First things first!

Take the quiz for today!

Quiz Lecture 2 in Module 2.

(It times out soon.)

When you are done, please download:

- **Titanic.csv**
- **ML&O Lecture 2 – Exercise Book.ipynb**

Make sure they are in the same folder!

The Business School for the World®

Machine Learning and Optimization
Lecture 2

Data and Feature Engineering

Professor Georgina Hall

Agenda for today

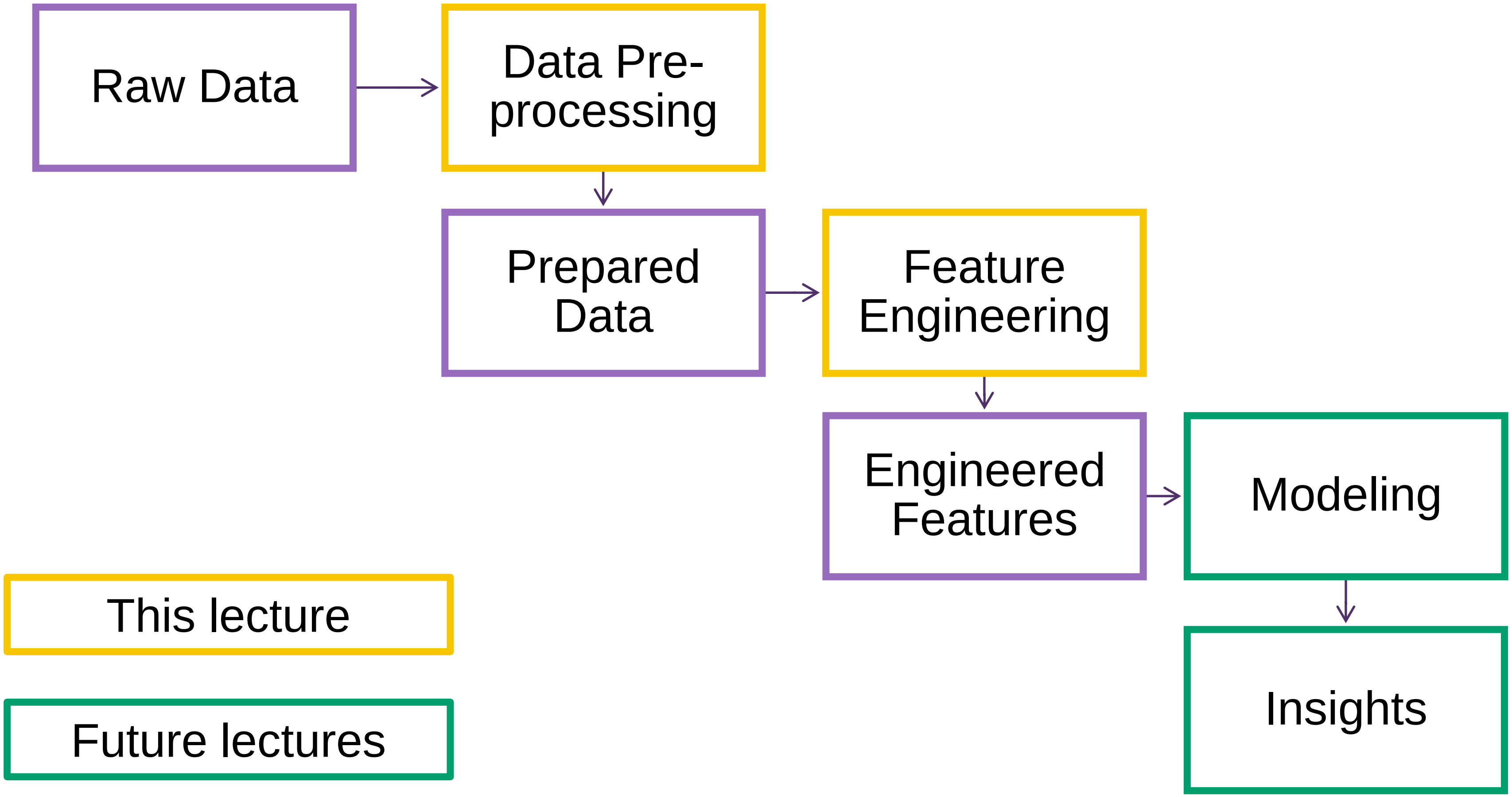
Learning objectives:

- Understand the key issues that can be faced when considering raw data
- Learn how to identify and deal with them

How will we get there?

- Appeal to your intuition: if you had to identify these issues how would you go about it? How would you deal with them?
- Use of the **Titanic Dataset**

The Machine Learning pipeline



About this lecture

A very important part of the process.

