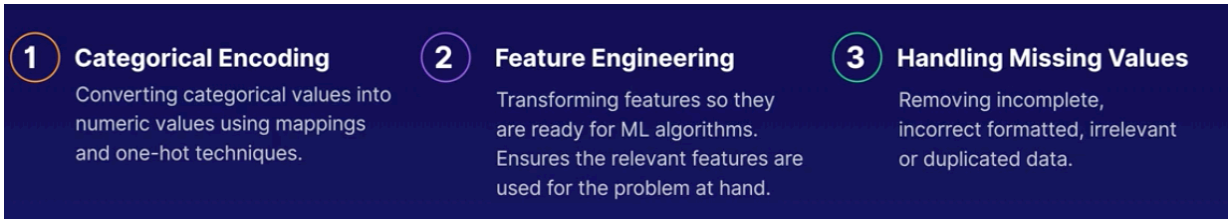
CloudGuru - 3 - Data Preparation

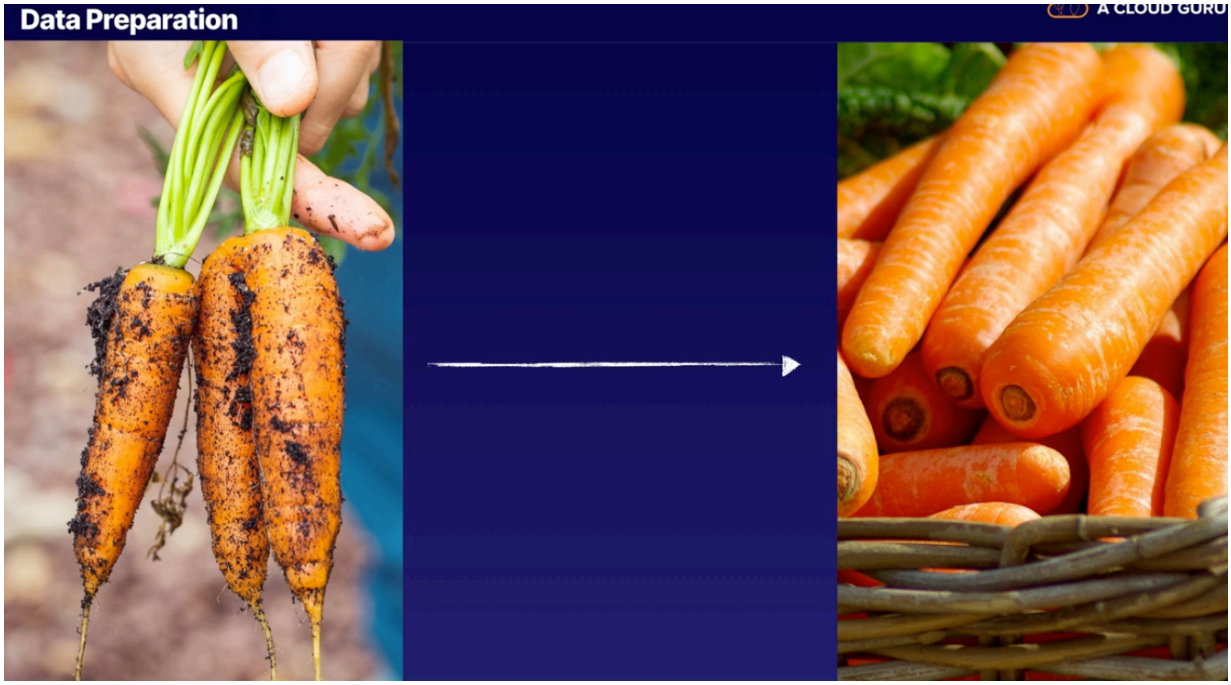
https://acloud.guru/course/aws-certified-machine-learning-specialty/learn/f399041c-e040-9d9a-d60c-35317e0e9a97/844cc92f-b2b7-dcb4-e7b8-5e38306d3205/watch?backUrl=~2Fcourses

Data preparation is the process of transforming a dataset using different techniques to prepare it for model training and testing.
Data preparation is usually the most time consuming step of ML

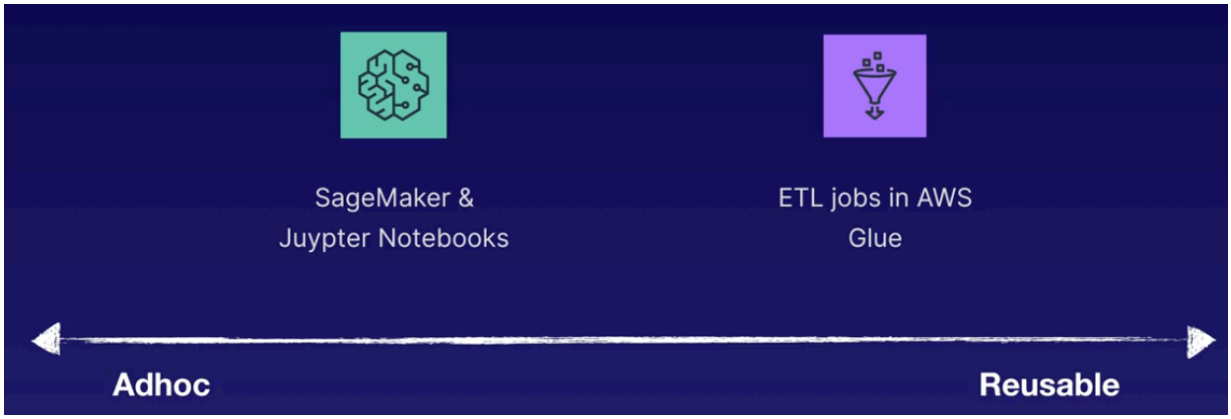


Typical data preparation example:
- Formatting: some values may not be standard like “No” instead of a 0/1 integer
- Missing values
- Duplicates - they can skew results (remove them)
- Invalid Data
- Encoding - when an ML algorithm does not accept text based values or categorical values => need to encode into 0 or 1 (or numerical value)

In summary, data preparation is to take unprepared, dirty data and transform it to something useful, that we can consume in our model.



There are a few tools in AWS that can help with data transformation:
- SageMaker (and Jupyter notebooks)
- AWS Glue (that is an ETL tool)

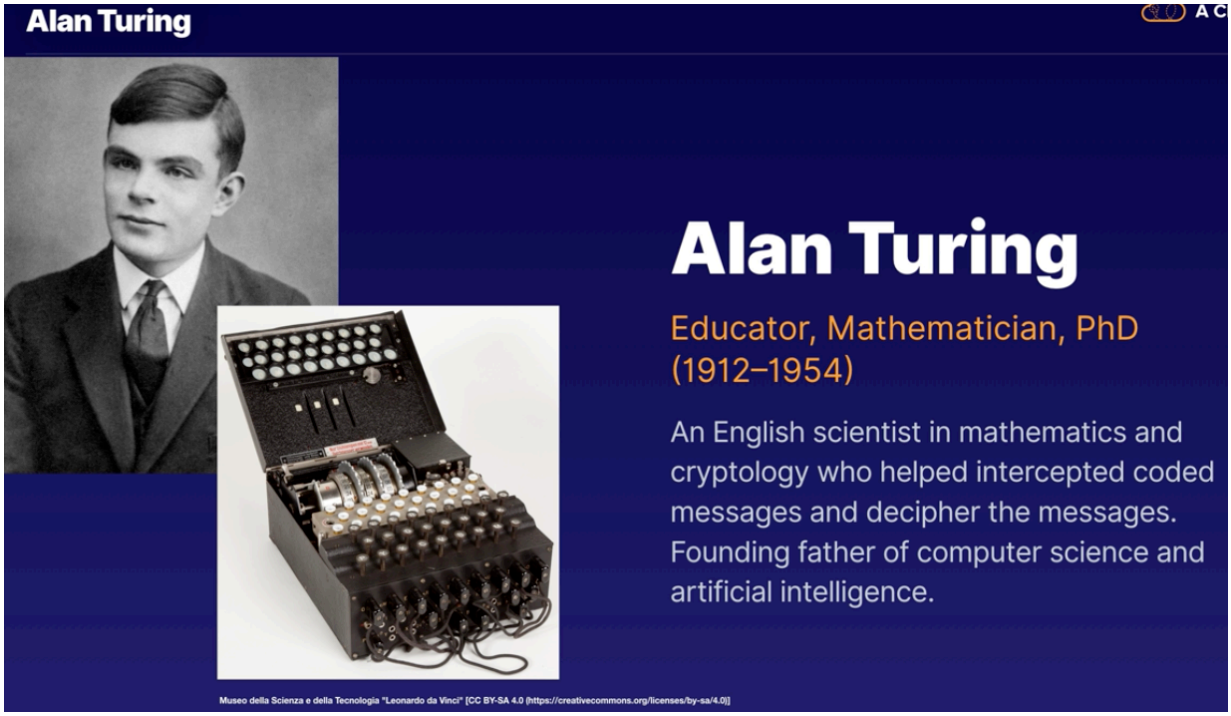


Categorical Encoding

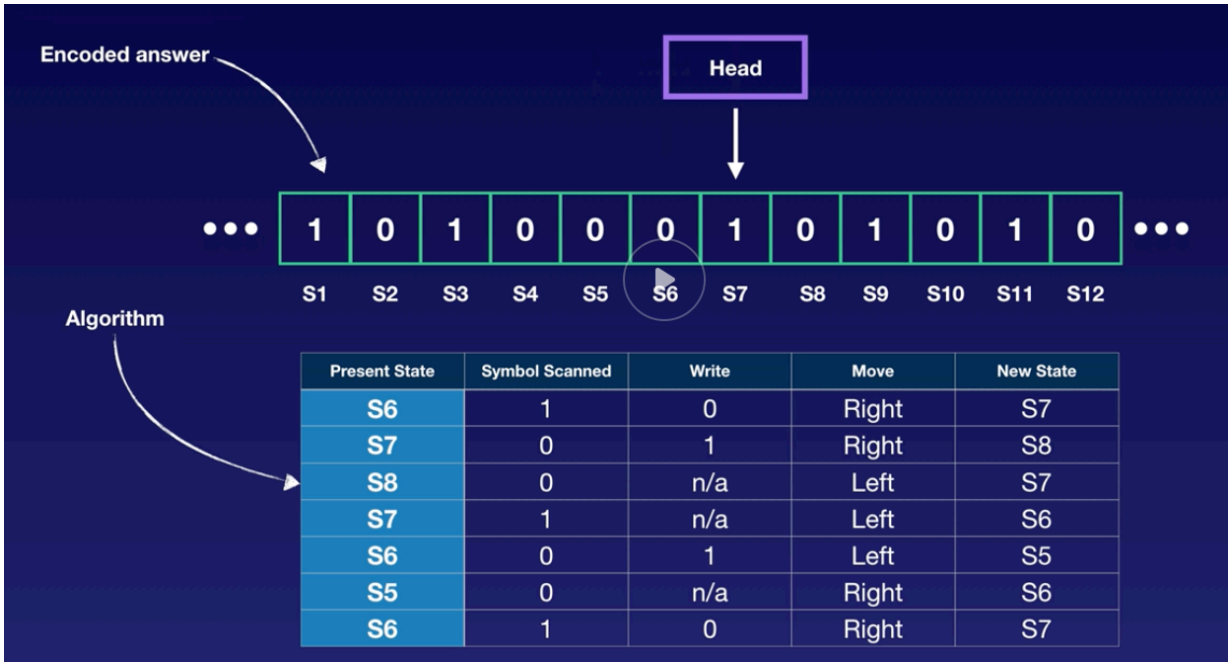
Categorical Encoding is the process of manipulation categorical variables when ML algorithms expect numerical values as inputs
=> it's about changing category values in our datasets to numbers.

categorical variable = categorical feature = discrete feature

Historical reference: Alan Turing - managed to decipher message on the “enigma” machine from the German



He proved any algorithm can be defined with a virtual tape encodings in 0 and 1
He is one of the founding fathers of CS and AI

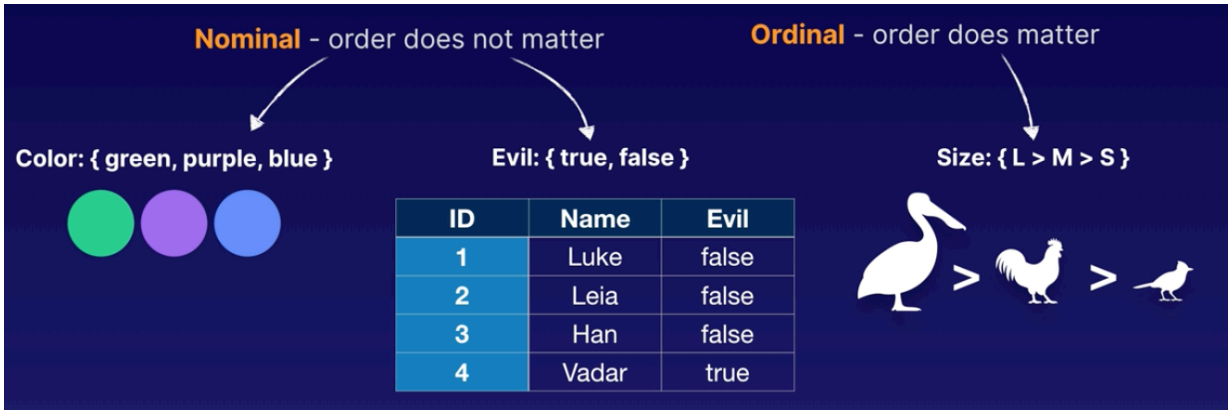


When is it a good time to encode?

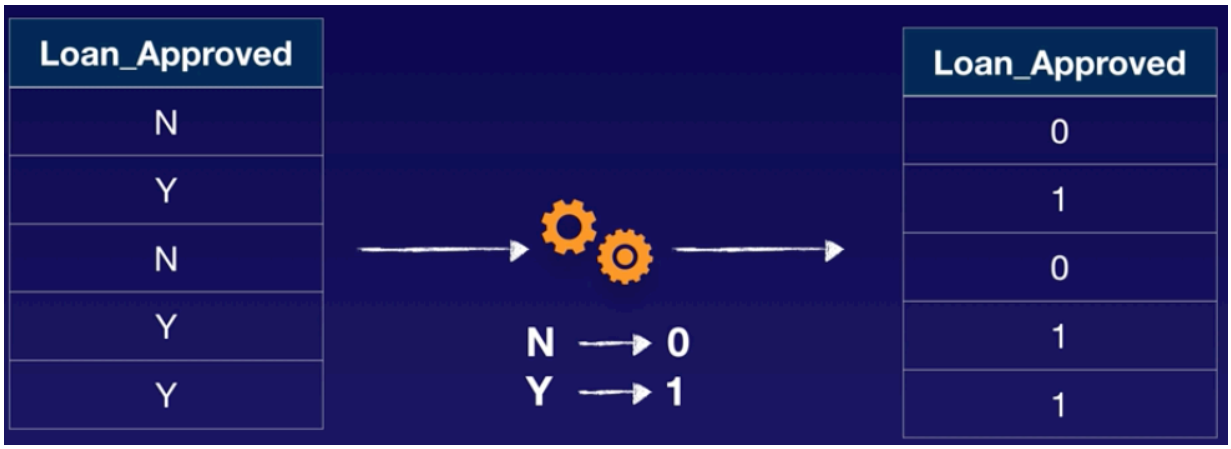
Problem	Algorithm	Encoding
Predicting the price of a home	Linear Regression	Encoding necessary
Determine wether given text is about sports or not	Naive Bayes	Encoding not necessary
Detecting malignancy in radiology images	Convolutional Neural Network	Encoding necessary

Nominal Vs Ordinal:

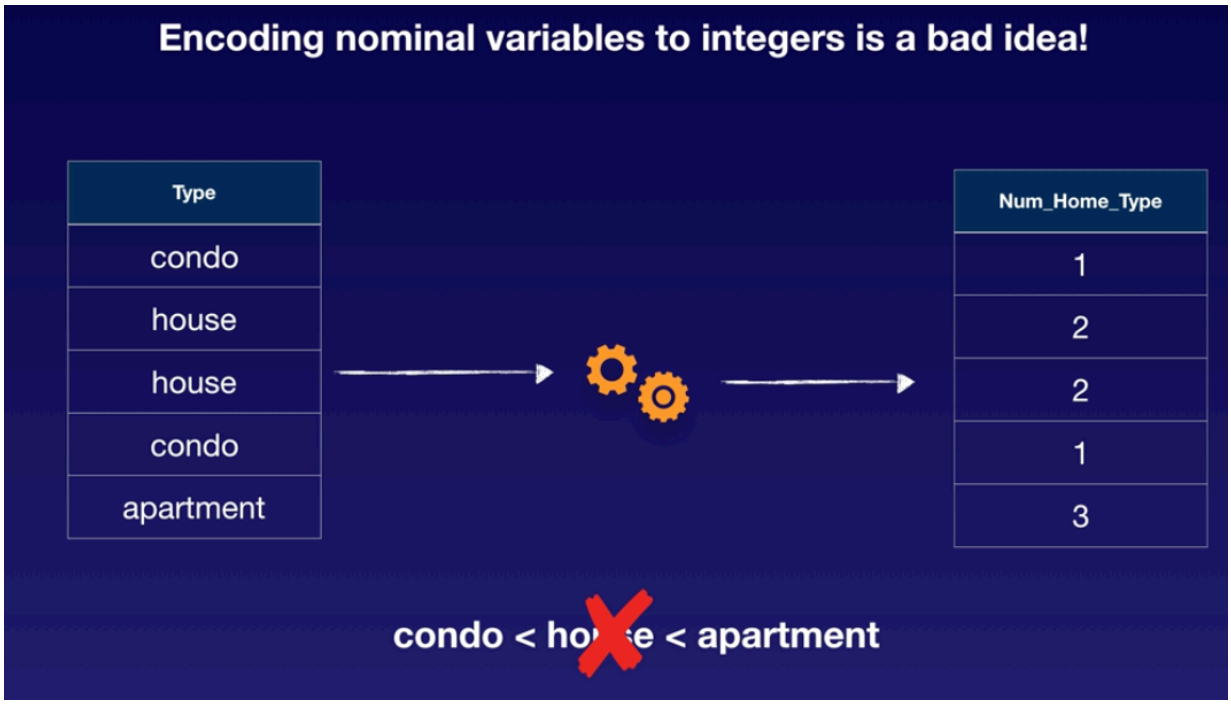
- Nominal: order does not natter
- Ordinal: order DOES matter



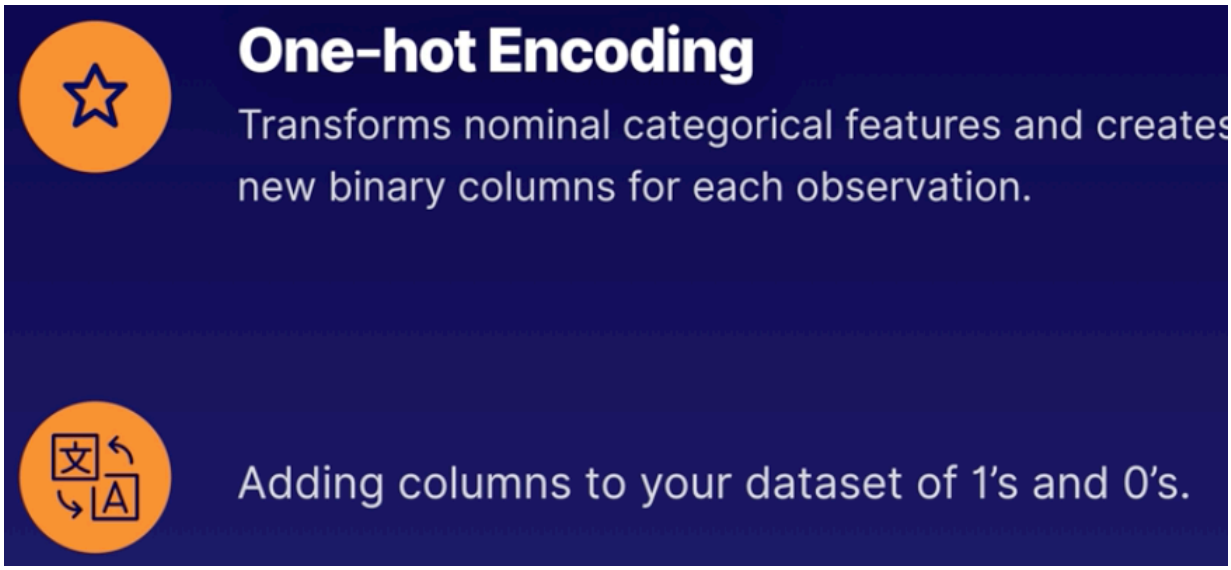
Encoding categorical features



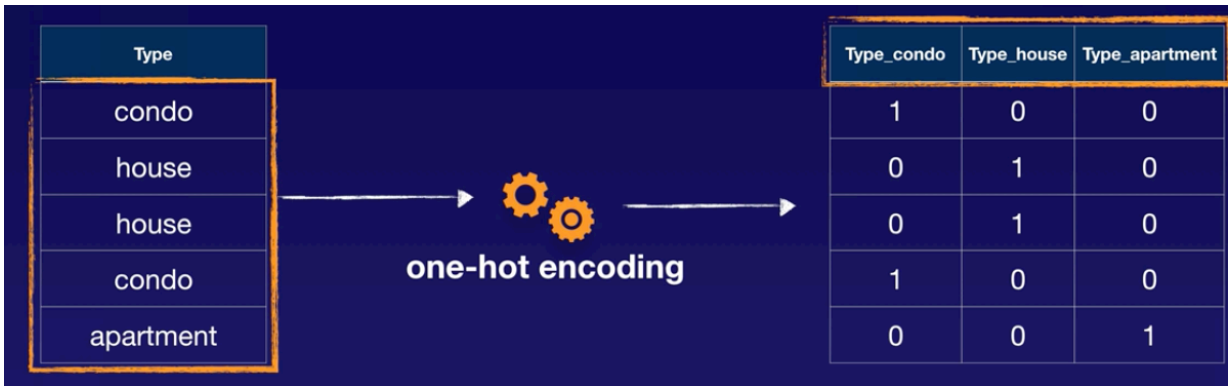
Encoding nominal values into integers is a bad idea
In the below example, a model could interpret the house incorrectly.

