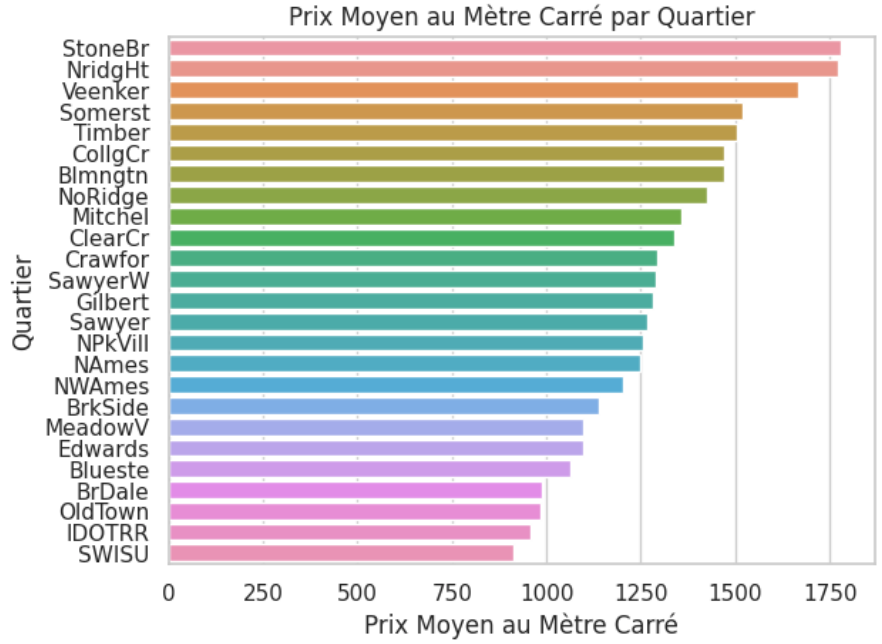


Uptown vs working-class neighborhood

```
mean_prices = data.groupby('Neighborhood')['SalePrice'].mean().sort_values(ascending=False)
sns.barplot(x=mean_prices.index, y=mean_prices.values)
plt.xticks(rotation=45)
plt.title('Prix Moyen des Maisons par Quartier')
plt.ylabel('Prix Moyen')
plt.xlabel('Quartier')
plt.show()
```

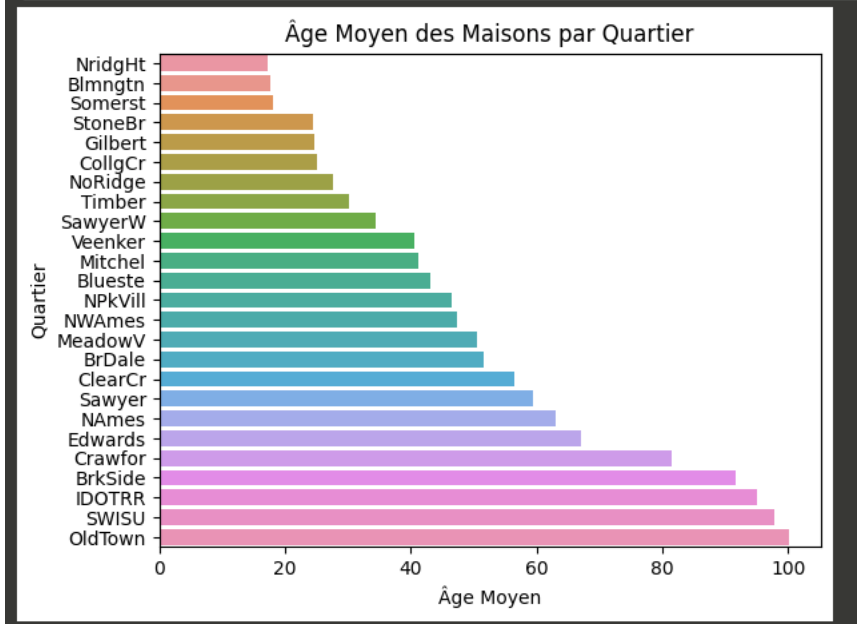



```
FTSQ_MSQ_RATIO = 0.092903 # 1 ftsq = 0.092903 m2
data['PricePerSqMt'] = data['SalePrice'] / (data['GrLivArea'] * FTSQ_MSQ_RATIO)
mean_price_per_sqmt = data.groupby('Neighborhood')['PricePerSqMt'].mean().sort_values(ascending=False)
# Visualisation
sns.barplot(x=mean_price_per_sqmt.values, y=mean_price_per_sqmt.index)
plt.title('Prix Moyen au Mètre Carré par Quartier')
plt.xlabel('Prix Moyen au Mètre Carré')
plt.ylabel('Quartier')
plt.show()
```