- Many, many techniques for data preprocessing and feature engineering
- Often requires business knowledge/detective work combined with good technical knowledge to identify and correct
- Focus on the main issues, give some work-arounds and how to use Python to deal with these issues

The Titanic Dataset (1/2)

- Gives **a list of passengers on the Titanic** with some of their features and whether they survived (see Lectures 5 and 6)

- Discover the dataset using **df.head(), df.info()**
  - ⇒ what are the features? do you understand them all?
  - ⇒ how many observations?

```
Titanic.head()
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	third	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	first	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	third	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	first	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	third	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Observation

Feature

The Titanic Dataset (2/2)

```
Titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 893 entries, 0 to 892
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  893 non-null    int64
 1   Pclass       893 non-null    object
 2   Name         893 non-null    object
 3   Sex          893 non-null    object
 4   Age          716 non-null    float64
 5   SibSp        893 non-null    int64
 6   Parch        893 non-null    int64
 7   Ticket       893 non-null    object
 8   Fare         893 non-null    float64
 9   Cabin        205 non-null    object
 10  Embarked     891 non-null    object
dtypes: float64(2), int64(3), object(6)
memory usage: 76.9+ KB
```

- 11 features, 893 observations
- 2 are floats, 3 are integers and 6 are objects (objects are either strings or mixed strings and numbers)

Difference between categorical and numerical variables

A **numerical variable** denotes a **measurement or a count.**
Examples: fare, age, sibsp

A **categorical variable** denotes membership to a **category** (can be a number but the number is a stand in for a category).
Examples: Pclass, sex

Raw data

INSEAD

What could be some issues encountered in raw data?

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Data
Pre-Processing

Data cleansing

A dataset may have any of the following **issues**:

- Values that are not in the right **format** (e.g., a number that's been written as a string)

- **Invalid values** (e.g., negative values for a variable that is only positive)

- Features that bring no additional information: they contain the same **value** for all observations or different values for all observations with no useful pattern

- **Duplicate observations**

- Observations/Features that have **missing elements**

Corrupt and invalid values (1/4)

How to detect **corrupt or invalid values** in a dataset?

For **values that are numbers**: use **df.describe()** to understand what is going on

Does anything seem off to you?

```
Titanic.describe()
```

	PassengerId	Age	SibSp	Parch	Fare
count	893.000000	716.000000	893.000000	893.000000	893.000000
mean	447.000000	29.649679	0.521837	0.380739	32.076091
std	257.931192	14.540967	1.101784	0.805355	49.725466
min	1.000000	0.420000	0.000000	0.000000	-50.000000
25%	224.000000	20.000000	0.000000	0.000000	7.895800
50%	447.000000	28.000000	0.000000	0.000000	14.454200
75%	670.000000	38.000000	1.000000	0.000000	31.000000
max	893.000000	80.000000	8.000000	6.000000	512.329200

- A fare should be positive…
- Everything seems okay except for this
- Where does the negative fare occur?

Corrupt and invalid values (2/4)

We filter the dataset based on the fare being negative:

```
Titanic[Titanic["Fare"]<0]
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
892	893	err	err	err	5.0	0	0	err	-50.0	err	err

Row 892 is completely corrupt! We need to remove it… How to delete it?

df.drop(index=number)

```
Titanic=Titanic.drop(index=892)
```

```
Titanic.describe()
```

	PassengerId	Age	SibSp	Parch	Fare
count	892.000000	715.000000	892.000000	892.000000	892.000000
mean	446.500000	29.684154	0.522422	0.381166	32.168105
std	257.642517	14.521835	1.102264	0.805706	49.677238
min	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.750000	20.000000	0.000000	0.000000	7.895800
50%	446.500000	28.000000	0.000000	0.000000	14.454200
75%	669.250000	38.000000	1.000000	0.000000	31.000000
max	892.000000	80.000000	8.000000	6.000000	512.329200

All good now!

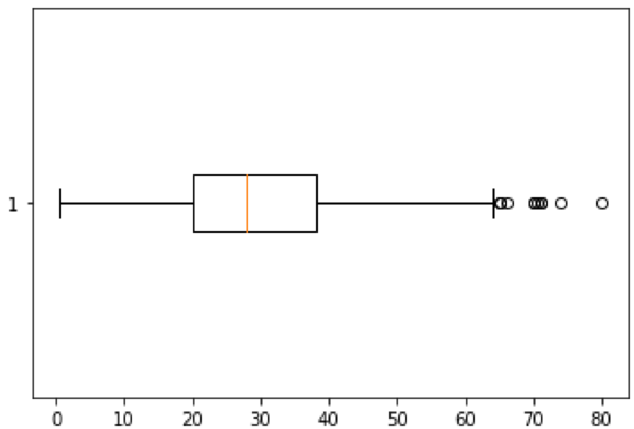
Corrupt and invalid values (3/4)

Looking for outliers:

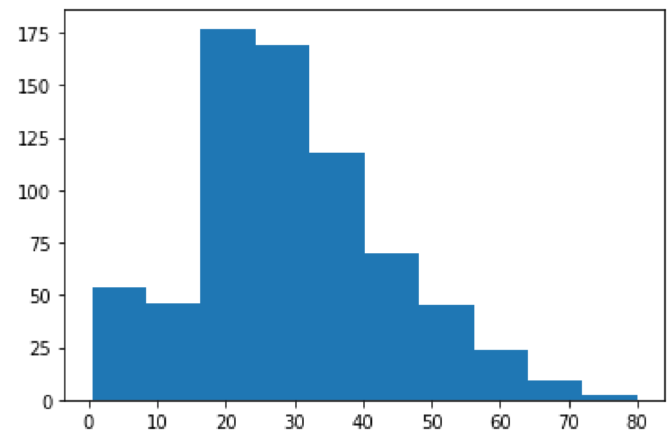
An outlier is an observation that doesn't "fit" into your dataset.

- This can be due to corrupt data, typos, or real outliers (e.g., Michael Jordan)
- Three main ways of checking for outliers: boxplots, Z-score charts (numbers above 3 or below -3), anomaly detection (unsupervised learning technique)
- Serves to flag possible outliers: area-specific knowledge to discard/keep

```
plt.boxplot(Titanic.loc[~Titanic["Age"].isna(),"Age"],vert=False)
```




```
plt.hist(Titanic["Age"])
```



Corrupt and invalid values (4/4)

How to detect **corrupt or invalid values** in a dataset?

For **values that are strings or objects**: use **df[“column”].unique()**

Does anything seem off to you?

```
Titanic["Pclass"].unique()
```

```
array(['third', 'first', 'second'], dtype=object)
```

```
Titanic["Name"].unique().shape
```

```
(891,)
```

```
Titanic["Sex"].unique()
```

```
array(['male', 'female'], dtype=object)
```

```
Titanic["Ticket"].unique().shape
```

```
(681,)
```

```
Titanic["Cabin"].unique().shape
#some are missing: there are "nan"
```

```
(148,)
```

```
Titanic["Embarked"].unique()
```

```
array(['S', 'C', 'Q', nan], dtype=object)
```

- Two passengers share the same name
- Some passengers share ticket numbers
- There are missing values in “embarked” and “cabin”
- Valid values otherwise

Features with no information

How to **delete columns/features?**

- **Delete** features which are unique to each observation and bring no additional info (e.g., allocated at random)
- **Delete** features which are the same for every observation

What could we delete here?

Use df.drop(columns=[“column1”, “column2”])

```
Titanic=Titanic.drop(columns=["PassengerId"])
```

```
Titanic
```

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	third	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	first	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	third	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	first	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	third	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...
887	first	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	third	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	first	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	third	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q
891	third	Johnson, Mr. William Cahoone Jr	male	19.0	0	0	LINE	0.0000	NaN	S

892 rows × 10 columns

Duplicates

How to detect and delete **duplicate rows?**

Your turn!

```
dups=Titanic.duplicated() #checks each row of the dataset and returns TRUE or FALSE depending on whether it is a duplicate
print(dups.any()) #returns TRUE if there is any value in dups that is equal to TRUE
Titanic[dups] #returns the problematic row
```

```
True
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
891	0	third	Johnson, Mr. William Cahoone Jr	male	19.0	0	0	LINE	0.0	NaN	S

Don’t need to run this every time: can simply delete duplicates in an automated way using Python:

```
print(Titanic.shape) #gives current size of dataset
Titanic.drop_duplicates(inplace=True) # delete duplicate rows
print(Titanic.shape)
```

```
(892, 10)
(891, 10)
```

Scaling and Normalizing (1/2)

What is **scaling/normalizing** data? Only for **numerical features.**

Makes sure that your data is on scales that are comparable.

Normalizing: operation that ensures that your data is **between 0 and 1**

Scaling: operation that ensures that the **mean** of your data is **0** and the **std dev** is **1**

Your turn!

