Feature Engineering

[Slides from Fardina Fathmiul Alam]

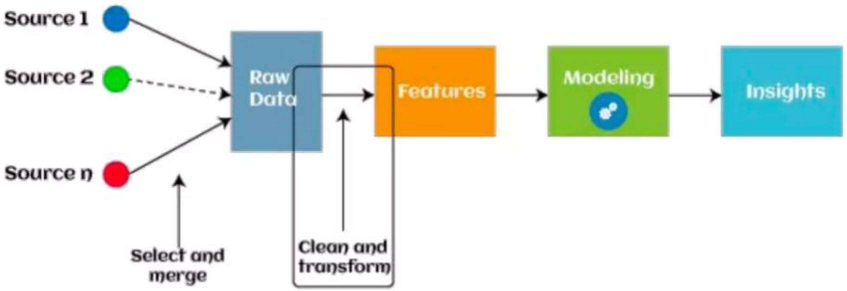
What we have covered so far

Data Preprocessing: This step involves cleaning the data, handling missing values, dealing with outliers, and any other data cleaning tasks.

Next: Before we build a ML Model, we need to apply "Feature Engineering"

What is “Feature” in a Dataset

Model **features (variables or columns)** are the inputs that machine learning (ML) models use during training and inference to make predictions.



Next: Before we build a ML Model, we need to apply "Feature Engineering"

Feature Engineering: This is where you transform the raw data into features that are suitable for machine learning algorithms.

- Involves selecting, transforming, and creating features from the raw data that will be used to train a machine learning model.

Next: Before we build a ML Model, we need to apply "Feature Engineering"

Feature Engineering: This is where you transform the raw data into features that are suitable for machine learning algorithms.

- Involves selecting, transforming, and creating features from the raw data that will be used to train a machine learning model.
- Produce new features for both supervised and unsupervised learning,

Remember, Feature engineering is crucial because the q**uality of features directly impacts the ML model's ability** to learn patterns and make accurate predictions.

Feature Engineering

Involves selecting, transforming, and creating features from the raw data to make them more suitable for the ML model. Where:

- **Selection:** Choose the **most relevant or most informative features** (with predictive power)to include in the model .with predictive power. Remove irrelevant or redundant ones.

Feature Engineering

Involves selecting, transforming, and creating features from the raw data to make them more suitable for the ML model. Where:

- **Transformation:** **Modify existing features** to make them more suitable for modeling. Techniques include scaling, normalization, binning, and mathematical transformations.

Feature Engineering

Involves selecting, transforming, and creating features from the raw data to make them more suitable for the ML model. Where:

- **Creation: Generate new features from existing ones or from domain knowledge.** Include combining features, creating interaction terms, or extracting useful information from text, images, or other unstructured data.

Feature Engineering: Example

REMEMBER: Feature engineering involves **creating new features from the existing ones** or **transforming the existing features** to better represent the underlying patterns in the data

House ID	Bedrooms	Bathrooms	Square Footage	Year Built	Sale Price
1	3	2	2000	1995	$300,000
2	4	2.5	2500	2005	$400,000
3	2	1.5	1500	1980	$250,000

Sample Data: House Prediction

Feature Engineering: Example

After Applying Feature Engineering

House ID	Bedrooms	Bathrooms	Square Footage	Year Built	Sale Price	Age of the House	Total Number of Rooms	Price per Square Foot
1	3	2	2000	1995	$300,000	28	5	$150/sq ft
2	4	2.5	2500	2005	$400,000	18	6.5	$160/sq ft
3	2	1.5	1500	1980	$250,000	43	3.5	$166.67/sq ft

These new features will help us understand a lot about our data.

Why Apply “Feature Engineering”

Goal: simplify and speed up data transformations while also enhancing model accuracy.

- The creation or modification of columns to make life easier for us or our machine learning models. Generally, mathematical models:
  - Don’t like categorical data
  - Don’t do super well with columns that have different scales
  - Are fussy in lots of other ways

Some Common ways to apply feature engineering

1. Transformation
   a. Normalization
   b. Standardization
   c. Log Transformation
2. Encoding categorical variables
   a. One-Hot Encoding
3. Discretization
4. Feature Selection
5. Feature Reduction

Common ways to apply feature engineering

Transformation: Altering the representation of data, e.g., taking the square root of a feature.