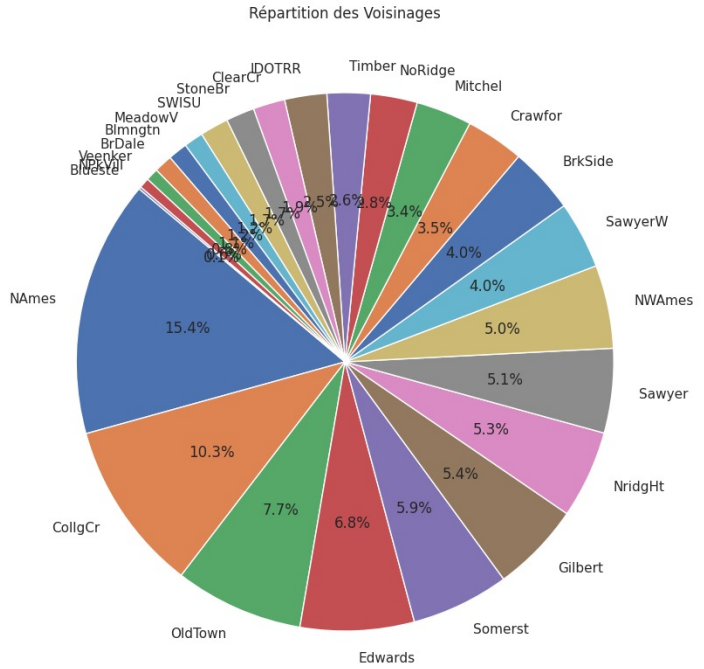


Uptown vs working-class neighborhood

```
data['HouseAge'] = 2023 - data['YearBuilt']
mean_age = data.groupby('Neighborhood')['HouseAge'].mean().sort_values()
sns.barplot(x=mean_age.values, y=mean_age.index)
plt.title('Âge Moyen des Maisons par Quartier')
plt.xlabel('Âge Moyen')
plt.ylabel('Quartier')
plt.show()
```

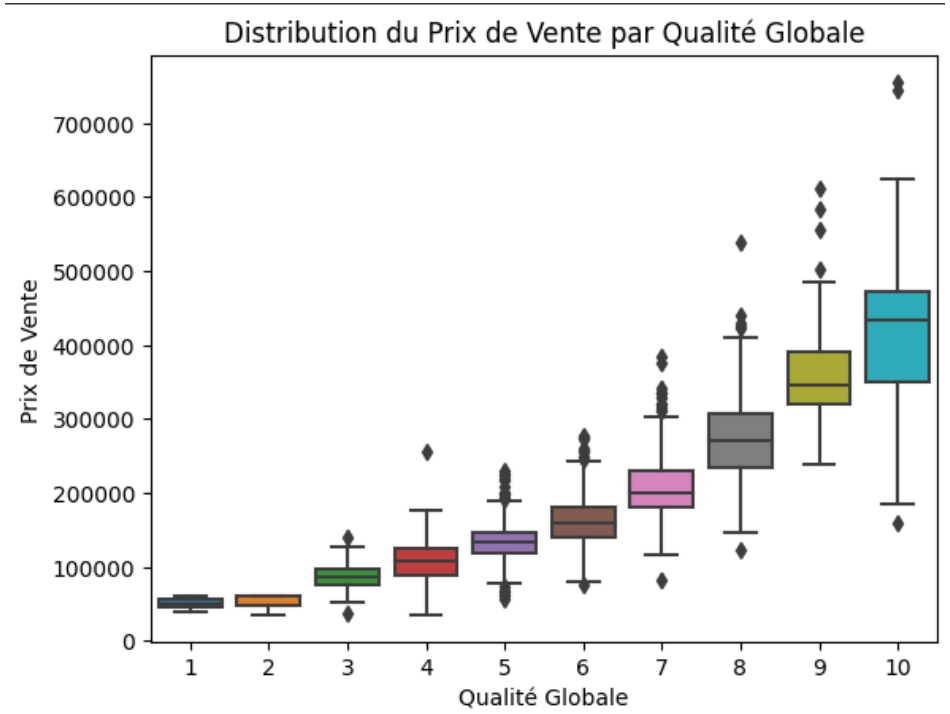



```
plt.figure(figsize=(10, 10))
sns.set(style="whitegrid")
plt.pie(neighborhood_counts, labels=neighborhood_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Répartition des Voisinages')
plt.show()
```