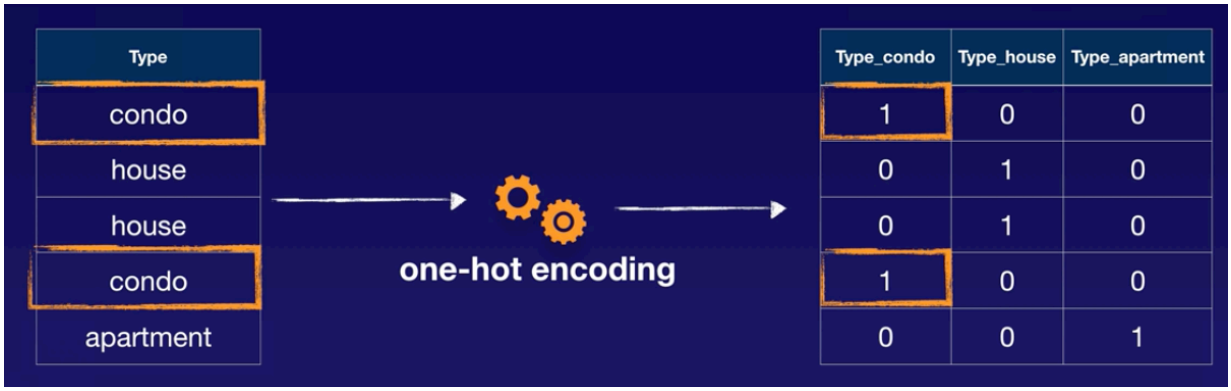


Use **One-hot encoding** instead
=> adds new features for each new category and assign a 0 if the observation does not include that category, or 1 if it does

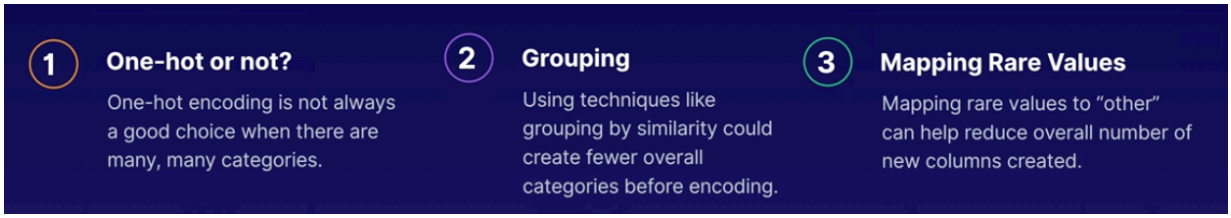


Here is the house type with One-Hot encoding





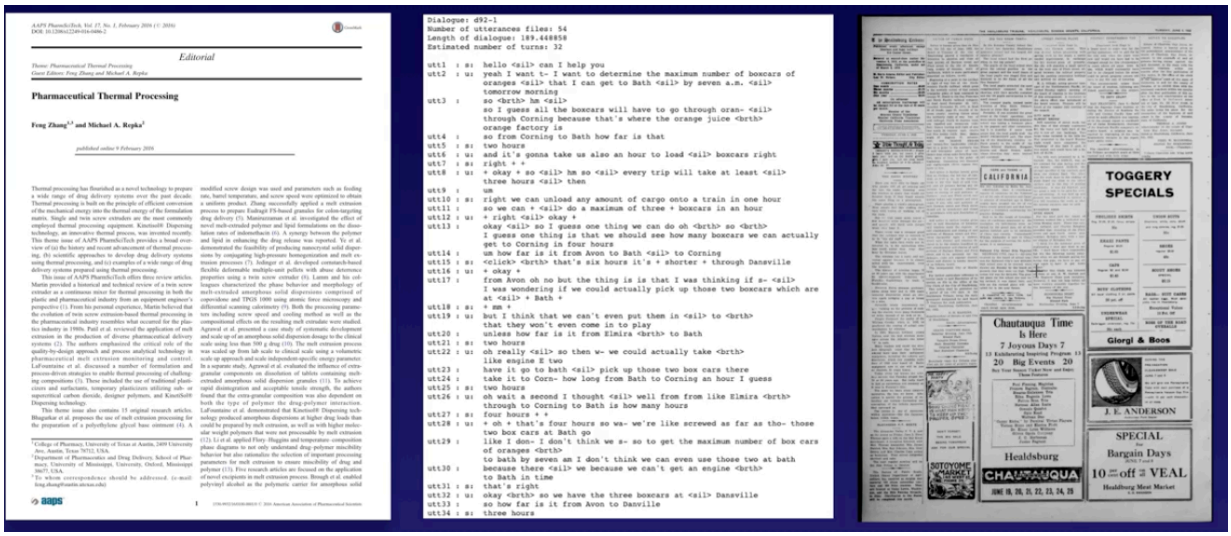
One-Hot encoding is not goof for many features though - it will grow exponentially
There are other techniques like **Grouping** or **Mapping rare values**



Text Feature Engineering

=> Transforming text within our data so ML Lagos can better analyze it.
In other words, it's about splitting text into bide size pieces

Examples of text data (or corpus data): white paper, dialog between 2 people, newspaper clipping



Editorial

Pharmaceutical Thermal Processing

Dialogue: d92-1
Number of utterances files: 54
Length of dialogue: 189.448858
Estimated number of turns: 32

utt1 : s: hello <sil> can I help you
utt2 : u: yeah I want t- I want to determine the maximum number of boxcars of oranges <sil> that I can get to Bath <sil> by seven a.m. <sil> tomorrow morning
utt3 : so <brth> hm <sil> so I guess all the boxcars will have to go through oran- <sil> through Corning because that's where the orange juice <brth> orange factory is
utt4 : so from Corning to Bath how far is that
utt5 : s: two hours
utt6 : u: and it's gonna take us also an hour to load <sil> boxcars right
utt7 : s: right + +
utt8 : u: + okay + so <sil> hm so <sil> every trip will take at least <sil> three hours <sil> then
utt9 : um
utt10 : s: right we can unload any amount of cargo onto a train in one hour
utt11 : so we can + <sil> do a maximum of three + boxcars in an hour
utt12 : u: + right <sil> okay +
utt13 : okay <sil> so I guess one thing we can do oh <brth> so <brth> I guess one thing is that we should see how many boxcars we can actually get to Corning in four hours
utt14 : um how far is it from Avon to Bath <sil> to Corning
utt15 : s: <click> <brth> that's six hours it's + shorter + through Dansville
utt16 : u: + okay +
utt17 : from Avon oh no but the thing is is that I was thinking if s- <sil> I was wondering if we could actually pick up those two boxcars which are at <sil> + Bath +
utt18 : s: + mm +
utt19 : u: but I think that we can't even put them in <sil> to <brth> that they won't even come in to play
utt20 : unless how far is it from Elmira <brth> to Bath
utt21 : s: two hours
utt22 : u: oh really <sil> so then w- we could actually take <brth> like engine E two
utt23 : have it go to bath <sil> pick up those two box cars there
utt24 : take it to Corn- how long from Bath to Corning an hour I guess
utt25 : s: two hours
utt26 : u: oh wait a second I thought <sil> well from from like Elmira <brth> through to Corning to Bath is how many hours
utt27 : s: four hours + +
utt28 : u: + oh + that's four hours so wa- we're like screwed as far as tho- those two box cars at Bath go
utt29 : like I don- I don't think we s- so to get the maximum number of box cars of oranges <brth> to bath by seven an I don't think we can even use those two at bath
utt30 : because there <sil> we because we can't get an engine <brth> to Bath in time
utt31 : s: that's right
utt32 : u: okay <brth> so we have the three boxcars at <sil> Dansville
utt33 : so how far is it from Avon to Danville
utt34 : s: three hours

TOGGERY SPECIALS

Chautauqua Time Is Here

7 Joyous Days 7
12 Enlightening Inspiring Program 13
20 Big Events 20

Healdsburg

CHAUTAUQUA

Giorgi & Boos

J. E. ANDERSON

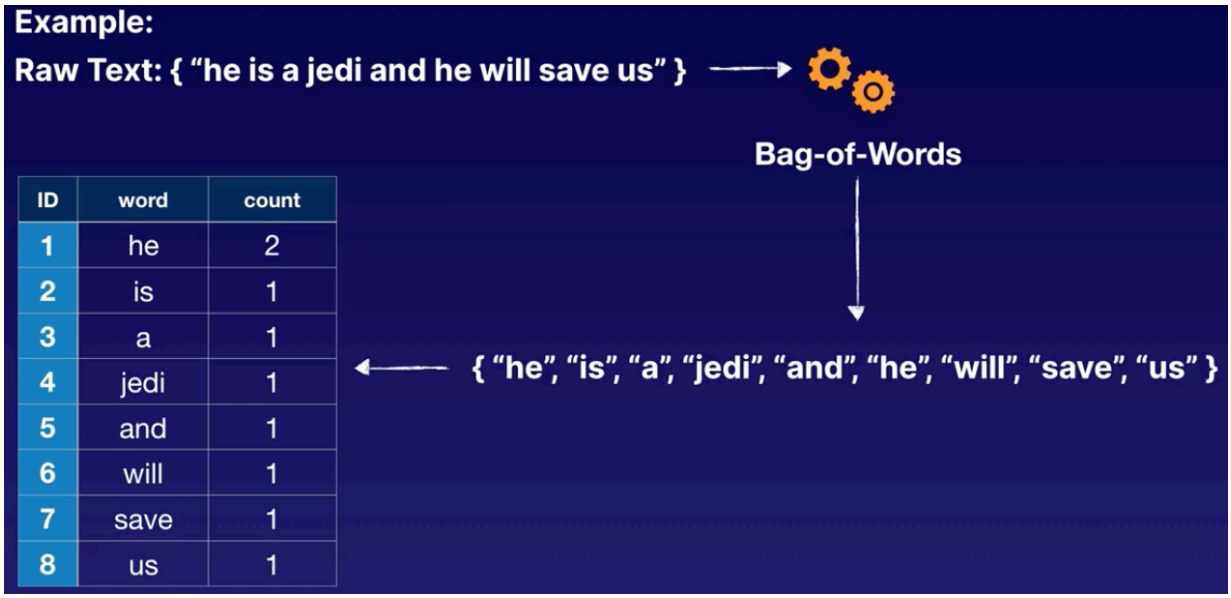
SPECIAL
Bargain Days
10 off 2 VEAL

Healdsburg Meat Market

Other example: street address extraction from a long set string

Bag-ofWords: Tokenizes raw text and creates statistical representation of the text.
In other words it breaks up text by white space into single words.
=> It’s the simple method to break down text into single tokens.

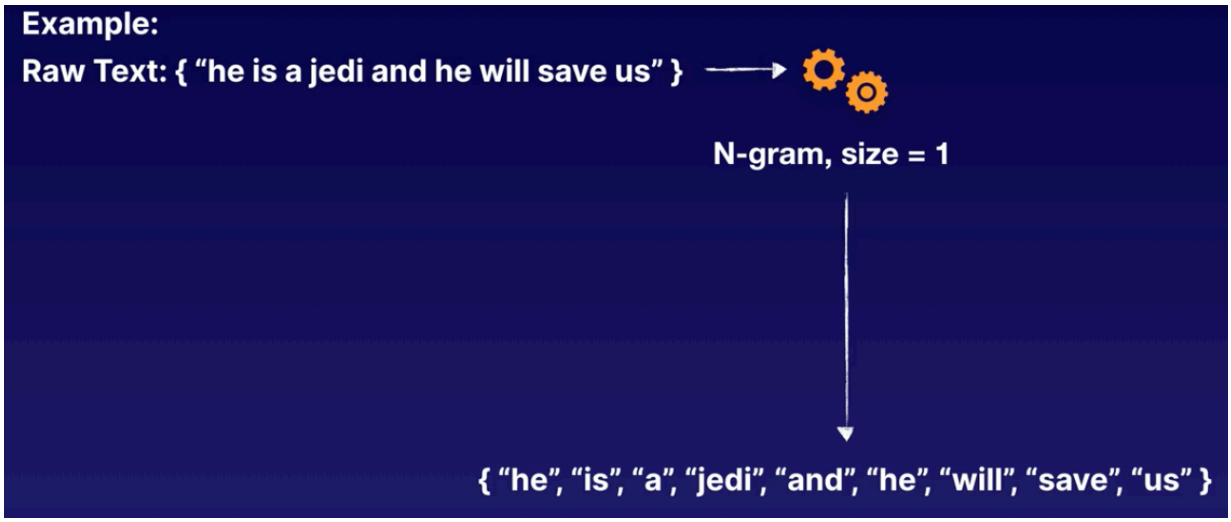
From this we can create word counts and see the different distribution of hose words are produced/used.
Example:



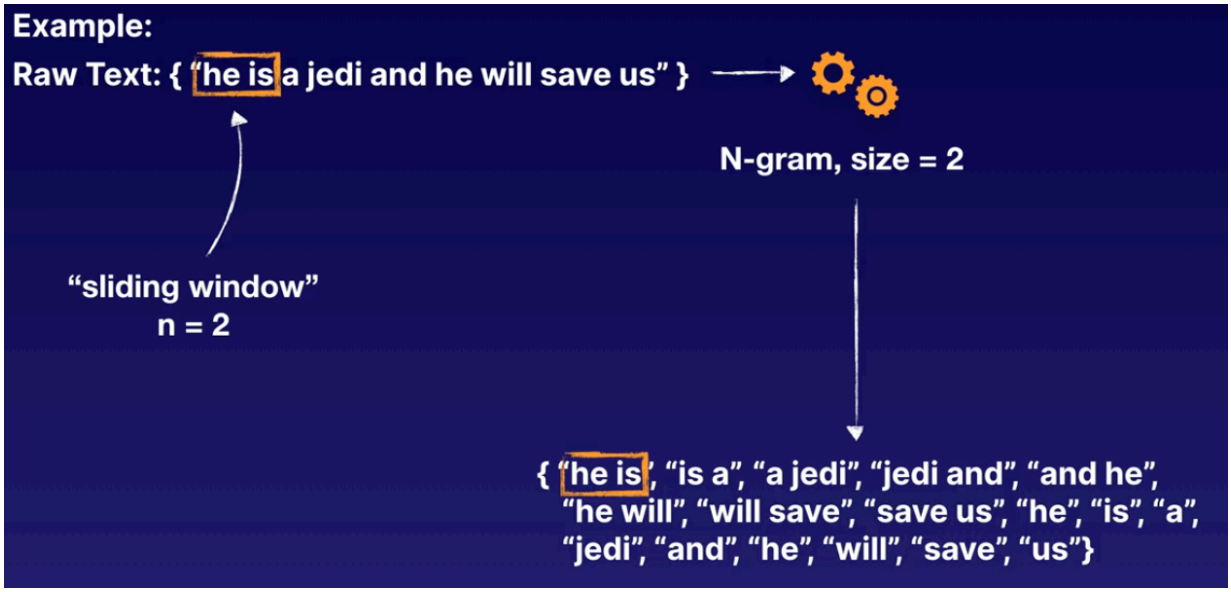
ID	word	count
1	he	2
2	is	1
3	a	1
4	jedi	1
5	and	1
6	will	1
7	save	1
8	us	1

N-Gram: builds of Bag-of-Words and produces groups of words of n size
In other words it breaks up text by white space into groups of words.

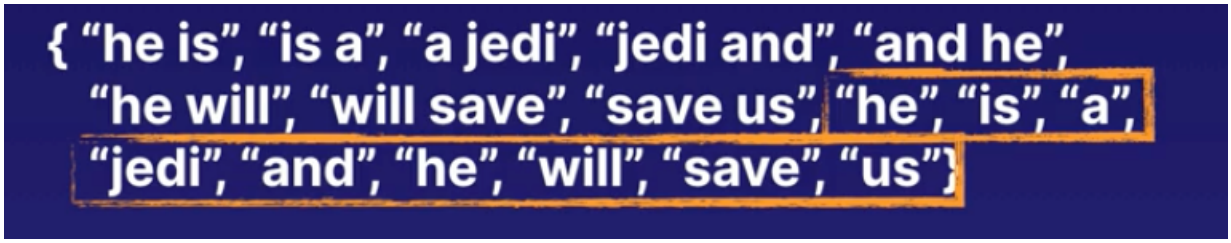
Example
N-gram, size = 1



N-gram, size = 2
We use a window of 2 words and slides the window to tokenize all the pairs of 2 words.
To note it can also



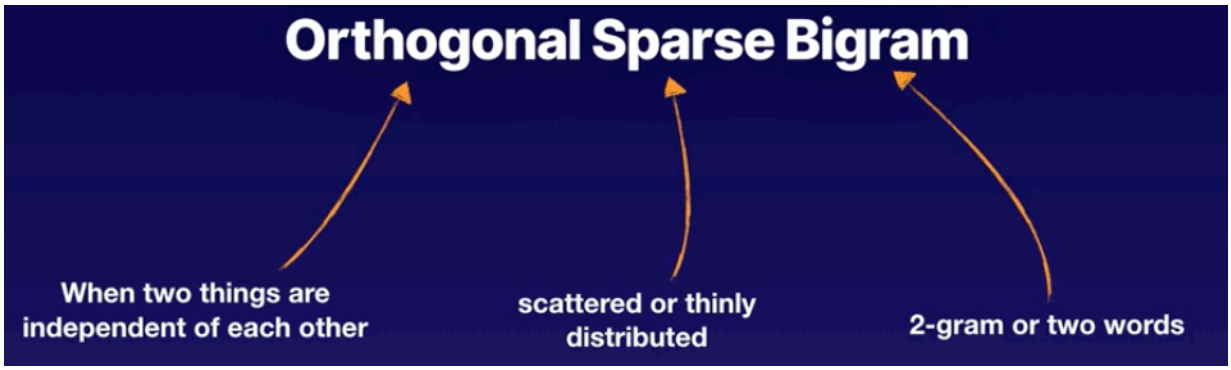
To note it can also create n-grams of lower size (size = 1)



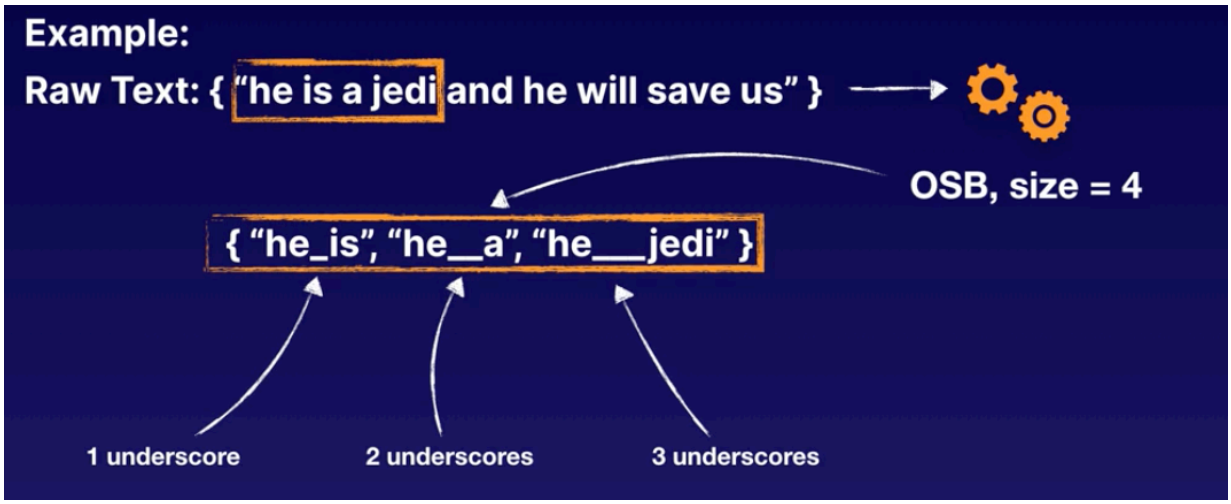
When is that useful?
Spam filter to find pairs filter like "you win", "click here"

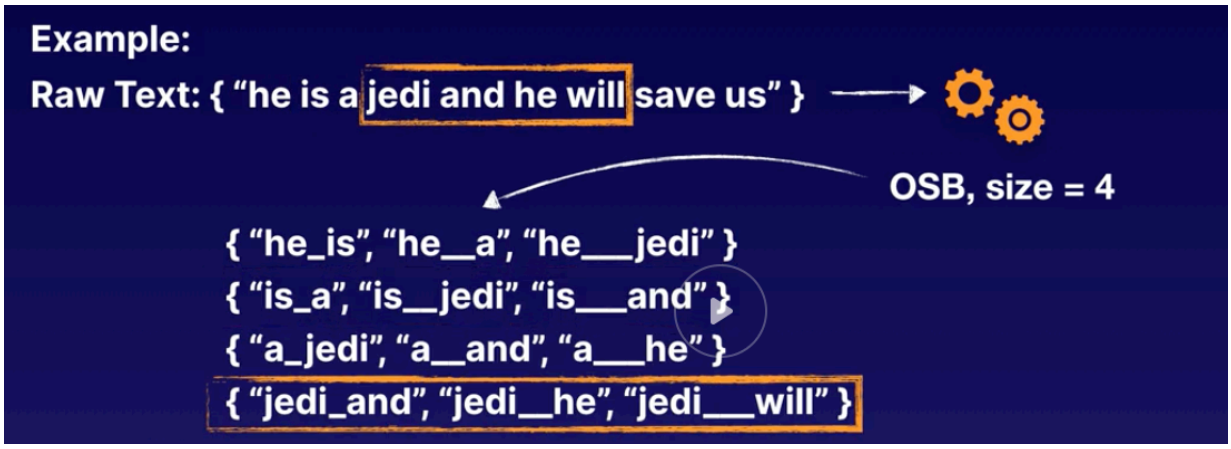
Unigram = 1 word token
Bigram = 2 word tokens
Trigram = 3 word tokens