- **Normalization:** Scaling features to a standard range (e.g., [0, 1]).
- **Standardization:** Scaling features to have a mean of 0 and a standard deviation of 1.
- **Log Transformation:** Applying a logarithmic function to handle skewed data.

Common ways to apply feature engineering

Encoding → One-Hot Encoding: Converting categorical variables into binary (0/1) vectors.

Discretization → Binning: Grouping continuous values into discrete bins or categories.

Feature Creation→ Clustering: Creating features based on data similarity using techniques like k-means. [LATER TOPIC]

Feature Reduction→Dimensionality Reduction: Reducing the number of features while preserving relevant information (e.g., PCA). [LATER TOPIC]

Transforms

Transforms

Sometimes, we create a new column by applying a function over a column!

- **Data transformation** involves changing the format or values of data.
  - Why? make the data more suitable for modeling or to better represent the underlying patterns in the data.

Transforms can be **linear (scaling or shifting values)** or **non-linear (taking the logarithm or square root of a variable)**

Linear Transformation: Feature Scaling

In most cases, the **numerical features of the dataset do not have a certain range** and they differ from each other.

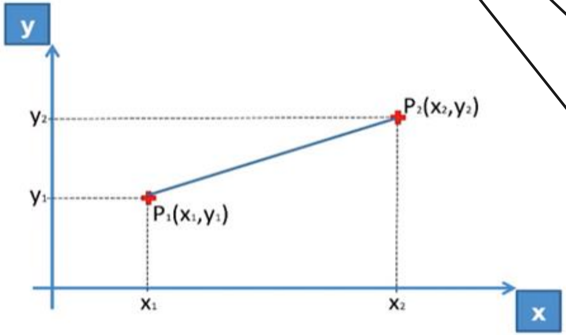
- Thing about ‘age’ and ‘income’ column range

The goal of Scaling is to make sure features (mostly continuous features) are on **almost the same scale (similar range) so that each feature** is equally important and make it easier to process by most ML algorithms.

NOTE: Scaling isn't mandatory for all algorithms but is crucial for those relying on distance calculations (e.g., k-NN) or weighted sums (e.g., linear regression) to ensure accurate predictions and model convergence

Let, X → Salary, Y → Age
Calculate Euclidean distance between row2 P1(x1,y1) and row 9 P2(x2,y2):

- The number of $(x2-x1)^2$ is much bigger than the number of $(y2-y1)^2$ which means the Euclidean distance will be dominated by the salary if we do not apply feature scaling. The difference in Age contributes less to the overall difference.



$$\text{Euclidean Distance between } P_1 \text{ and } P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

	Country	Age	Salary	Purchased
1	France	44	72000	No
2	Spain	27	48000	Yes
3	Germany	30	54000	No
4	Spain	38	61000	No
5	Germany	40		Yes
6	France	35	58000	Yes
7	Spain		52000	No
8	France	48	79000	Yes
9	Germany	50	83000	No
10	France	37	67000	Yes

```
dataset['Age'].min()
```

27.0

```
dataset['Salary'].min()
```

48000.0

```
dataset['Age'].max()
```

50.0

```
dataset['Salary'].max()
```

83000.0

The range of Age: 27 - 50

The range of Salary:48,000 - 83,000

Features with **larger scales will contribute more to the distance calculation**, potentially **biasing the results** towards those features

Let x be the no. of Salary and y be the no. of Age

Example: x1&y1 are in row 2, x2&y2 are in row 9

(x2-x1)^2= (83000-48000)^2

=1225000000

(y2-y1)^2=(50-27)^2

=529

Normalization or or Min-Max Scaling

Normalization (or min-max normalization) scale all values **in a fixed range between 0 and 1.**

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

The equation of Max-Min Normalization (Min-Max scaling)

Where:
X is the original feature value.
$\mathbf{X_{min}}$ = minimum value of the feature in the dataset.
$\mathbf{X_{max}}$ =maximum value of the feature in the dataset.
$\mathbf{X_{scaled}}$ = the scaled feature value.

Example: Normalization or or Min-Max Scaling

Normalization (or min-max normalization) scale all values **in a fixed range between 0 and 1.**

EXAMPLE: We have a dataset: given

- **Income in dollars** (ranges 0 to 12.53 billion)
- **Age in years** (ranges 0 to 100)

After normalization:

- Income: 0 -> 0, 12.53 billion -> 1
- Age: 0 -> 0, 100 -> 1
- Here min val: 0 and max val: 1

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Example: Normalization or Min-Max Scaling

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

#	Emp	Age	Salary
1	Emp1	44	73000
2	Emp2	27	47000
3	Emp3	30	53000
4	Emp4	38	62000
5	Emp5	40	57000
6	Emp6	35	53000
7	Emp7	48	78000

Normalization

Age	Normalized Age	Salary	Normalized Salary
44	0.80952381	73000	0.838709677
27	0	47000	0
30	0.142857143	53000	0.193548387
38	0.523809524	62000	0.483870968
40	0.619047619	57000	0.322580645
35	0.380952381	53000	0.193548387
48	1	78000	1
	Range 0-1		**Range 0-1**

How to calculate Normalized value?
X = 35, min = 27, max = 48 for column Age.
Xnorm(for 35) = $\frac{35-27}{48-27}$ = 0.3809

How to implement Normalization or Min-Max Scaling

```
data = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]})

data['normalized'] = (data['value'] - data['value'].min()) /
(data['value'].max() - data['value'].min())
```

	value	normalized
0	2	0.23
1	45	0.63
2	-23	0.00
3	85	1.00
4	28	0.47
5	2	0.23
6	35	0.54
7	-12	0.10

Good News! In Python, we can use **Sklearn library** to apply MinMaxScaler (more efficient and faster than a custom implementation, especially for large datasets.)

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(df)
scaled_features = scaler.transform(df)
#Convert to table format - MinMaxScaler
df_MinMax = pd.DataFrame(data=scaled_features, columns=["Age", "Salary","Purcha
```

Common uses of Normalization or Min-Max Scaling

Commonly used to assist ML algorithms, particularly those like K-Nearest Neighbors (K-NN) and Neural Networks, which do not assume any specific distribution of the data (E.g. the distribution of your data is not necessarily Gaussian).

- Does not significantly alter the distribution of the feature.
- Ensures no single feature dominates statistics.
    - Preserves relative distances between data points.

Normalization

Normalization is useful when you want to bound the values to a specific range. It preserves the relative proportions of the data, meaning the relationships between values remain the same.

- Can be sensitive to outliers because it depends on the minimum and maximum values in the dataset. Outliers can skew the scaling.

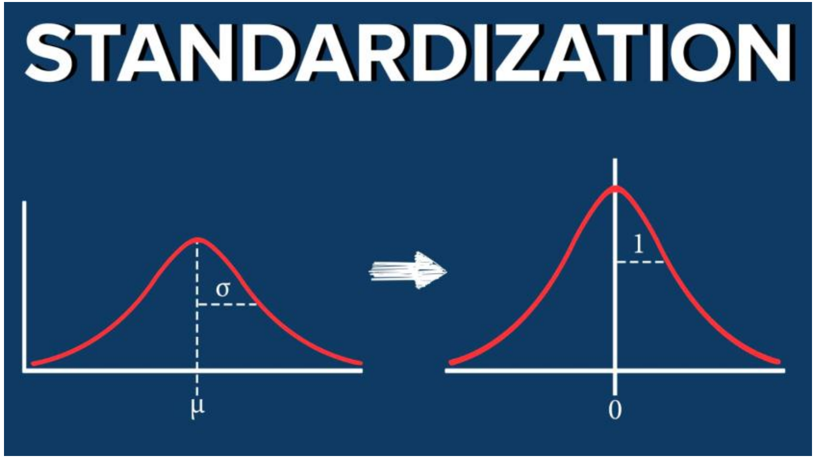
$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Standardization or Z-score normalization

Scales the values of features by subtracting their mean and considering their standard deviation. It does not alter the distribution shape but ensures that features have comparable scales.

$$z = \frac{x_i - \mu}{\sigma}$$

The features will be rescaled to ensure the **mean and the standard deviation to be 0 and 1**, respectively. (zero mean and unit variance).



- Assists algorithms that **expect features to follow a standard normal distribution** or use distance-based metrics, resulting in better performance and convergence, where the algorithm reaches a stable or optimal solution.

How to implement Standardization or Z-score normalization

```
data = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]})

data['standardized'] = (data['value'] - data['value'].mean()) /
data['value'].std()
```

	value	standardized
0	2	-0.52
1	45	0.70
2	-23	-1.23
3	85	1.84
4	28	0.22
5	2	-0.52
6	35	0.42
7	-12	-0.92

Good News! In Python, we can use **Sklearn library** to apply StandardScaler (more efficient and faster than a custom implementation, especially for large datasets.)