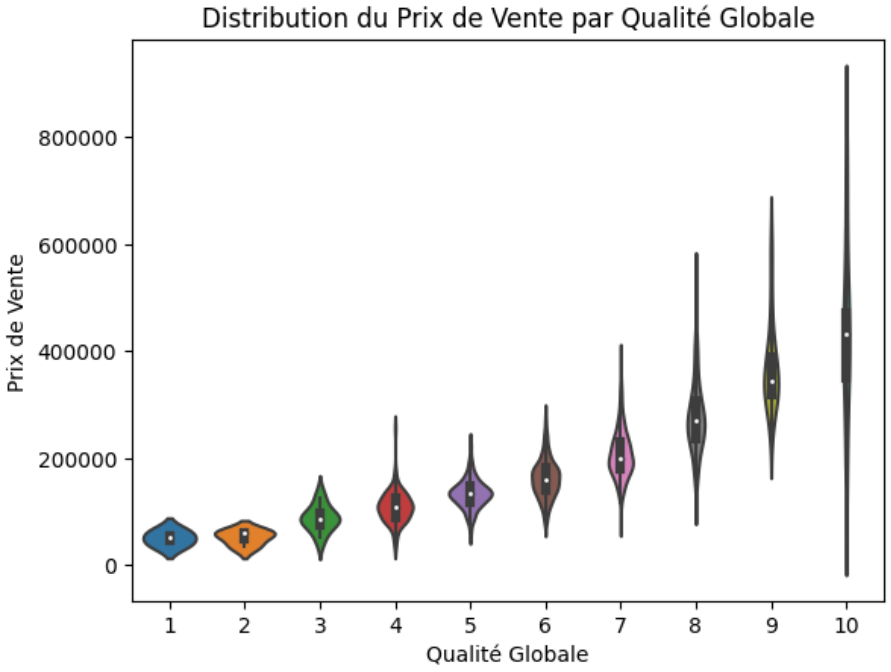


OverallQual vs SalePrice

```
sns.boxplot(x='OverallQual', y='SalePrice', data=data)
plt.title('Distribution du Prix de Vente par Qualité Globale')
plt.xlabel('Qualité Globale')
plt.ylabel('Prix de Vente')
plt.show()
```

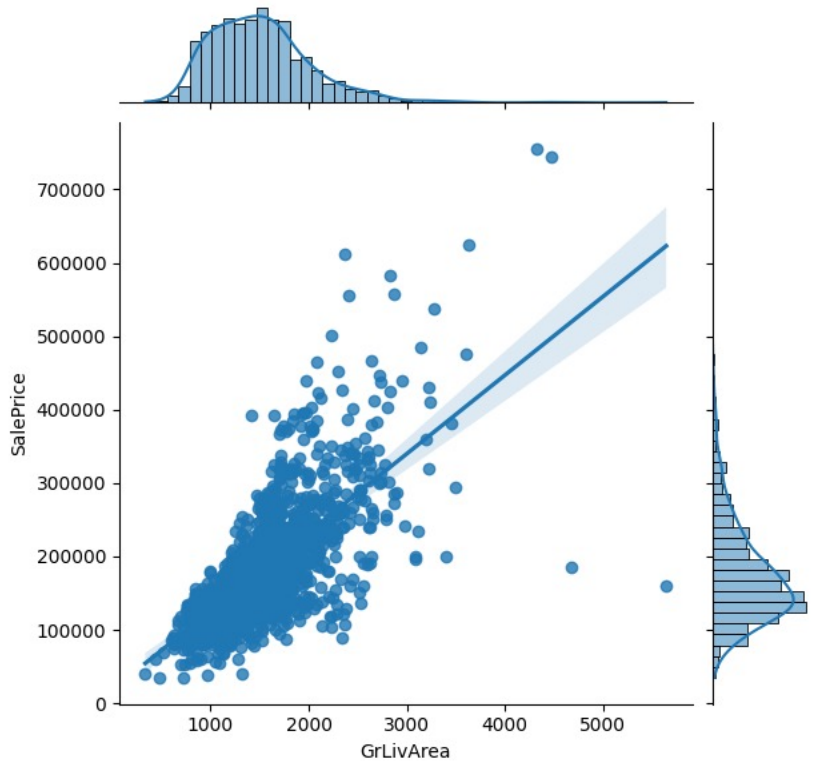



```
sns.violinplot(x='OverallQual', y='SalePrice', data=data)
plt.title('Distribution du Prix de Vente par Qualité Globale')
plt.xlabel('Qualité Globale')
plt.ylabel('Prix de Vente')
plt.show()
```



