(Important)

Column Transformer: In a dataframe while doing our feature engineering we have to encode our categorical features but we will not apply the same encoding for each feature. ~~For~~ Like

For some column → Ordinal encoding

For some column → OHE

For target Column → Label encoding

So, if we want to do this for the features, after we apply encoding method to a dataframe column, it gives us back a ~~n~~ numpy array which we have to again append to the dataframe. We have to keep doing it for all the categorical columns, which ~~a~~ is a hectic job to do.

To resolve that issue "Column Transformer" concept came. We can ~~do~~ apply the encoding techniques using column transformer where we can do the work for all the categorical features at a time.

It is a scikit learn ~~p~~ module which helps us to make our work easier. Not only encoding, we can handle missing values together with doing encoding with the columns with this technique.

Lab instruction:

- Import librairies
- Import Simple Imputer, Ordinal Encoding, One hot encoding, Label encoding
- Use covid toy dataset
- Train, test, split
- Now try all the imputation and encoding techniques manually
  - Gender, city (Nominal Category) → One hot encoding
  - Cough (Ordinal Category) → Ordinal Encoding
  - fever (Missing values) → Simple imputer
  - has_covid (target column) → label encoders.
- Now do that with Column Transformer.

→ Code has been uploaded to github.

Scikit-learn Pipelines: (Important technique)

Pipelines chains together multiple ~~set~~ steps so that the output of each step is used as input to the next step.

Pipelines makes it easy to apply the same preprocessing to train and test.

What's the benefit?

When you deploy your model to the server, and when new dataset comes as input, then you have to do the same preprocessing steps again for the new data. So, if you don't create a pipeline, you have to make the preprocessing steps there also, which is a bit hectic.

Function Transformer: It's another technique in Feature Engineering which is to apply mathematical formulas to your columns and transform them.

Types →

1) Log transform
2) Reciprocal Transform
3) Power Transform
   - square
   - sqrt
4) Box Cox Transform
5) Yeo-Johnson Transform

CamScanner

These transformers helps your data distribution of a column to be converted into "normal distribution"

Statistical ML Algorithms like (Linear, Logistic) Regression want their data to behave like normal distribution. So, in those case, these function Transformer techniques are very important.

Some other ml algorithms like Decision Tree or Random Forest: They don't require it.

Scikit learn provides 3 types of transformer:

Sklearn

- Function Transformer
  - log transformer
  - Reciprocal Transformer
  - Sq, sqrt Transformer
  - Custom Transformer
- Power Transformer
  - Box cox
  - Yeo Johnson
- Quantile Transformer

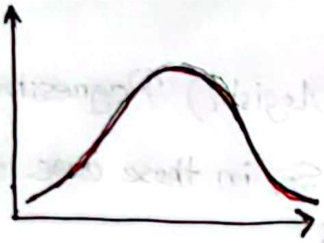
How to find my data is normally distributed or not?

→ Use sns.distplot()

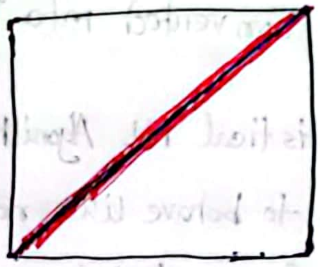
→ Use pd.skew()

→ Make QQ plot

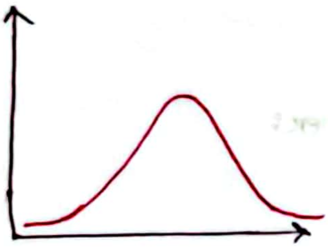
Normally Distribute data



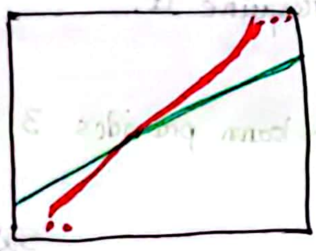
QQ plot



Data too picke in the middle

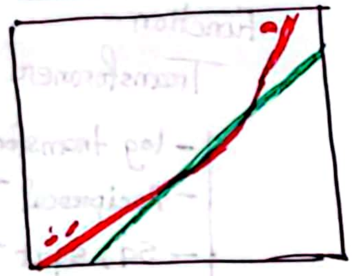


QQ plot

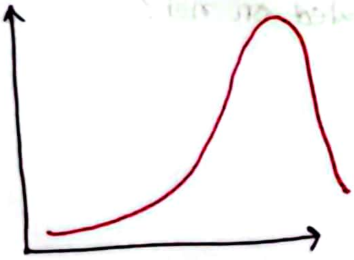


Skewed Data (Right)

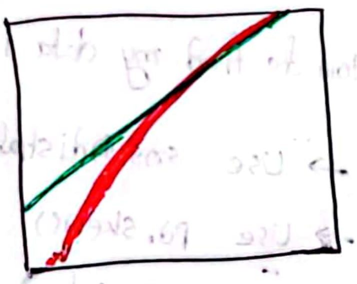
QQ Plot



Skewed Data (Left)

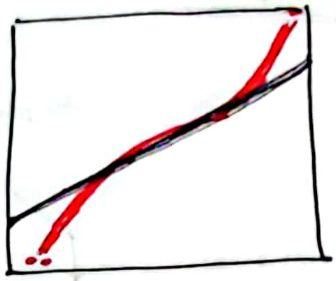


QQ Plot



Fat tails (Data)

QQ Plot



Log transform:

In log transform, we just transform the column values with their log values.

When to use?

- If your data has no negative value
- If your column data is Right skewed.

Other Transforms

Reciprocal ($1/x$)	Square (x^2)	Sqrt ($\sqrt{x}$)
You can try it to see if getting better results	→ Left ~~ske~~ skewed data	You can try it and see if getting better results.

Power Transformer:

Power Transformer

→ Box-Cox Transform

→ Yeo - Johnson Transform

Box - Cox Transform:

Formula → $$x_i^{(\lambda)} = \begin{cases} \dfrac{x_i^{\lambda} - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \ln(x_i) & \text{if } \lambda = 0 \end{cases}$$

The exponent here is a variable called (lambda) (λ) that varies over the range of −5 to 5, and in the process of searching, we examine all values of λ. Finally we choose the optimal value (resulting in the best approximation to a normal distribution) for your variable.

- This will work on the column where values > 0.

This issue solves by the next transform - Yeo-Johnson Transform

Yeo - Johnson Transform:

$$x_i^{(\lambda)} = \begin{cases} [(x_i+1)^{\lambda} - 1]/\lambda & \text{if } \lambda \neq 0,\ x_i \geq 0 \\ \ln(x_i) + 1 & \text{if } \lambda = 0,\ x_i \geq 0 \\ -[(-x_i+1)^{2-\lambda} - 1]/(2-\lambda) & \text{if } \lambda \neq 2,\ x_i < 0 \\ -\ln(-x_i+1) & \text{if } \lambda = 2,\ x_i < 0 \end{cases}$$

This transformation is somewhat of an adjustment of the Box-cox transformation, by which we can apply it to negative numbers.

For algorithms who needs their data to be ~~no~~ normally distributed, we can use both of the technique and check their accuracy and the choose the best one.