```
#Import library
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_X = sc_X.fit_transform(df)
#Convert to table format - StandardScaler
sc_X = pd.DataFrame(data=sc_X, columns=["Age", "Salary","Purchased","Country_Fr
sc_X
```

Standardisation vs Max-Min Normalization

Standardization is more robust to outliers, and in many cases, it is **preferable** over Max-Min Normalization.

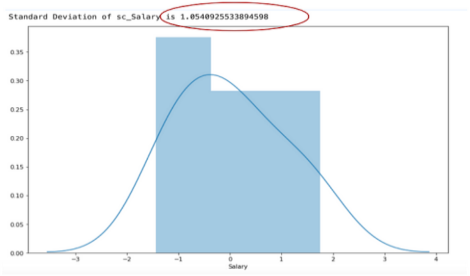
Standardisation vs Max-Min Normalization

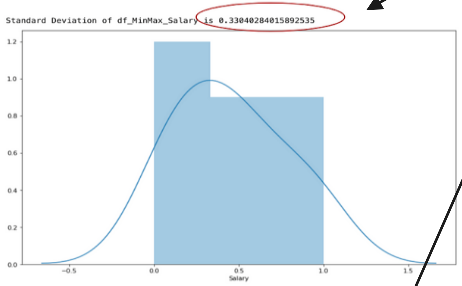
Notice: Max-Min Normalization yields smaller standard deviations compared to standardization

Column: Salary

Standard Deviation (Salary):
Max-Min Normalization (0.33) < Standardisation (1.05)

Standardisation	Max-Min Normalisation

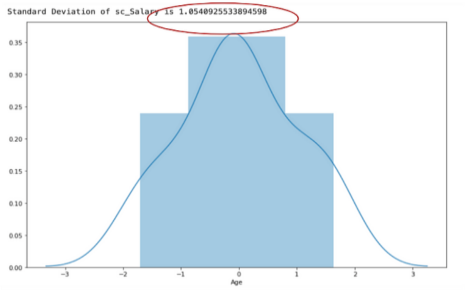


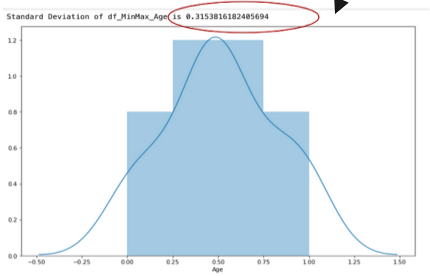


Column:Age

Standard Deviation (Age):
Max-Min Normalization (0.315) < Standardisation (1.05)

Standardisation	Max-Min Normalisation





Standardisation

	Age	Salary
0	0.758874	7.494733e-01
1	-1.711504	-1.438178e+00
2	-1.275555	-8.912655e-01
3	-0.113024	-2.532004e-01
4	0.177609	6.632192e-16
5	-0.548973	-5.266569e-01
6	0.000000	-1.073570e+00
7	1.340140	1.387538e+00
8	1.630773	1.752147e+00
9	-0.258340	2.937125e-01

Max-Min Normalization

	Age	Salary
0	0.739130	0.685714
1	0.000000	0.000000
2	0.130435	0.171429
3	0.478261	0.371429
4	0.565217	0.450794
5	0.347826	0.285714
6	0.512077	0.114286
7	0.913043	0.885714
8	1.000000	1.000000
9	0.434783	0.542857

Standardisation vs Max-Min Normalization

As Max-Min Normalization produce smaller standard deviations, increases the impact of outliers.

- Compresses data into a smaller range (0 to 1), making outliers have a disproportionate influence, distorting the overall data distribution.

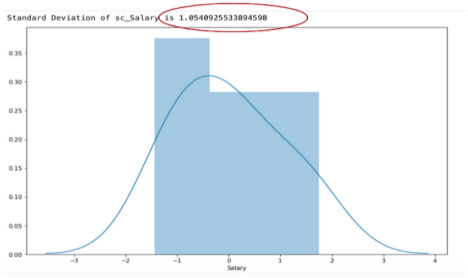
It's recommended to handle outliers before normalization to prevent bias and maintain accurate model performance.

On the other hand, Standardization is less sensitive to outliers because it centers the data around the mean and scales it by the standard deviation, which are less influenced by extreme values.

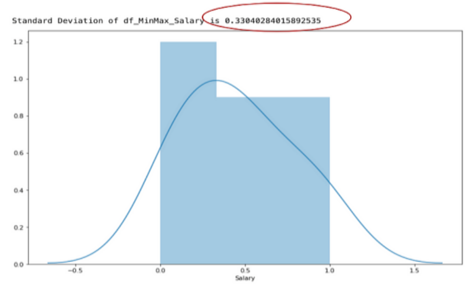
Column: Salary

Standard Deviation (Salary):
Max-Min Normalization (0.33) < Standardisation (1.05)

Standardisation



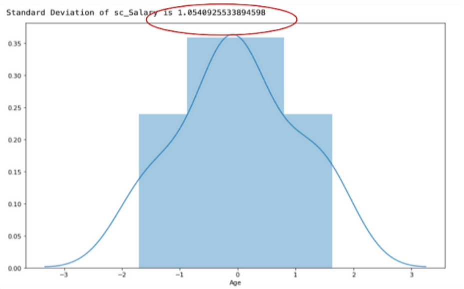
Max-Min Normalisation



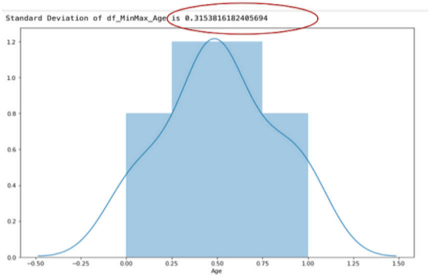
Column:Age

Standard Deviation (Age):
Max-Min Normalization (0.315) < Standardisation (1.05)

Standardisation



Max-Min Normalisation

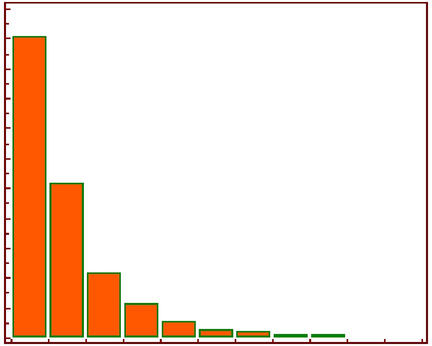


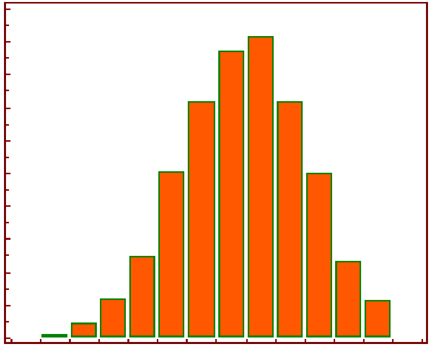
Log Transforms

It helps to handle skewed data and after transformation, the distribution becomes more approximate to normal (reduce skewness, spread the data and make it look more like a normal distribution).

How? Simply take the log of the distribution Log(x).

It can make the data more suitable for analysis techniques (t-test, Anova) that assume a more normal (bell-shaped) distribution of values.





Log Transforms

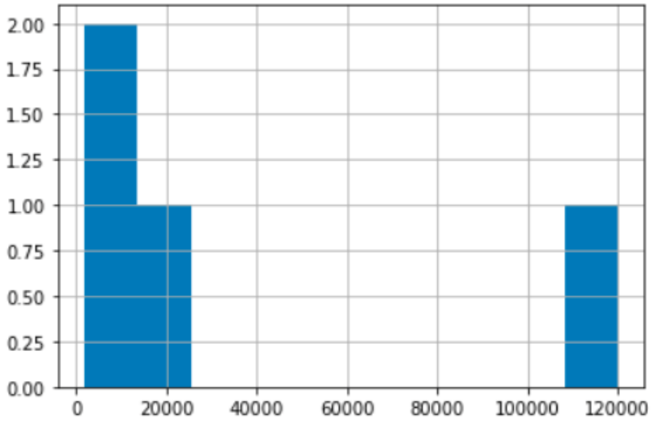


Fig: Plot the histogram of Income, it ranges from 0 to 1,20,000:

Let us see what happens when we apply log on this column:

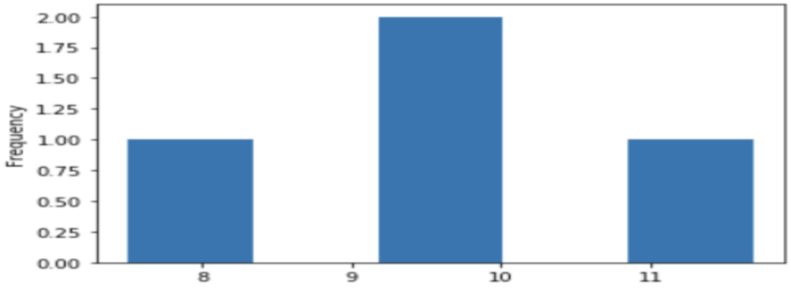
```
df['log_income'] = np.log(df['Income'])
# We created a new column to store the log values
```

This is how the dataframe looks like:

	Income	Age	Department	log_income
0	15000	25	HR	9.615805
1	1800	18	Legal	7.495542
2	120000	42	Marketing	11.695247
3	10000	51	Management	9.210340

Let us plot a histogram of the above, using 5 bins:

