→ Column Transformer helps automate this without doing manual preprocessing for each column seperately.

⇒ Eg:-

Age	Salary	City
25	40000	Delhi
30	60000	Mumbai.

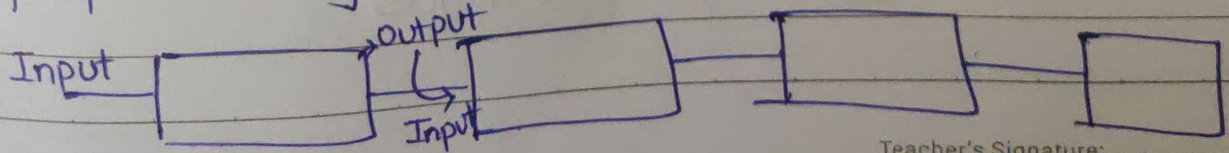
- Age & Salary → Scale (numerical transformation
- City → One-Hot Encode (categorical transformati

→ Column Transformer ek ~~sk~~ scikit learn tool che je apne dataset ma different type na columns par alag alag preprocessing ek sath apply karva de che, jaise numerical columns par scaling ane categorical encoding, etle mixed data types hoi to pan apde easily ek pipeline ma preproce kari sakye without ~~s~~ writing seperate code for each column.

* ML Pipelines :- (Day-29)

"Pipelines chain together multiple 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."



→ There are two functions pipeline & make-pipeline. But we usually use make-pipeline function. bcz we don't need to pass the extra parameters

→ "A Pipeline in scikit-learn is a tool that combines multiple data preprocessing steps and a ML model into a single ~~flow~~ workflow. It allows all steps - like scaling, encoding, and model training - to be executed together automatically. This makes the code cleaner, easier to manage, and helps ensure that the same transformations are applied consistently during both training and prediction."

→ Pipeline ek scikit-learn tool che je multiple data preprocessing steps ane ML model ne ek single workflow ma combine kare che. Aa thi apde pura process - like scaling, encoding, and model prediction sudhi na steps automatically sequence ma execute kare che

⇒ Points to be noted :-

(1) Order matters - The steps in the pipeline are executed in the same order you define them, so always put preprocessing steps before the model.

(2) Last step must be an estimator - The final step should be a model that has .fit() and .predict() methods.

(3) Intermediate steps should be transformers - All steps before the last one must have both .fit() and .transform () methods (like StandardScaler(), etc.

(4) Consistency - The same transformations are automatically applied to both training and test data, reducing the chances of data leakage.

(5) Parameter tuning - You can perform hyperparamete tuning directly on the entire pipelin, not just the model.

(6) Clean and resuable code - Pipelines make your code easier to maintain and ~~res~~ reuse for other datasets.

(7) Avoid data leakage - Since transformations are fitted only on the training set inside the pipelines, your model won't get biased from the test data.

Teacher's Signature: