Feature Engineering
(Data Engineering)

Stephen Choi Ph.D.
Artificial Intelligence Program Director
Associate Professor of Information Systems
Department of Information Systems and Decision Sciences
Craig School of Business
California State University Fresno
Fresno, CA 93740
choi@csufresno.edu

What is Feature Engineering?

- Feature engineering is the process of transforming raw data into meaningful features that can be used to improve the performance of machine learning models.
- It involves selecting, creating, and transforming variables to provide more relevant information for the model to learn from.




The Significance of Feature Engineering

The significance of feature engineering in machine learning is substantial.

- ✓ It enhances model accuracy by providing more meaningful and relevant information.
- ✓ It can uncover hidden patterns and relationships in the data.
- ✓ It simplifies complex data transformations, making it easier for models to understand and process the information.
- ✓ It's crucial for addressing real-world business problems effectively.

ML engineers often spend up to 80% of their time on feature engineering, highlighting its importance in the machine learning pipeline.

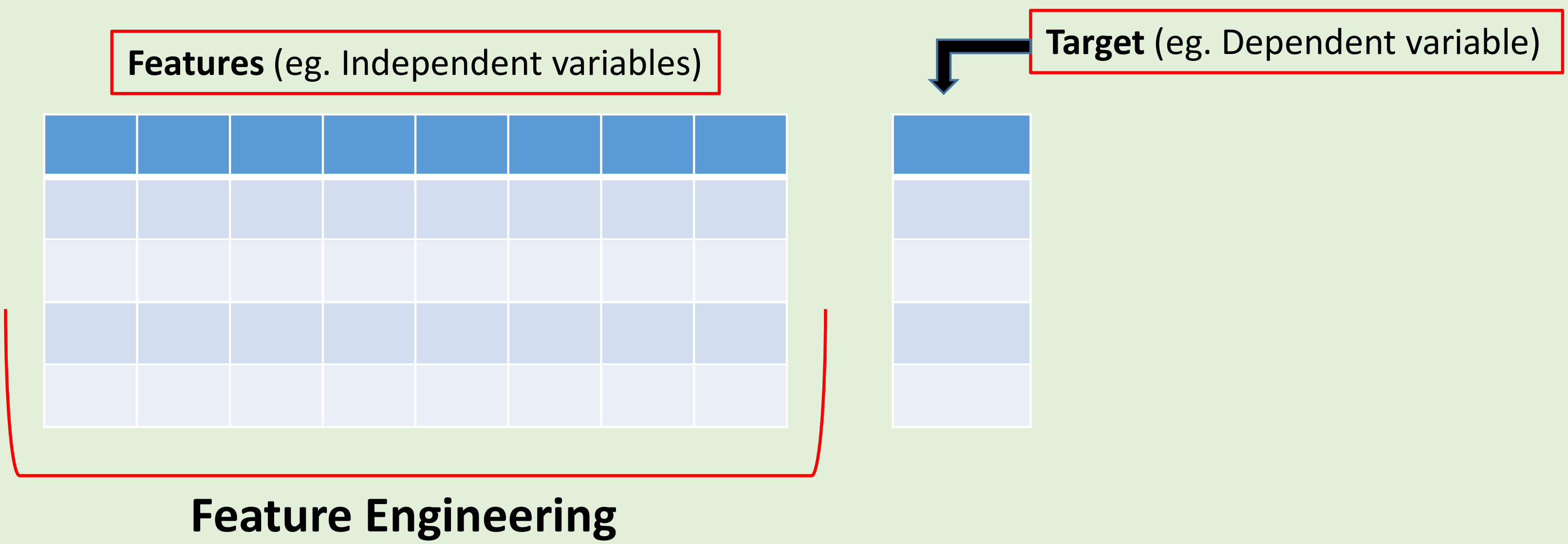
Major Companies in Feature Engineering

Several companies provide feature engineering services and tools:

- dotData
- Databricks
- IBM
- AlteryxTibil Solutions

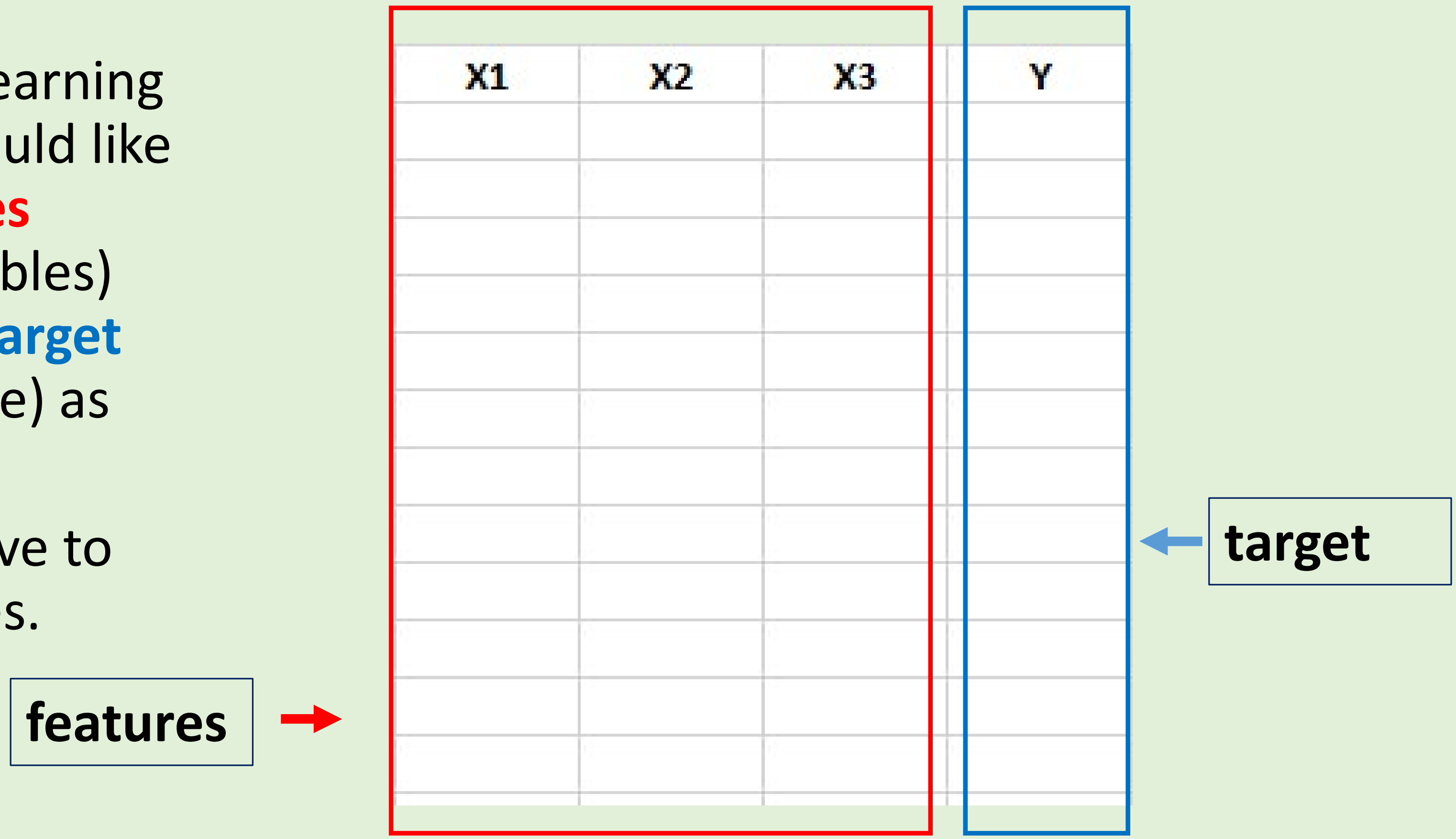
These companies typically service other businesses by providing tools and platforms that enable more efficient and effective feature engineering, ultimately improving the performance of machine learning models across various industries.

Feature Engineering – this is the term we use in referring to some of the feature (independent variable) value manipulation or conversion steps that are required before the algorithm function starts.



Typical Setting for Supervised Learning

- In the supervised learning algorithms, you would like to have the **features** (independent variables) first and have the **target** (dependent variable) as the last column.
- IF not, then you have to make some changes.



Most common Feature Engineering maneuvers

Most common maneuvers

- ✓ Deal with missing or null values
- ✓ String to integer conversion (LabelEncoder - the first value gets “0”label)
- ✓ One-Hot Encoding (ref: Association Rules-TransactionEncoder)
- ✓ Data normalization
- ✓ Column manipulation

Deal with missing or null values

Deal with missing or null values

- There are three ways we can clean the missing data.
  - **replace** the missing data with another value
  - **fill in** the missing data using existing data
  - **drop** the data from our data set.

Drop Missing Values

- Common way to deal with the missing values is simply dropping them.
- We can use the dropna method to drop missing values.

```
df1 = df.dropna()
```

```
df
   a  b
0  1  2
1  2  3
2  3  4
3  4  6
4  0  0
5  0  5
6  0  3
7  0  8

df['a']=df['a'].replace(0,df['a'].mean())

df
   a  b
0  1  2
1  2  3
2  3  4
3  4  6
4  1  0
5  1  5
6  1  3
7  1  8
```

Replace the zero values with the mean value

Replace the missing data with another value

- We can use the **fillna** method to recode the missing values to another value. For example, suppose we wanted the missing values to be recoded as a **0**.

Rows (index)

Columns

```
print(ebola.fillna(0).iloc[0:10, 0:5])
```

	Date	Day	Cases_Guinea	Cases_Liberia	Cases_SierraLeone
0	1/5/2015	289	2776.0	0.0	10030.0
1	1/4/2015	288	2775.0	0.0	9780.0
2	1/3/2015	287	2769.0	8166.0	9722.0
3	1/2/2015	286	0.0	8157.0	0.0
4	12/31/2014	284	2730.0	8115.0	9633.0
5	12/28/2014	281	2706.0	8018.0	9446.0
6	12/27/2014	280	2695.0	0.0	9409.0
7	12/24/2014	277	2630.0	7977.0	9203.0
8	12/21/2014	273	2597.0	0.0	9004.0
9	12/20/2014	272	2571.0	7862.0	8939.0

```
>>> df
       name        toy       born
0    Alfred        NaN        NaT
1    Batman  Batmobile 1940-04-25
2  Catwoman   Bullwhip        NaT
```

Drop the rows where at least one element is missing.

```
>>> df.dropna()
     name        toy       born
1  Batman  Batmobile 1940-04-25
```

Drop the columns where at least one element is missing.

```
>>> df.dropna(axis='columns')
       name
0    Alfred
1    Batman
2  Catwoman
```

Drop the rows where all elements are missing.

```
>>> df.dropna(how='all')
       name        toy       born
0    Alfred        NaN        NaT
1    Batman  Batmobile 1940-04-25
2  Catwoman   Bullwhip        NaT
```

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html

```
>>> df
       name        toy       born
0    Alfred        NaN        NaT
1    Batman  Batmobile 1940-04-25
2  Catwoman   Bullwhip        NaT
```

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html

Keep only the rows with at least 2 non-NA values.

```
>>> df.dropna(thresh=2)
       name        toy       born
1    Batman  Batmobile 1940-04-25
2  Catwoman   Bullwhip        NaT
```

Define in which columns to look for missing values.

```
>>> df.dropna(subset=['name', 'toy'])
       name        toy       born
1    Batman  Batmobile 1940-04-25
2  Catwoman   Bullwhip        NaT
```

Drop a row if there are NaN in **both** of the columns

Keep the DataFrame with valid entries in the same variable.

```
>>> df.dropna(inplace=True)
>>> df
     name        toy       born
1  Batman  Batmobile 1940-04-25
```

Only the full entries, no missing values

How to deal with missing values

```
import pandas as pd
import numpy as np

df=pd.read_csv("basketball.csv")
df
```

	team	pts	assists	rebounds
0	A	25.0	5.0	11
1	NaN	NaN	7.0	8
2	B	15.0	7.0	10
3	B	NaN	9.0	6
4	B	19.0	12.0	6
5	C	23.0	9.0	5
6	C	25.0	NaN	9
7	C	29.0	4.0	12

```
[19] print(df.isnull().sum())
```

```
team        1
pts         2
assists     1
rebounds    0
dtype: int64
```

```
#replace NaN with zero
df.fillna(value=0, inplace=True)
df
```

	team	pts	assists	rebounds
0	A	25.0	5.0	11
1	0	0.0	7.0	8
2	B	15.0	7.0	10
3	B	0.0	9.0	6
4	B	19.0	12.0	6
5	C	23.0	9.0	5
6	C	25.0	0.0	9
7	C	29.0	4.0	12

```python
#replace NaN with zero for the pts column

df=pd.read_csv("basketball.csv")
df['assists']=df['assists'].fillna(value=0)
df
```

	team	pts	assists	rebounds
0	A	25.0	5.0	11
1	NaN	NaN	7.0	8
2	B	15.0	7.0	10
3	B	NaN	9.0	6
4	B	19.0	12.0	6
5	C	23.0	9.0	5
6	C	25.0	0.0	9
7	C	29.0	4.0	12

```
[24] #replace NaN with zeo for the pts & assists columns

df[['pts','assists']]=df[['pts','assists']].fillna(value=0)
df
```

	team	pts	assists	rebounds
0	A	25.0	5.0	11
1	NaN	0.0	7.0	8
2	B	15.0	7.0	10
3	B	0.0	9.0	6
4	B	19.0	12.0	6
5	C	23.0	9.0	5
6	C	25.0	0.0	9
7	C	29.0	4.0	12

```
[26] #this mean values do NOT count the zero values
     df=pd.read_csv("basketball.csv")
     mean_values=df[['pts','assists']].mean()
     print(mean_values)
```

```
pts          22.666667
assists       7.571429
dtype: float64
```

```
[27] #replacing the missing values with the mean values
df[['pts','assists']]=df[['pts','assists']].fillna(value=df[['pts','assists']].mean())
df
```

	team	pts	assists	rebounds
0	A	25.000000	5.000000	11
1	NaN	22.666667	7.000000	8
2	B	15.000000	7.000000	10
3	B	22.666667	9.000000	6
4	B	19.000000	12.000000	6
5	C	23.000000	9.000000	5
6	C	25.000000	7.571429	9
7	C	29.000000	4.000000	12

How to Convert String to Numeric Values

Numeric values required

- Any categorical values must be converted to numeric values.
- For example:

Yes/No	→	0 or 1
Purchase/ No Purchase	→	0 or 1
disease X/ disease Y/ disease Z	→	0, 1 or 2

How to convert categorical to numeric values

- Here is a CSV file with its target column with the categorical values: “**Yes or NO**”

- We need to convert this to **0 or 1**

```
#importing data using .read_csv() function
df=pd.read_csv('data.csv')

df
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

- LabelEncoder () is the function that makes the conversion.
- Upon executing it, the output shows the 0s and 1s where 0 is “No” and 1 is “Yes”. This is temporarily saved as the ‘label’ column.

```
[ ]
# Importing LabelEncoder from Sklearn
# library from preprocessing Module.
from sklearn.preprocessing import LabelEncoder

# Creating a instance of label Encoder.
le = LabelEncoder()

# Using .fit_transform function to fit label
# encoder and return encoded label
label = le.fit_transform(df['Purchased'])

# printing label
label
```

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

- The original “Purchased” column is dropped.
- The label column is now saved as the “Purchased” column.

```
[ ]
    # removing the column 'Purchased' from df
    # as it is of no use now.
    df.drop("Purchased", axis=1, inplace=True)

    # Appending the array to our dataFrame
    # with column name 'Purchased'
    df["Purchased"] = label

    # printing Dataframe
    df
```

- We have successfully converted the “Purchased” column to numeric values, 0 or 1. (we can do the same with the country column, if we need to).

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	0
1	Spain	27.0	48000.0	1
2	Germany	30.0	54000.0	0
3	Spain	38.0	61000.0	0
4	Germany	40.0	NaN	1
5	France	35.0	58000.0	1
6	Spain	NaN	52000.0	0
7	France	48.0	79000.0	1
8	Germany	50.0	83000.0	0
9	France	37.0	67000.0	1

Let's do one more example

- The file is 'ecoli.csv'
- The target column has the categorical values: **cp, im, imS, imL, imU, om, omL, and pp.**
- We want these to be converted into integers.