```
df['log_income'].plot.hist(bins = 5)
```



One Hot Encoding

Some Machine Learning Methods Hate Categoricals!

Index	Profession
1	Nurse
2	Doctor
3	Actor
4	Teacher
5	Nurse

- Jobs like "Nurse" or "Doctor" don't have an order.
- Labeling (like making Nurse 0 and Doctor 1) might make the model think one job is more important.
- One hot encoding makes each job a separate choice (binary feature) without a rank.
- It treats all jobs fairly, without thinking one is better than another.
    - It can be useful when data has no relationship to each other, or when the order of numbers is not significant.

One-hot encoding is a data preprocessing technique that converts categorical variables into binary vectors.

- Each category becomes a binary column.
- "1" indicates the presence of a category; "0" indicates absence.

Apply: One-Hot Encoding

ID	Profession
1	Nurse
2	Doctor
3	Actor
4	Doctor
5	Nurse
6	Nurse

Index	Profession	Nurse	Doctor	Actor	Teacher
1	Nurse	1	0	0	0
2	Doctor	0	1	0	0
3	Actor	0	0	1	0
4	Teacher	0	0	0	1
5	Nurse	1	0	0	0
6	Nurse	1	0	0	0

More Examples

Color
Red
Green
Blue

One-hot encoding →

d1	d2	d3
1	0	0
0	1	0
0	0	1

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

How to implement One-Hot Encoding