GrLiveArea vs SalePrice

```
sns.jointplot(x='GrLivArea', y='SalePrice', data=data, kind='reg')
plt.show()
```



- There is a clear **positive linear relationship** between GrLivArea (above grade (ground) living area square feet) and SalePrice. As GrLivArea increases, SalePrice tends to increase as well.
- **Data Spread:** The concentration of data points around the line of best fit suggests that GrLivArea is a good predictor of SalePrice.
- **Outliers:** Some houses with large living areas that are priced lower than the trend would suggest, and a few with high SalePrice regardless of GrLivArea.
- **Distribution Analysis:** The histograms show that both GrLivArea and SalePrice are right-skewed, indicating that the majority of houses have smaller living areas and lower sale prices, with a few exceptions on the higher end.
- **Densest Regions:** The darkest area of the scatter plot, where the points are most concentrated, indicates the most common range for GrLivArea and SalePrice.

Handle Categorical Variables
Ordinal case : mapping

```
quality_mapping = {'Po': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5}
data['ExterQual'] = data['ExterQual'].map(quality_mapping)
data['ExterQual'].head()
```

```
0    4
1    3
2    4
3    3
4    4
Name: ExterQual, dtype: int64
```

Handle Categorical Variables
Ordinal case : mapping

```
data['CentralAir'].unique()
```

```
array(['Y', 'N'], dtype=object)
```

=> We can't order the variable « CentralAir »
=> So how can we convert them into a numerical ?
Just put No -> 0, Yes -> 1

```
print(data['CentralAir'].head())
data['CentralAir'] = data['CentralAir'].map({'N': 0, 'Y': 1})
print(data['CentralAir'].head())
```

```
0    Y
1    Y
2    Y
3    Y
4    Y
Name: CentralAir, dtype: object
0    1
1    1
2    1
3    1
4    1
Name: CentralAir, dtype: int64
```

```
data['CentralAir'].unique()
```

```
array([1, 0])
```

Handle Categorical Variables : Dummy coding : simple case

```
data['CentralAir'].unique()
```

```
array(['Y', 'N'], dtype=object)
```

=> We can't order the variable « CentralAir »
=> So how can we convert them into a numerical ?
Just put No -> 0, Yes -> 1

```
print(data['CentralAir'].head())
data['CentralAir'] = data['CentralAir'].map({'N': 0, 'Y': 1})
print(data['CentralAir'].head())
```

```
0    Y
1    Y
2    Y
3    Y
4    Y
Name: CentralAir, dtype: object
0    1
1    1
2    1
3    1
4    1
Name: CentralAir, dtype: int64
```

```
data['CentralAir'].unique()
```

```
array([1, 0])
```

Handle Categorical Variables : Dummy coding : general case

```
data[['Id', 'Foundation', 'SalePrice']].head()
```

	Id	Foundation	SalePrice
0	1	PConc	208500
1	2	CBlock	181500
2	3	PConc	223500
3	4	BrkTil	140000
4	5	PConc	250000

=> We can't order the variable « Foundation »
=> So how can we convert them into a numerical ?

```
data['Foundation'].unique()
```

```
array(['PConc', 'CBlock', 'BrkTil', 'Wood', 'Slab', 'Stone'], dtype=object)
```

```
data['Foundation'].describe()
```

```
count      1460
unique        6
top       PConc
freq        647
Name: Foundation, dtype: object
```