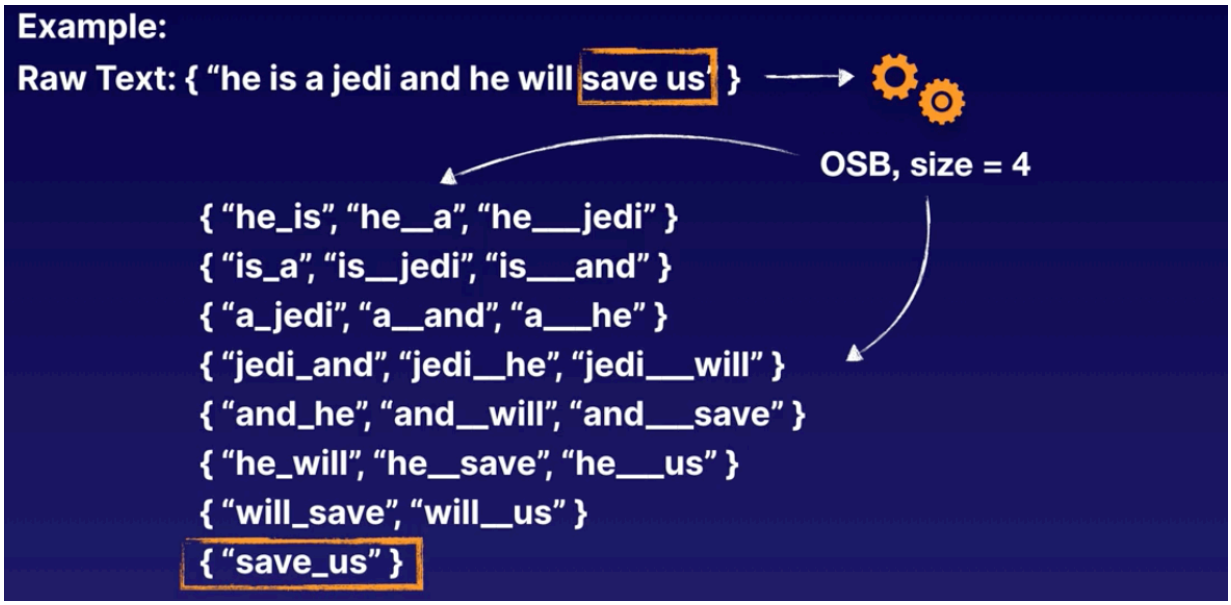
Orthogonal Sparse Bigram (OSB) - similar to n-gram
Creates groups of words of size n and outputs every pair of words that includes the first word.
In other words, it creates groups of 2 words that always include the first word.



Example:
OSB with size = 4
Uses “_” where there is a word or white space

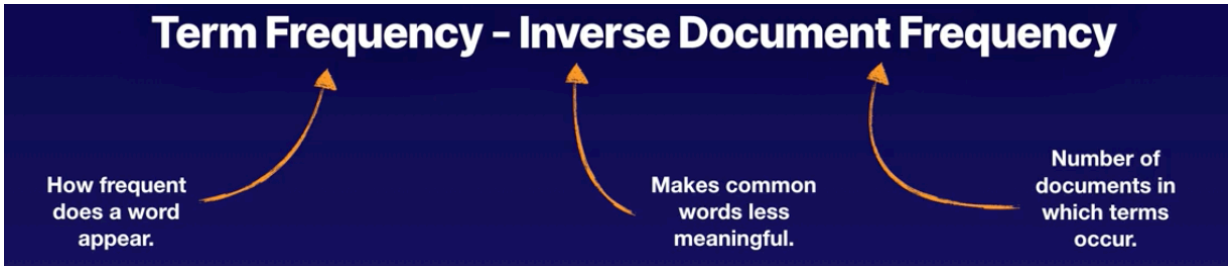




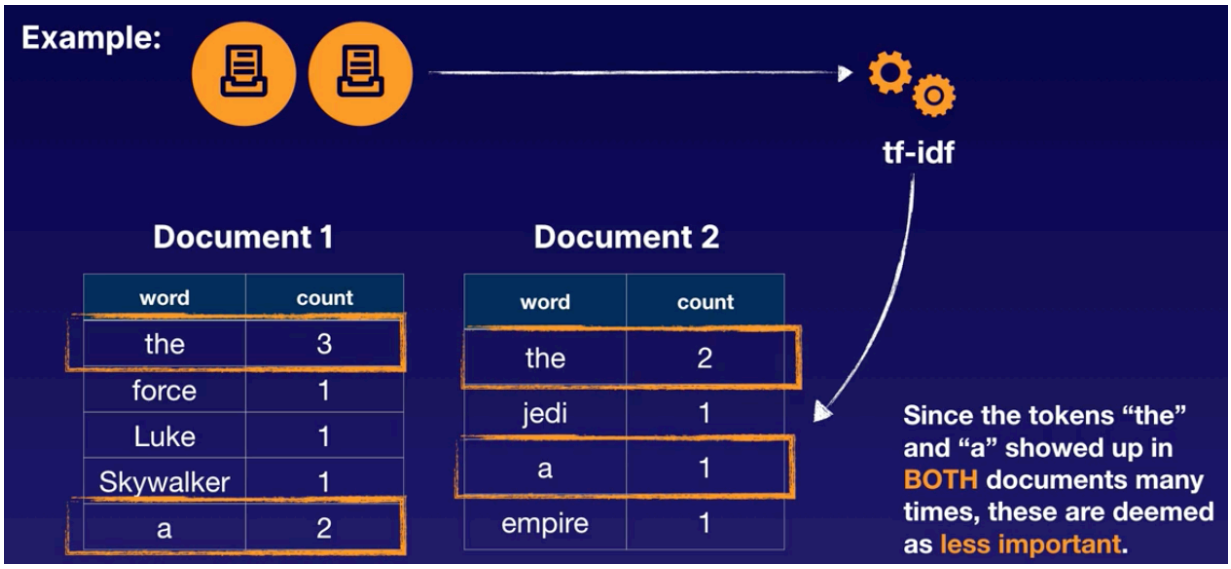


N-grams and OSB are good way to tokenize a text.
But sometimes we need to find importance of words and maybe reduce importance of common words too like next technique.

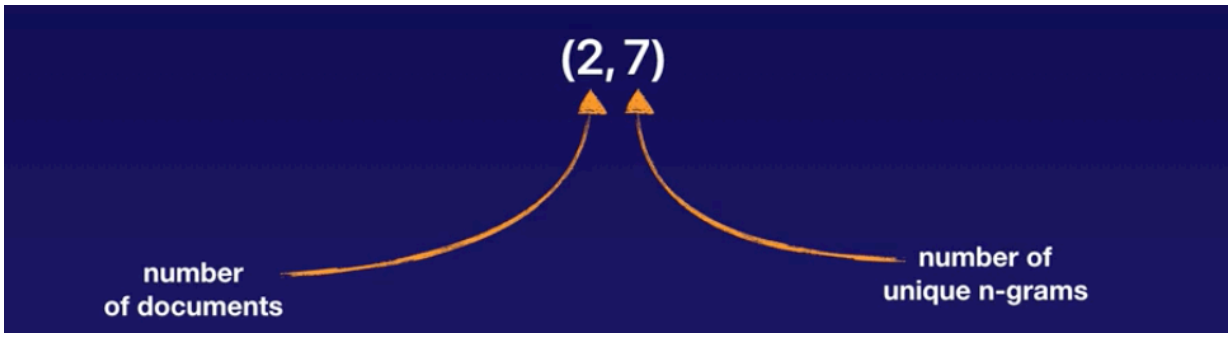
TF-IDF: Term Frequency - Inverse Document Frequency
Represents how important a word or words are to a given set of text by providing appropriate weights to terms that are common and less common in the text.
In other words, it shows us the popularity or a word or words in text data by making common words like “the” or “and” less important.

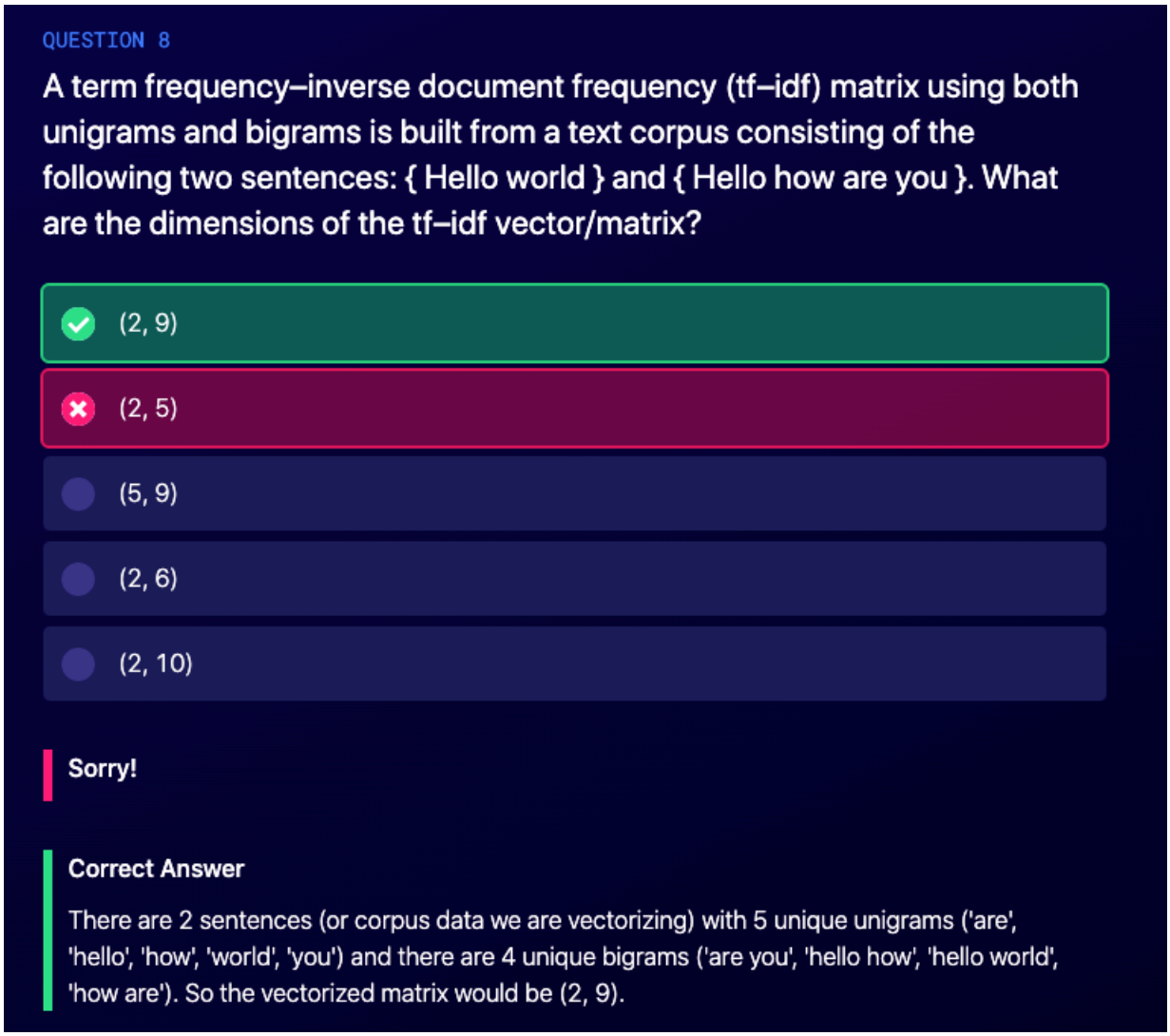


Example:



In exam, we can ask to vectorize a document with TF-IDF
It is show as (number of documents, number of unique n-grams)
Example:





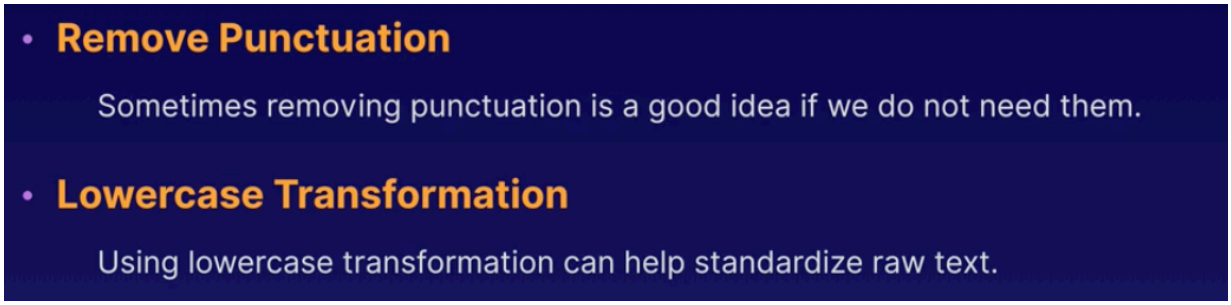
What to use when?

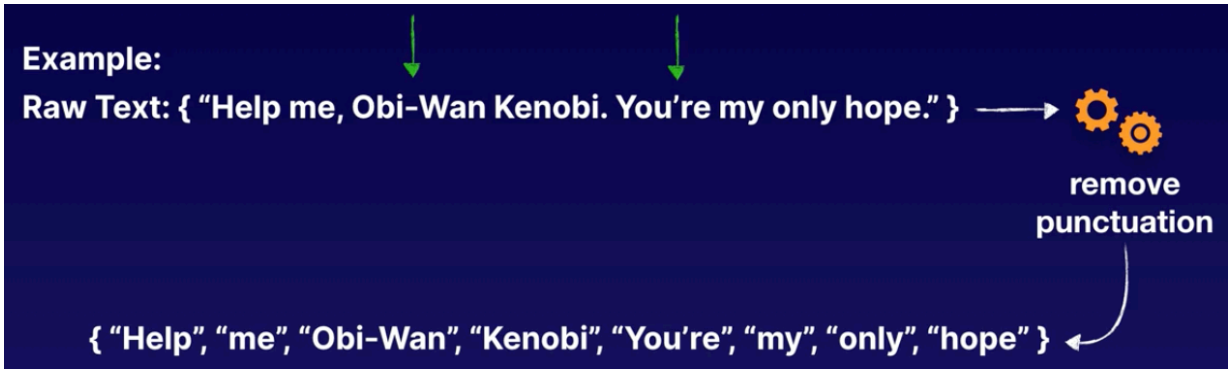
Problem	Transformation	Why
Matching phrases in spam emails	N-Gram	Easier to compare whole phrases like “click here now” or “you’re a winner”.
Determining the subject matter of multiple PDFs	Tf-idf Orthogonal Sparse Bigram	Filter less important words in PDFs. Find common word combinations repeated throughout PDFs.

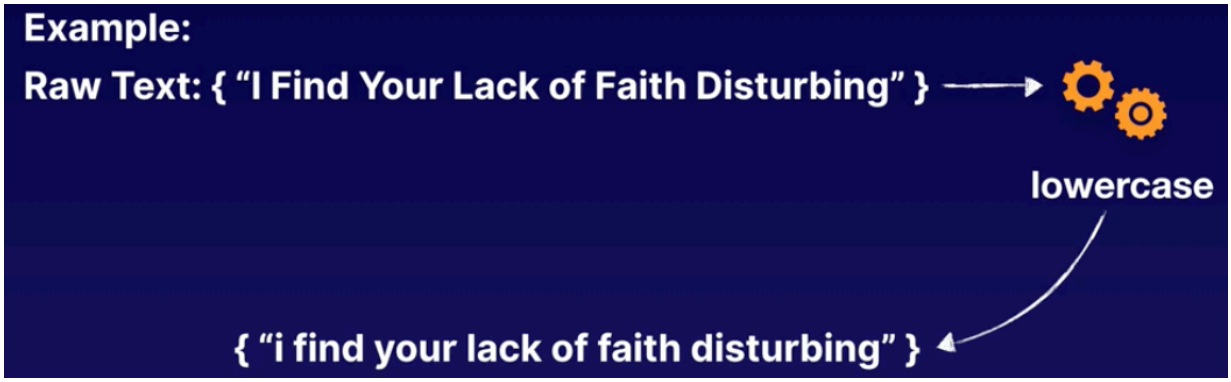
Remember that:

- N-gram can be used to math whole words or phrases
- TF-IDF helps us determine which words are more important in document than others, and help us filter out common words like “and” or “the”
- OSB - find common words combination or bi-grams

Before using these Lagos, it is probably a good thing to remove punctuation and use lower casing



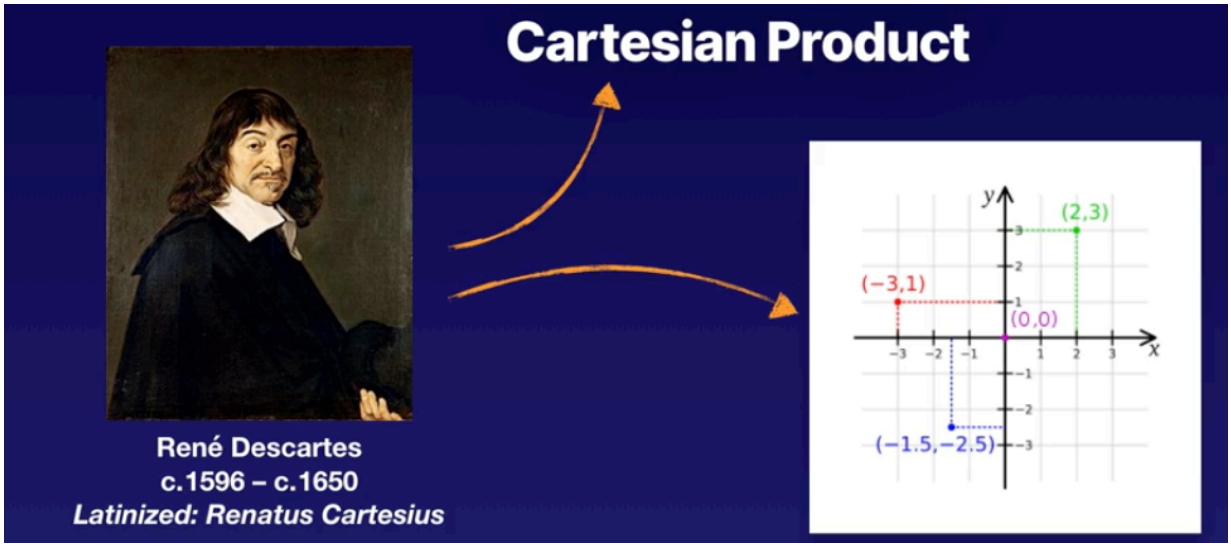




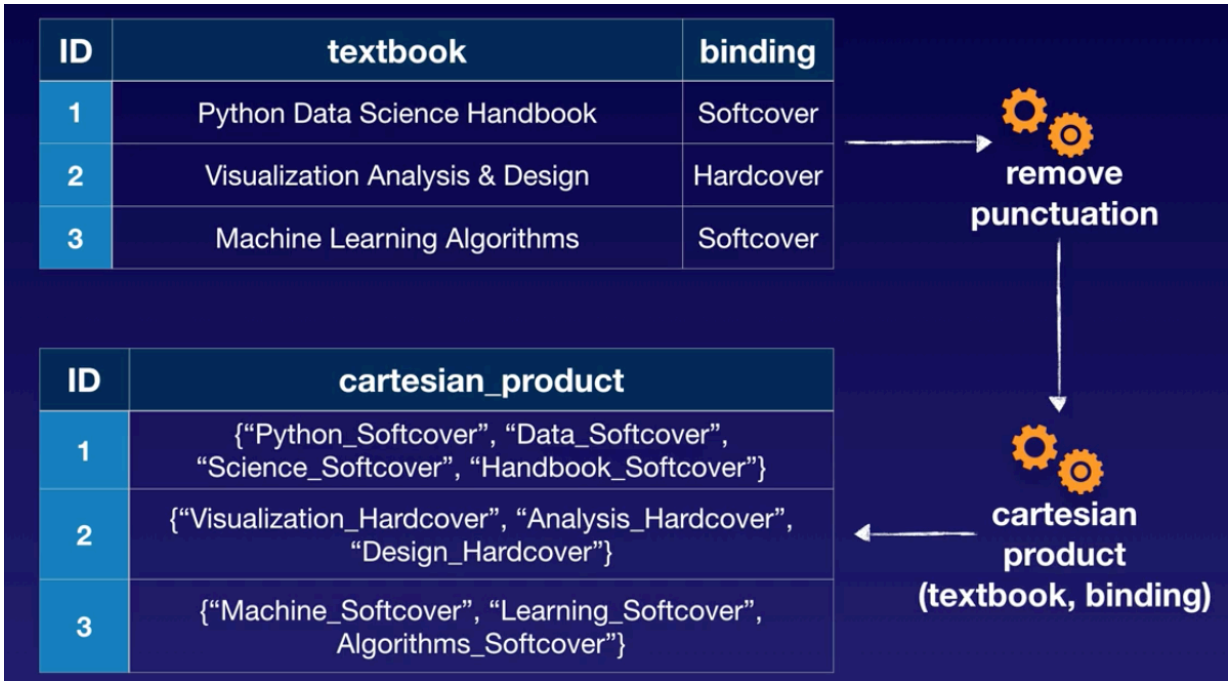
Cartesian product Transformation - creates a new feature from the combination

of two or more text or categorical values
In other words, it combines sets of words together

Cartesian product comes from Rene Descartes.