```python
import pandas as pd

# Create a DataFrame with categorical data
data = {'Profession': ['Nurse', 'Doctor', 'Actor', 'Engineer']}
df = pd.DataFrame(data)

# Perform one hot encoding
df_encoded = pd.get_dummies(df, columns=['Profession'])

# Display the encoded DataFrame
print(df_encoded)
```

```python
from sklearn.preprocessing import OneHotEncoder
import numpy as np

# Create a DataFrame with categorical data
data = {'Profession': ['Nurse', 'Doctor', 'Actor', 'Engineer']}
df = pd.DataFrame(data)

# Initialize OneHotEncoder
encoder = OneHotEncoder()

# Fit and transform the data
encoded_data = encoder.fit_transform(df[['Profession']])

# Convert the encoded data to a DataFrame
df_encoded = pd.DataFrame(encoded_data.toarray(), columns=encoder.get_fea

# Display the encoded DataFrame
print(df_encoded)
```

One Hot Encoding: Pandas Example

```
s = pd.Series(list('abca'))
pd.get_dummies(s)
```

→ perform one-hot encoding on the categorical data in the Series

	a	b	c
0	True	False	False
1	False	True	False
2	False	False	True
3	True	False	False

One Hot Encoding

Upsides:

- You get to use your categorical variables

Downsides:

- If there are 50 categories, your dataset explodes.