Handle Categorical Variables : Dummy coding

```
data[['Id', 'Foundation', 'SalePrice']].head()
```

	Id	Foundation	SalePrice
0	1	PConc	208500
1	2	CBlock	181500
2	3	PConc	223500
3	4	BrkTil	140000
4	5	PConc	250000

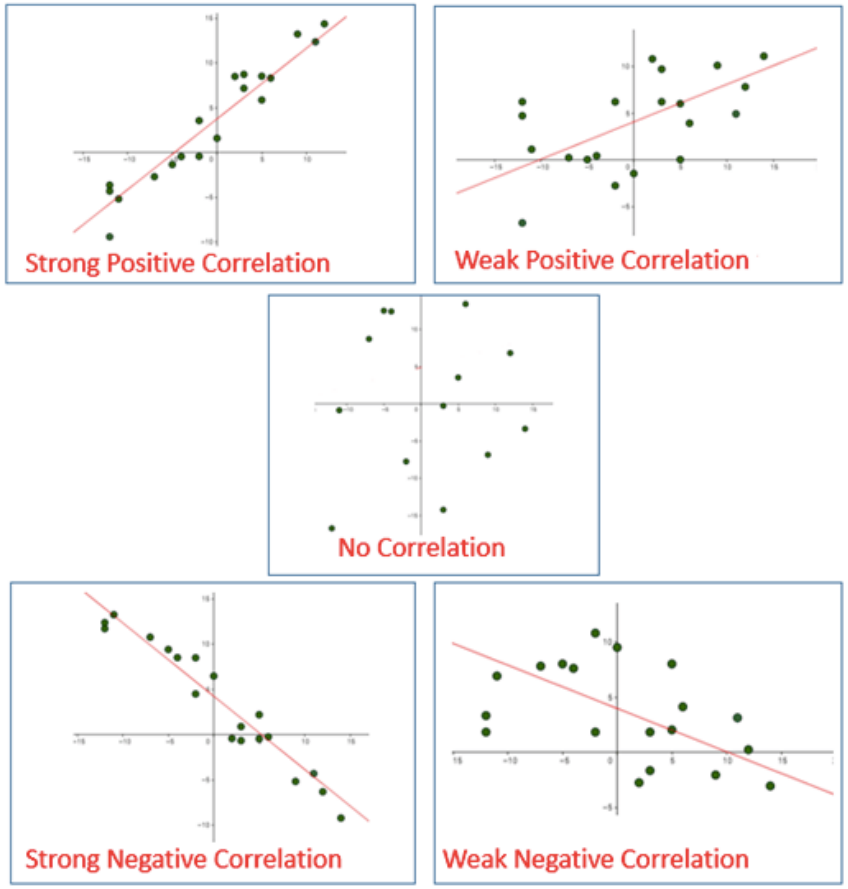
```
foundation_dummies = pd.get_dummies(data['Foundation'], prefix='Foundation', dtype=int)
data = pd.concat([data, foundation_dummies], axis=1)
data = data.drop(columns=['Foundation'])
```

```
frames = [data[['Id']], foundation_dummies, data[['SalePrice']]]
result = pd.concat(frames, axis=1)
result.head()
```

	Id	Foundation_BrkTil	Foundation_CBlock	Foundation_PConc	Foundation_Slab	Foundation_Stone	Foundation_Wood	SalePrice
0	1	0	0	1	0	0	0	208500
1	2	0	1	0	0	0	0	181500
2	3	0	0	1	0	0	0	223500
3	4	1	0	0	0	0	0	140000
4	5	0	0	1	0	0	0	250000

- **Model Compatibility:** Transforms categorical variables into a format that can be easily used by machine learning algorithms.
- **Prevents Misinterpretation:** Avoids erroneous ordinal assumptions by algorithms that may misinterpret categorical data as being ordinal.
- **Feature Engineering:** Enhances model performance by creating explicit features for each category.
- **Preserves Information:** Retains all the information present in categorical variables while making it usable for algorithms.
- **Improves Interpretability:** Makes model output easier to interpret and understand, especially in regression models.
- **Facilitates Complex Relationships:** Allows models to capture more complex relationships that might be present in categorical data.

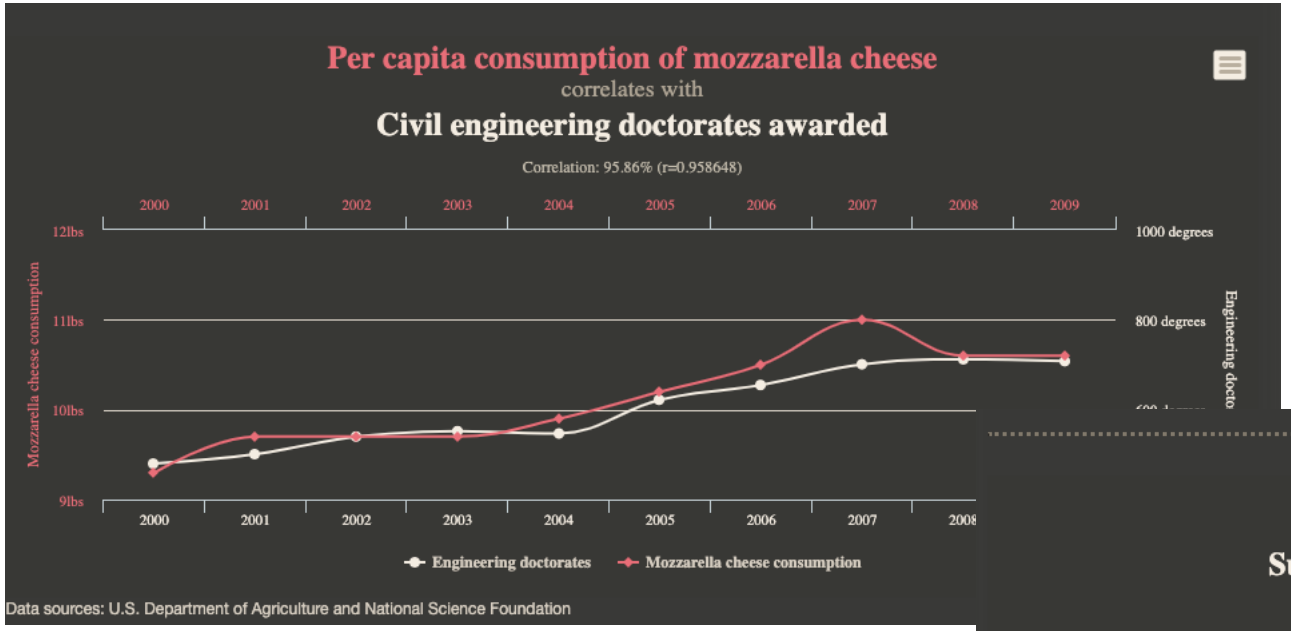
Correlation

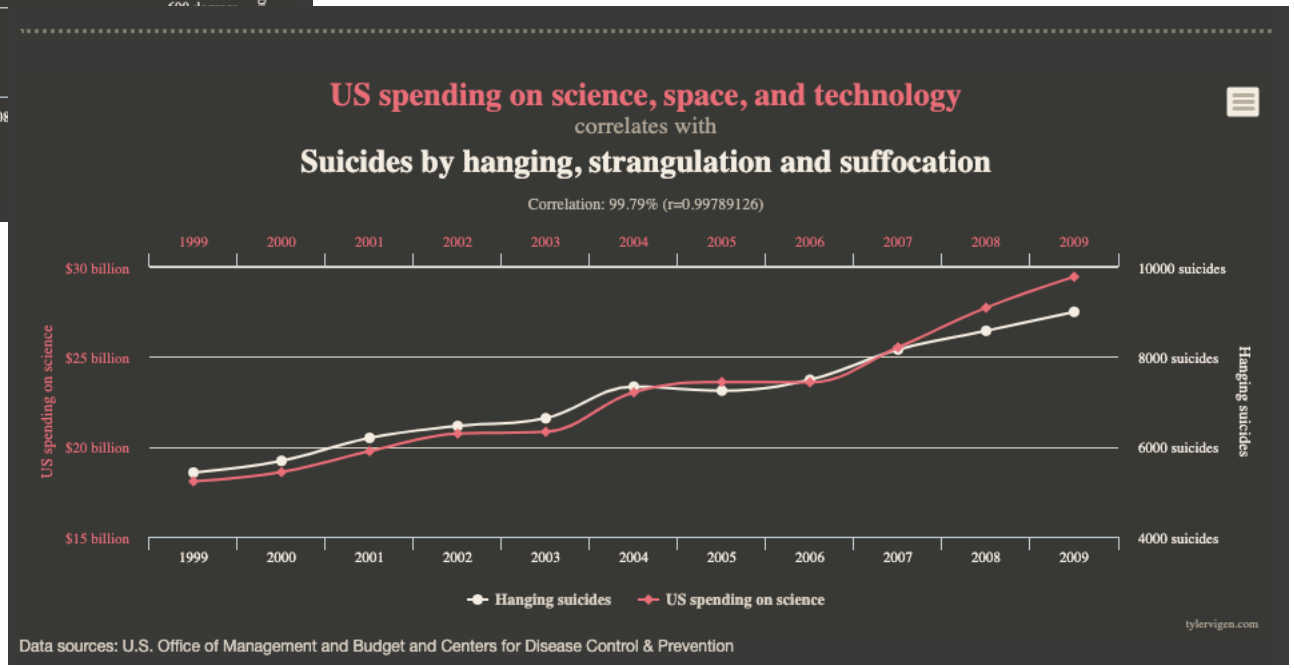


- **Correlation Overview:**
  - Correlation quantifies the degree to which two variables are related.
  - It measures both the strength and direction of the linear relationship.
- **Coefficient Values:**
  - The correlation coefficient ranges from -1 to +1.
  - A value close to +1 indicates a strong positive correlation.
  - A value close to -1 indicates a strong negative correlation.
  - A value close to 0 suggests no linear correlation.
- **Positive vs. Negative:**
  - **Positive correlation:** as one variable increases, the other tends to increase.
  - **Negative correlation:** as one variable increases, the other tends to decrease.
- **Visual Representation:**
  - Scatter plots visually depict the relationship between variables.
  - The pattern of the plotted points can suggest the correlation type.
- **Statistical Significance:**
  - Not all correlations are meaningful; statistical significance must be tested.
  - Correlation does not confirm causation; it merely implies a relationship.
- **Correlation vs. Causation:**
  - **Critical Reminder:** Correlation does not imply causation.
  - Correlated variables may or may not have a cause-and-effect relationship.

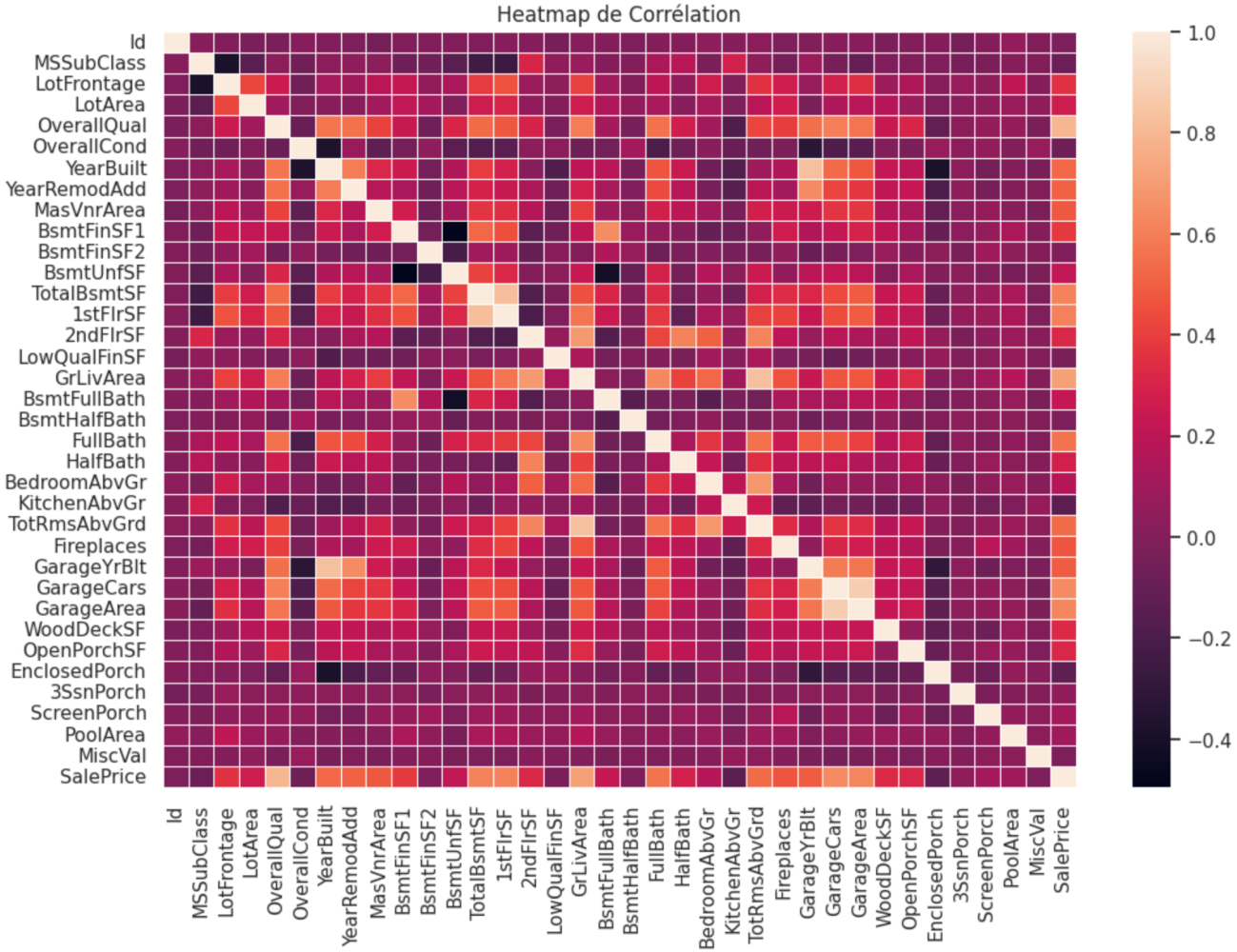
Correlation does not imply causation !