```
from sklearn import preprocessing
```

```
X = np.array([[ 1., -1.,  2.],
...                [ 2.,  0.,  0.],
...                [ 0.,  1., -1.]])
```

```
min_max_scaler=preprocessing.MinMaxScaler()
X_minmax = min_max_scaler.fit_transform(X)
X_minmax
```

```
array([[0.5       , 0.        , 1.        ],
       [1.        , 0.5       , 0.33333333],
       [0.        , 1.        , 0.        ]])
```

Normalizing

```
X_scaled = preprocessing.scale(X)
X_scaled
```

```
array([[ 0.        , -1.22474487,  1.33630621],
       [ 1.22474487,  0.        , -0.26726124],
       [-1.22474487,  1.22474487, -1.06904497]])
```

Scaling

Scaling and Normalizing (2/2)

When should I use it?

- **Useful for certain machine learning algorithms only**.
  - Useful for regressions, PCA. Not for tree algorithms.
  - If your algorithm takes features and multiplies them by numbers etc., then chances are scaling/normalizing could improve it.

- Some use cases:
  - Some columns are **orders of magnitude different** (e.g., column A has values around 1 and column B has values around 10,000,000,000)
  - Your algorithm is returning **warning message** of the type “poor condition number”
  - The **output** you get from your algorithm is **incomprehensible** (e.g., NAs)

Data Imputation (1/5)

INSEAD

How to **detect empty cells in the rows or columns**?

df.isna().any() tells you which columns are empty.
df.isna().sum() tells you how many of the entries are empty.

Your turn!

```
Titanic.isna().any()
```

```
Pclass      False
Name        False
Sex         False
Age          True
SibSp       False
Parch       False
Ticket      False
Fare        False
Cabin        True
Embarked     True
dtype: bool
```

```
Titanic.isna().sum()/891
```

```
Pclass      0.000000
Name        0.000000
Sex         0.000000
Age         0.198653
SibSp       0.000000
Parch       0.000000
Ticket      0.000000
Fare        0.000000
Cabin       0.771044
Embarked    0.002245
dtype: float64
```

Three columns have missing entries:
Age, Cabin, Embarked

Activity (with your neighbor): what could be different ways of dealing with the empty **Age** cells?
How would this change if you consider a missing observation for the **Embarked** feature?

Data imputation (2/5)

Easy

Numerical variables

Delete the row(s)/column(s) where values are missing

Replace the value with the **mean/the largest value/the smallest value**

Find the observation that is **“closest”** to it in other observations and use the value there

Find a **couple of observations** that are “close” to it and **randomly pick one of them**

Run a **regression** on rows where all the data is present and infer from it the missing values

Run a regression on rows where all the data is present and infer from it the missing values then **add noise** to the missing values

Hard

Data imputation (3/5)

Categorical variables

Easy

Delete the row(s)/column(s) where values are missing

Create a new category: missing values

Replace the value with the **value that appears most/least**

Find the observation that is **“closest”** to it in other observations and use the value there

Find observations that are **close** to it in other observations and randomly pick one

Run a **prediction algorithm** on rows that outputs a categorical variable (Lecture 5)

Hard

Data imputation (4/5)

	Sales	Size	Color
1	221	NA	Blue
2	157	Large	NA
3	NA	Medium	Red
4	50	NA	Green
5	122	Large	Red

Missing completely at random
(no pattern to the missing entries)

	Age	Mammography results
1	23	NA
2	55	Negative
3	34	Positive
4	18	NA
5	62	Positive

Missing at random
(absent entries depend on another feature)

	Weight (kgs)	Age	Diabetes
1	80	77	1
2	90	40	0
3	NA	62	1
4	50	18	0
5	NA	54	1

Missing not at random
(entries are absent due to their value or a feature not accounted for)

- The methods discussed work well for **missing completely at random/missing at random**
- For **Missing not at random**: much harder, can be mitigated by adding features

Data Imputation (5/5)

Back to Titanic dataset…

```
Titanic.isna().sum()/891
```

```
Pclass      0.000000
Name        0.000000
Sex         0.000000
Age         0.198653
SibSp       0.000000
Parch       0.000000
Ticket      0.000000
Fare        0.000000
Cabin       0.771044
Embarked    0.002245
dtype: float64
```

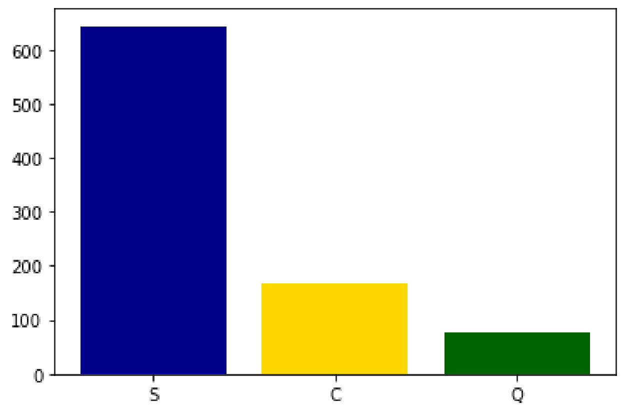
- Cabin has too many missing values

⇒ we drop this column: **df.drop(columns=[“column1”])**

- What do with Age and Embarked?

Use the package **sklearn.impute**

Embarked: most frequent value is “S”.
⇒ Replace missing with **S** (see notebook)



Age: replace, e.g., with the age of the observation that is closest on other features (see notebook)

Feature Engineering

Numerical ↔ Categorical (1/3)

- Some algorithms only accept one kind of input (generally numerical, e.g., regression).
- Useful to know how to go from one “type” of data to another

Categorical → Numerical

Activity (with neighbor): can you see a difference between these two sets of categorical entries? How would you propose to make them numbers?

Pclass
Second
First
Third

Embarked
S
C
Q

Numerical ↔ Categorical (2/3)

INSEAD

Pclass
Second
First
Third

⇒

Pclass
2
1
3

Ordinal variables
(these entries can be ranked)

Exercise left for homework

Embarked
S
C
Q

Nominal variables
(these entries cannot be ranked)

⇒

	S	C	Q
Observation 1	1	0	0
Observation 2	0	1	0
Observation 3	0	0	1

One-Hot Encoding

⇒

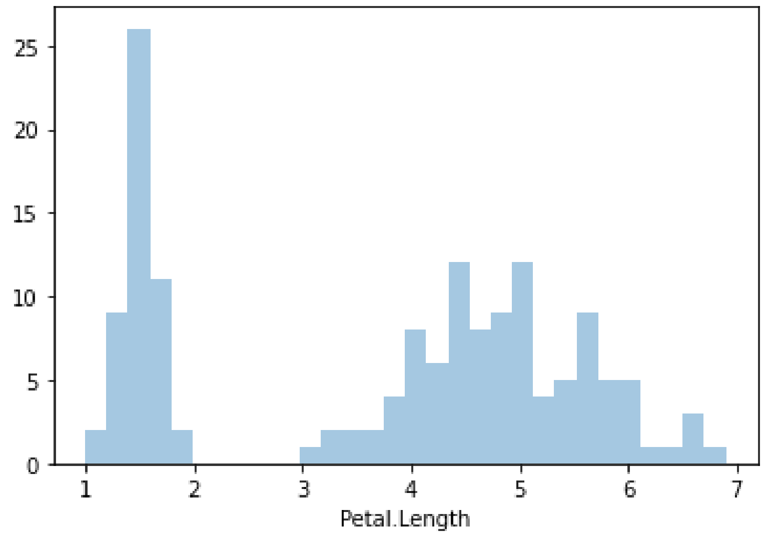
	S	C
Observation 1	1	0
Observation 2	0	1
Observation 3	0	0

Drop redundant column

```
Titanic=pd.get_dummies(Titanic,columns=["Sex","Embarked"], drop_first=True)
```

Numerical ↔ Categorical (3/3)

Numerical → Categorical



Idea: use bucketization or data binning. Take average for each bucket. Number of buckets=number of categories.

Can also serve to aggregate observations.

Feature selection/dimension reduction

- **Feature selection**

  Involves picking the **“right” features** for the model out of all possible features

- **Dimension reduction**

  - Involves “**merging**” features together to get as **few features** as possible to explain the variability in the data
  - Covered in unsupervised learning (Lecture 8)

Transforms & interactions

- Depending on the set-up it may be useful to **transform a feature**:
  - take powers of it
  - subtract/add a constant to it
  - divide/multiply by a constant
  - take an exponential of it or a log

Example in your homework on“Fare”: converting pounds from 1912 to today’s euros.

- **Feature interactions** involve adding/multiplying/dividing etc. two features together to obtain a new feature.

Example in your homework linking “ParCh” and “SibSp”

Wrap-up & Next time

Today, we:

- Discussed some challenges faced to deal with (1) corrupt data; (2) duplicates; (3) scaling issues; (4) missing data
- Understood how to go from numerical ↔ categorical
- Defined feature selection/transforms and dimension reduction

Taking a step back:

- Importance of business knowledge when doing data pre-processing
- Fundamental role of data pre-processing to get a good ML model

Next time:

- Regression recap and the basics of supervised learning
- Mini-quiz at the beginning of lecture + homework

INSEAD

The Business School for the World®

EUROPE | ASIA | MIDDLE EAST | NORTH AMERICA