```
#importing data using .read_csv() function
df = pd.read_csv('ecoli.csv')

#printing DataFrame
df
```

	seq_num	mcg	gvh	lip	chg	aac	alm1	alm2	target
0	AAT_ECOLI	0.49	0.29	0.48	0.5	0.56	0.24	0.35	cp
1	ACEA_ECOLI	0.07	0.40	0.48	0.5	0.54	0.35	0.44	cp
2	ACEK_ECOLI	0.56	0.40	0.48	0.5	0.49	0.37	0.46	cp
3	ACKA_ECOLI	0.59	0.49	0.48	0.5	0.52	0.45	0.36	cp
4	ADI_ECOLI	0.23	0.32	0.48	0.5	0.55	0.25	0.35	cp
...	...	...	...	...	...	...	...	...	...
331	TREA_ECOLI	0.74	0.56	0.48	0.5	0.47	0.68	0.30	pp
332	UGPB_ECOLI	0.71	0.57	0.48	0.5	0.48	0.35	0.32	pp
333	USHA_ECOLI	0.61	0.60	0.48	0.5	0.44	0.39	0.38	pp
334	XYLF_ECOLI	0.59	0.61	0.48	0.5	0.42	0.42	0.37	pp
335	YTFQ_ECOLI	0.74	0.74	0.48	0.5	0.31	0.53	0.52	pp

336 rows × 9 columns

- LabEncoder() is used for the conversion.
- ‘target’ column is processed and saved temporarily as ‘label’

```python
# Importing LabelEncoder from Sklearn
# library from preprocessing Module.
from sklearn.preprocessing import LabelEncoder

# Creating a instance of label Encoder.
le = LabelEncoder()

# Using .fit_transform function to fit label
# encoder and return encoded label
label = le.fit_transform(df['target'])

# printing label
label
```

- Here are the numeric values under the ‘label’ column. There are 336 of them as there are 336rows/records.
- Given these, we can assume that the following:

cp	**0**
im	**1**
imS	**2**
imL	**3**
imU	**4**
om	**5**
omL	**6**
pp	**7**

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       3, 3, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7,
       7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
       7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
       7, 7, 7, 7, 7, 7])
```

336 rows × 9 columns

- The original “target” column is dropped.
- The label column is now saved as the “target” column.

```python
# removing the column 'target' from df
# as it is of no use now.
df.drop("target", axis=1, inplace=True)

# Appending the array to our dataFrame
# with column name 'target'
df["target"] = label

# printing Dataframe
df
```

- We have successfully converted the “target” column to numeric values, 0 thru 7.

	seq_num	mcg	gvh	lip	chg	aac	alm1	alm2	target
0	AAT_ECOLI	0.49	0.29	0.48	0.5	0.56	0.24	0.35	0
1	ACEA_ECOLI	0.07	0.40	0.48	0.5	0.54	0.35	0.44	0
2	ACEK_ECOLI	0.56	0.40	0.48	0.5	0.49	0.37	0.46	0
3	ACKA_ECOLI	0.59	0.49	0.48	0.5	0.52	0.45	0.36	0
4	ADI_ECOLI	0.23	0.32	0.48	0.5	0.55	0.25	0.35	0
...	...	...	...	...	...	...	...	...	...
331	TREA_ECOLI	0.74	0.56	0.48	0.5	0.47	0.68	0.30	7
332	UGPB_ECOLI	0.71	0.57	0.48	0.5	0.48	0.35	0.32	7
333	USHA_ECOLI	0.61	0.60	0.48	0.5	0.44	0.39	0.38	7
334	XYLF_ECOLI	0.59	0.61	0.48	0.5	0.42	0.42	0.37	7
335	YTFQ_ECOLI	0.74	0.74	0.48	0.5	0.31	0.53	0.52	7