https://www.tylervigen.com/spurious-correlations





Correlation



- **High correlation** between GarageCars and GarageArea indicating **multicollinearity**; consider **dropping one**.
- TotalBsmtSF and 1stFlrSF also **highly correlated**, suggesting **multicollinearity**; advisable to remove one.
- **Strong correlation** of TotalBsmtSF with **target variable** SalePrice implies it's a **significant predictor** for the model.
- YearBuilt and GarageYrBlt are **highly correlated**; given missing data in GarageYrBlt, retain YearBuilt.
- Key predictors for SalePrice include OverallQual, GrLivArea, FullBath, TotRmsAbvGrd.
- TotRmsAbvGrd shows high correlation with GrLivArea; further analysis needed to decide which to keep.

Tranformations

```
# 1. Surface Totale
data['TotalSquareFootage'] = data['TotalBsmtSF'] + data['1stFlrSF'] + data['2ndFlrSF']
```

```
# 2. Indice de Qualité Globale (en considérant que les deux ont le même poids)
data['OverallGrade'] = (data['OverallQual'] + data['OverallCond']) / 2
```

```
# 4. Garage Score
data['GarageScore'] = data['GarageCars'] * data['GarageArea']
```

```
# 5. Score Salle de Bains
data['BathroomScore'] = data['FullBath'] + (0.5 * data['HalfBath']) + \
                        data['BsmtFullBath'] + (0.5 * data['BsmtHalfBath'])
```

```
# 6. Années Depuis la Rénovation
data['YearsSinceRemodel'] = data['YrSold'] - data['YearRemodAdd']
```

```
columns_to_drop = ['TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'OverallQual', 'OverallCond',
                   'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', 'GarageCars', 'GarageArea',
                   'FullBath', 'HalfBath', 'BsmtFullBath', 'BsmtHalfBath', 'YearRemodAdd']
data = data.drop(columns=columns_to_drop)
```

```
# mean price of sqmt
data['PricePerSqMt'] = data['SalePrice'] / data['GrLivArea']
average_price_per_neighborhood = data.groupby('Neighborhood')['PricePerSqMt'].mean().sort_values()

# thresholds
thresholds = average_price_per_neighborhood.quantile([0.33, 0.66]).tolist()

# Fonction pour assigner les catégories
def assign_category(price_per_sqmt):
    if price_per_sqmt <= thresholds[0]:
        return 0
    elif price_per_sqmt <= thresholds[1]:
        return 1
    else:
        return 2

data['NeighborhoodCat'] = data['Neighborhood'].map(average_price_per_neighborhood).apply(assign_cate
```