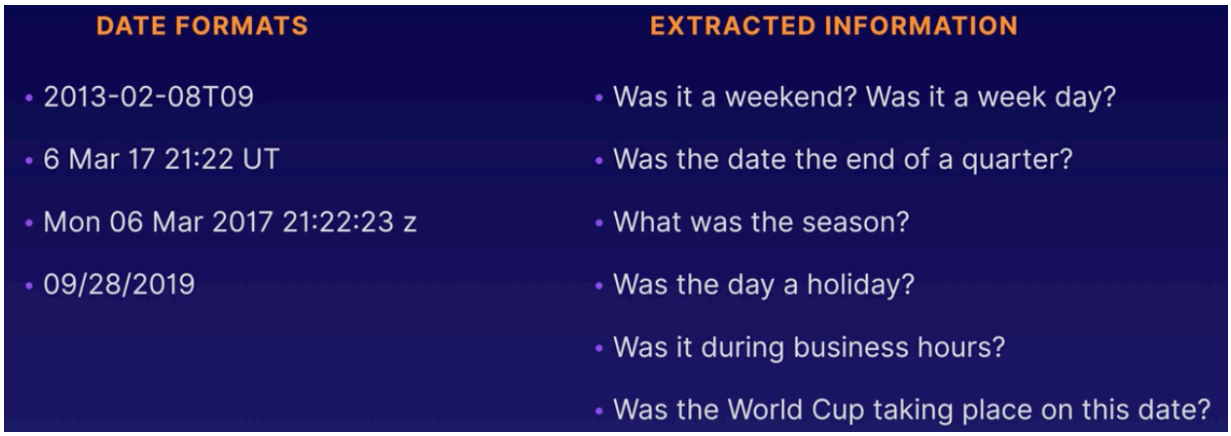


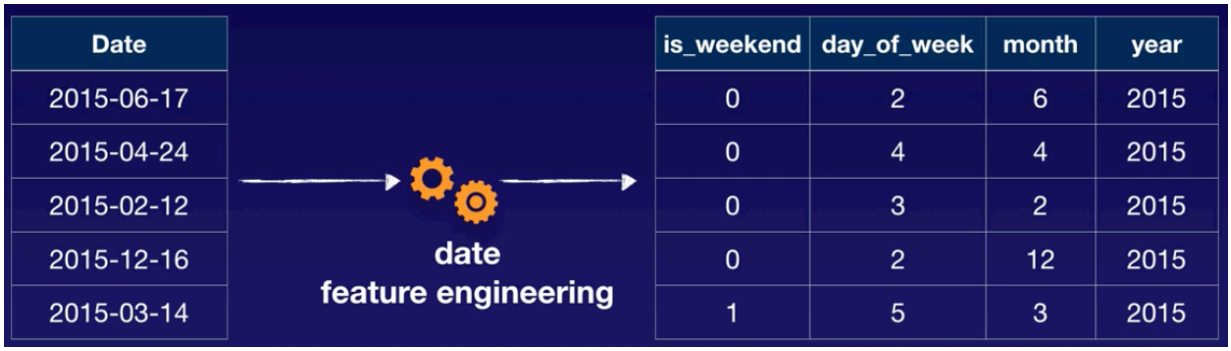
Example of Cartesian product Example, using "_" to concatenate the words



This can be used to make recommendations

Feature Engineering Dates.
Dates come in many format - need to translate them into useful info





With new features, we can now determine more important correlation or info about our observations.

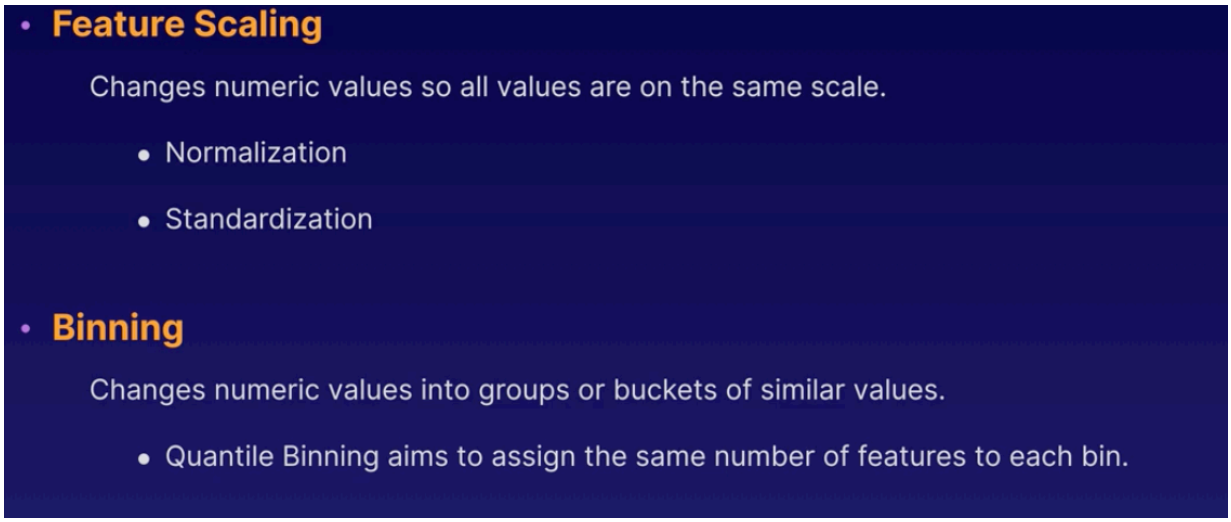
Recap of Text Feature Engineering:

Technique	Function
N-Gram	Splits text by whitespace with window size *n*
Orthogonal Sparse Bigram (OSB)	Splits text by keeping first word and uses delimiter with remaining whitespaces between second word with window size *n*
Term Frequency - Inverse Document Frequency (tf-idf)	Helps us determine how important words are within multiple documents
Removing Punctuation	Removes punctuation from text
Lowercase	Lowercase all the words in the text
Cartesian Product	Combines words together to create new feature
Feature Engineering Dates	Extracts information from dates and creates new features

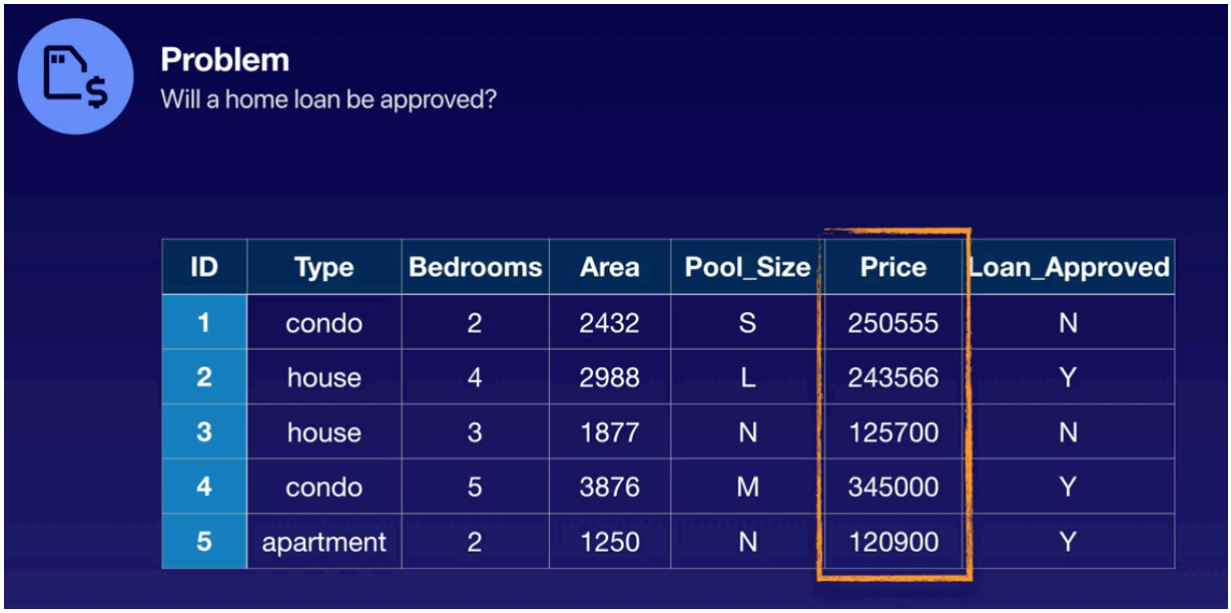
Numerical Feature Engineering

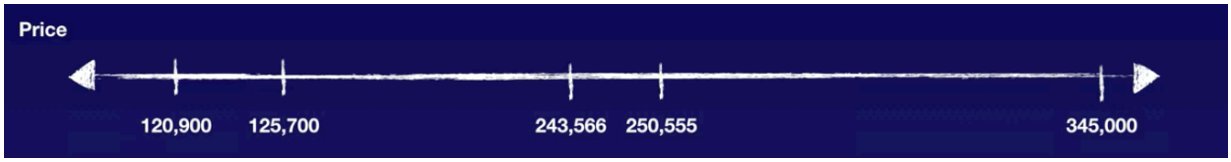
It’s about transforming numeric values within our data so ML Lagos can better analyze them.
In other words, it’s about changing numeric values in our datasets so they are easier to work with.

2 techniques:
- **Feature scaling**: take large numbers and scale them down to lower numbers and avoid skewing results
- **Binning**: takes numeric values and group them together in groups or bins.