336 rows × 9 columns

One-Hot Encoding

```
[['MILK', 'BREAD', 'BISCUIT'],
 ['BREAD', 'MILK', 'BISCUIT', 'CORNFLAKES'],
 ['BREAD', 'TEA', 'BOURNVITA'],
 ['JAM', 'MAGGI', 'BREAD', 'MILK'],
 ['MAGGI', 'TEA', 'BISCUIT'],
 ['BREAD', 'TEA', 'BOURNVITA'],
 ['MAGGI', 'TEA', 'CORNFLAKES'],
 ['MAGGI', 'BREAD', 'TEA', 'BISCUIT'],
 ['JAM', 'MAGGI', 'BREAD', 'TEA'],
 ['BREAD', 'MILK'],
 ['COFFEE', 'COCK', 'BISCUIT', 'CORNFLAKES'],
 ['COFFEE', 'COCK', 'BISCUIT', 'CORNFLAKES'],
 ['COFFEE', 'SUGER', 'BOURNVITA'],
 ['BREAD', 'COFFEE', 'COCK'],
 ['BREAD', 'SUGER', 'BISCUIT'],
 ['COFFEE', 'SUGER', 'CORNFLAKES'],
 ['BREAD', 'SUGER', 'BOURNVITA'],
 ['BREAD', 'COFFEE', 'SUGER'],
 ['BREAD', 'COFFEE', 'SUGER'],
 ['TEA', 'MILK', 'COFFEE', 'CORNFLAKES']]
```

```python
[ ]
#Let's transform the list, with one-hot encoding
from mlxtend.preprocessing import TransactionEncoder
a=TransactionEncoder()
a_data=a.fit(data).transform(data)
df=pd.DataFrame(a_data, columns=a.columns_)
df=df.replace(False,0)
df
```

The data presents many transactions of items, some are present but some are not according to all categories.

```python
[ ]  #Let's transform the list, with one-hot encoding
     from mlxtend.preprocessing import TransactionEncoder
     a=TransactionEncoder()
     a_data=a.fit(data).transform(data)
     df=pd.DataFrame(a_data, columns=a.columns_)
     df=df.replace(False,0)
     df
```

	BISCUIT	BOURNVITA	BREAD	COCK	COFFEE	CORNFLAKES	JAM	MAGGI	MILK	SUGER	TEA
0	True	0	True	0	0	0	0	0	True	0	0
1	True	0	True	0	0	True	0	0	True	0	0
2	0	True	True	0	0	0	0	0	0	0	True
3	0	0	True	0	0	0	True	True	True	0	0
4	True	0	0	0	0	0	0	True	0	0	True
5	0	True	True	0	0	0	0	0	0	0	True
6	0	0	0	0	0	True	0	True	0	0	True
7	True	0	True	0	0	0	0	True	0	0	True
8	0	0	True	0	0	0	True	True	0	0	True
9	0	0	True	0	0	0	0	0	True	0	0
10	True	0	0	True	True	True	0	0	0	0	0
11	True	0	0	True	True	True	0	0	0	0	0
12	0	True	0	0	True	0	0	0	0	True	0
13	0	0	True	True	True	0	0	0	0	0	0
14	True	0	True	0	0	0	0	0	0	True	0
15	0	0	0	0	True	True	0	0	0	True	0
16	0	True	True	0	0	0	0	0	0	True	0

The **TransactionEncoder** identifies the items that are present in a transaction and label it with "**True**" and empty ones with 0.

Data Normalization

Data Normalization

- **Normalization** – the feature values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.
- **Standardization** – the feature mean value is set to 0 and the values are rescaled to a standard deviation of 1 (unit variance).
- **When to use which one??**
- Generally speaking, normalization is good to use when the data does not follow normal distribution and standardization is when the data follows normal distribution.

Examples for Data Normalization

- For example, let’s say there is a data column where one uses 0-10 scale and the other uses 0-100 scale. What would you do?

- Let’s say you are comparing two professors’ grading systems where one category is aggregated quiz score 0-100 and the other is aggregated project grades: A, B, C, D & F (and you convert these to 0,1,2,3)

```python
#Data Standardization/Normalization
#The max value of Insulin is 846 while the max value of DiabetesPedigreeFunction is 2.42
#If we don't standardize, the greater scale tends to dominate during the train period

from sklearn import preprocessing

#standardize the data
df_scaled = preprocessing.scale(df)

#after the standardization, we bring it back to the pandas dataframe
df_scaled=pd.DataFrame(df_scaled,columns=df.columns)
```

```python
#since we do not want to scale the Outcome column (which is the target variable that we are tryping to predict)
#let's use the original Outcome column

df_scaled['Outcome'] = df['Outcome']
df=df_scaled
```

Column Manipulation

Move Columns Around in Dataframe

- Let's say we have a csv file called simple.csv (below) and we want to relocate the columns b and x to the last column.

```
[4] import pandas as pd
    import numpy as np

    df=pd.read_csv('simple.csv')
    df
```

	a	b	x	y
0	1	2	3	-1
1	2	4	6	-2
2	3	6	9	-3
3	4	8	12	-4

```
old_cols=df.columns.values
new_cols=['a','y','b','x']
df=df.reindex(columns=new_cols)
df
```

	a	y	b	x
0	1	-1	2	3
1	2	-2	4	6
2	3	-3	6	9
3	4	-4	8	12

Rename the column headers

- Renaming the columns ‘**a**’ and ‘**b**’ to ‘**apple**’ and ‘**mellon**’.

	a	y	b	x
0	1	-1	2	3
1	2	-2	4	6
2	3	-3	6	9
3	4	-4	8	12

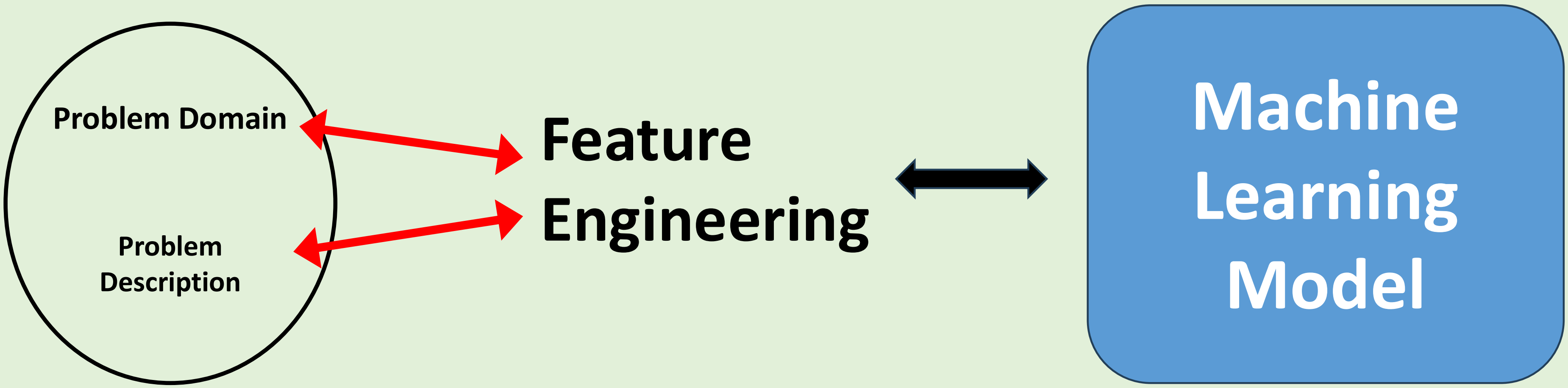
```
print(df.rename(columns={'a':'apple','b':'mellon'}))
```

```
   apple  y  mellon   x
0      1 -1       2   3
1      2 -2       4   6
2      3 -3       6   9
3      4 -4       8  12
```

We just finished the most common Feature Engineering maneuvers

When to Exercise the Feature Engineering?

- Always!
- Before you start, save the original data!
- Know your problem domain really well during your critical thinking!
- Start with the basics: missing or null values, string to integer conversion, one-hot encoding, data normalization.
- You will go thru cycles of this process until it is ready for the next step.



Know your data and study the target domain well!!

Q & A