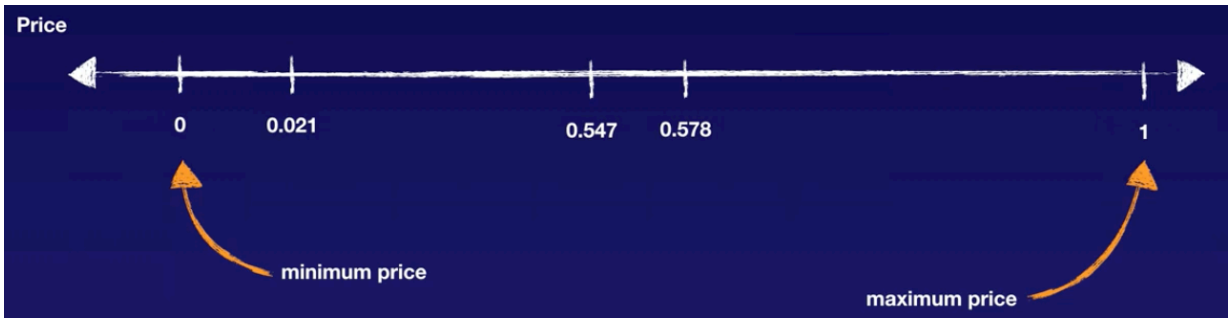


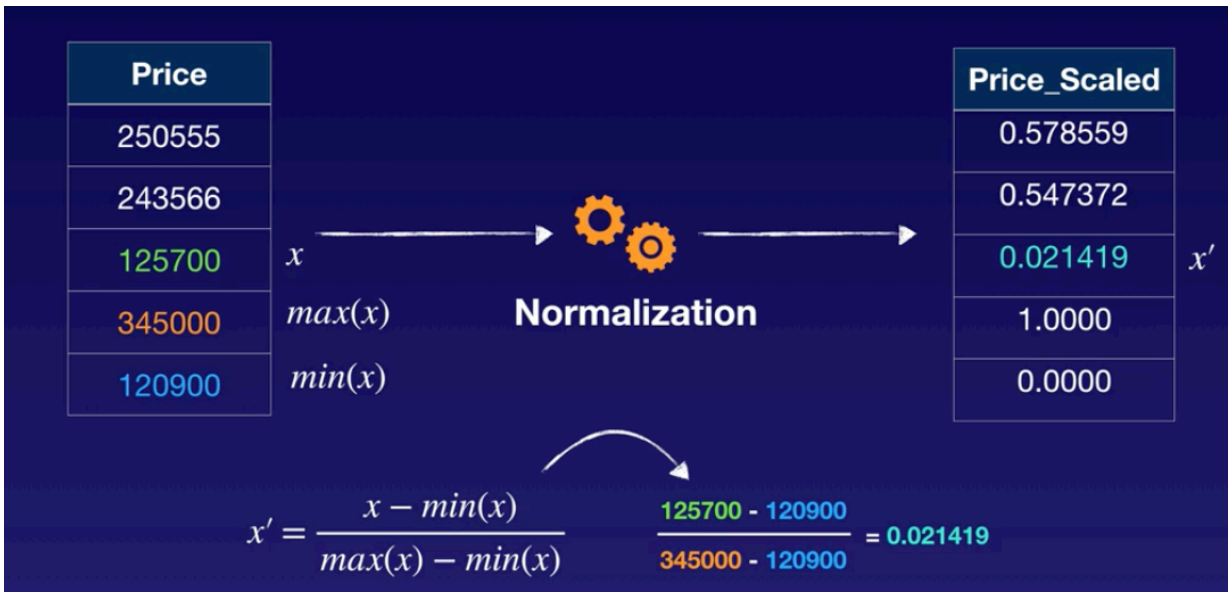
Normalization/Scaling
Pb - will a loan be approved.





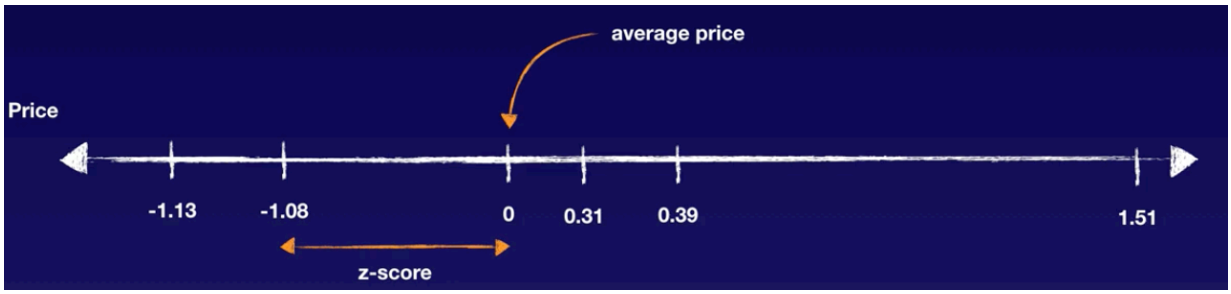
Changed to:





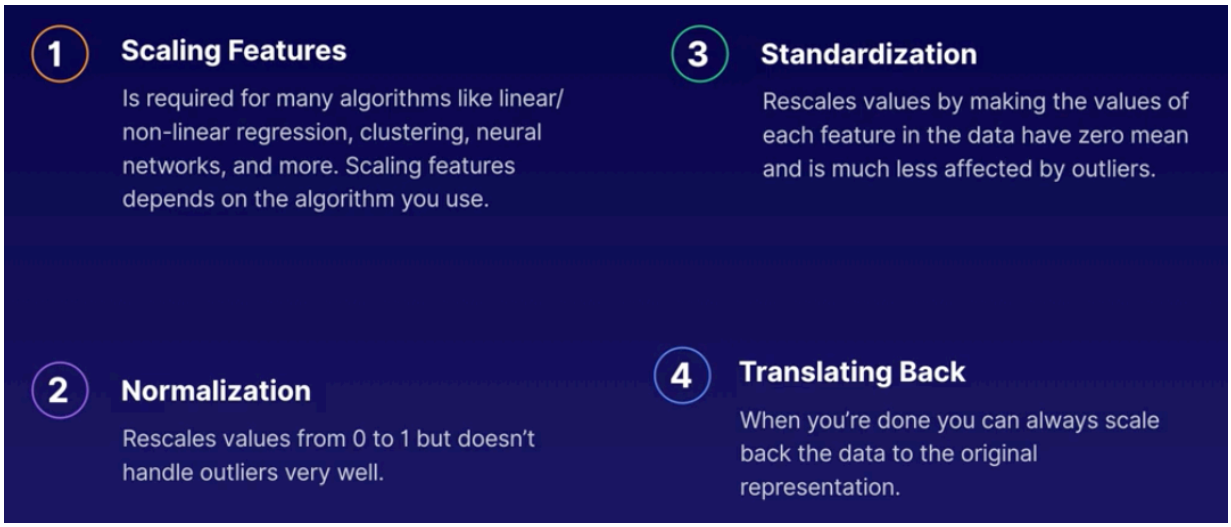
Normalization is the most common and easiest technique to scale down values. But pb is that **outliers** can throw off normalizers.

Another type of feature scaling is **Standardization**
Put avg on 0 and put z-score that takes into account stdev and smoothen the values so outliers are not as important

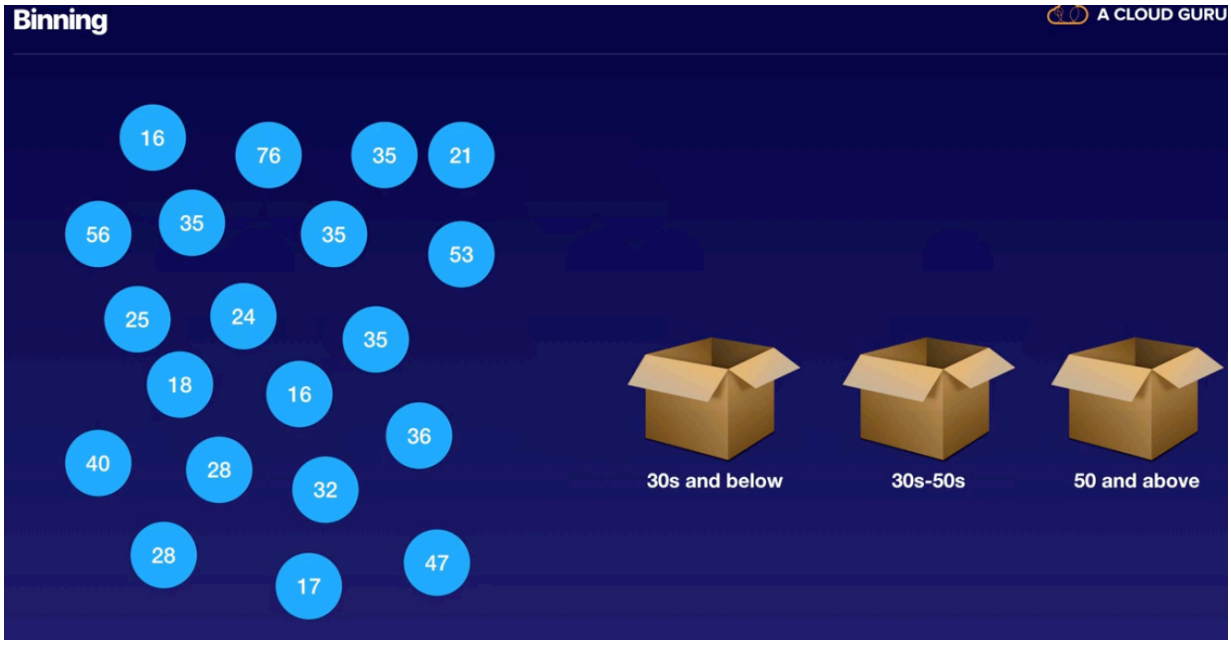


==> If outliers are a pb, use **Standardization** over **Normalization**
- **Normalization** is on a scale from 0 to 1
- **Standardization** uses the avg as 0 and other values scaled by their z-score

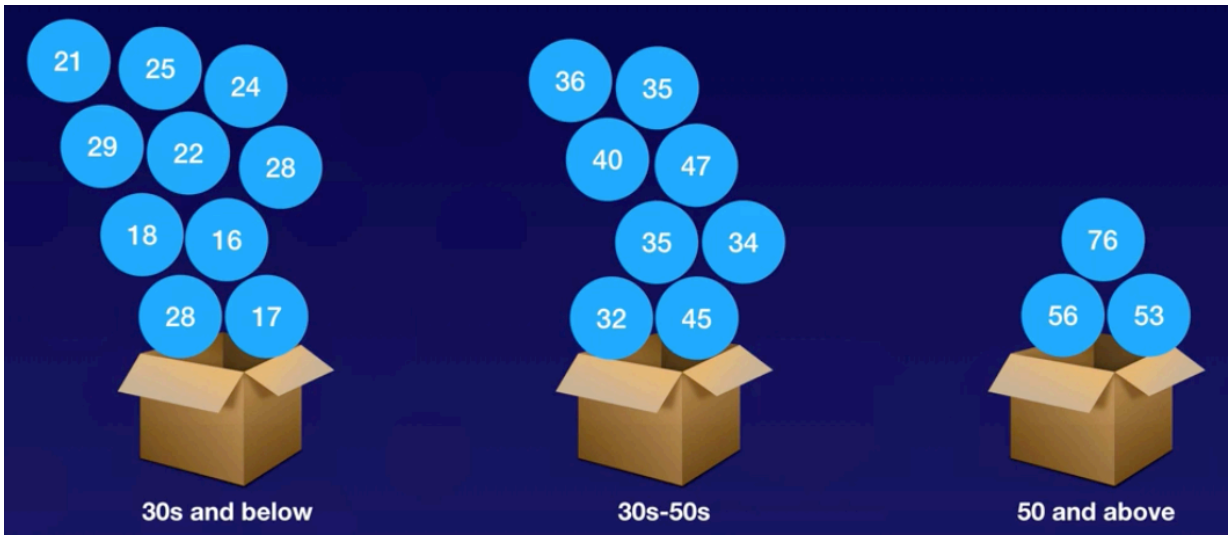
Scaling summary:



Binning

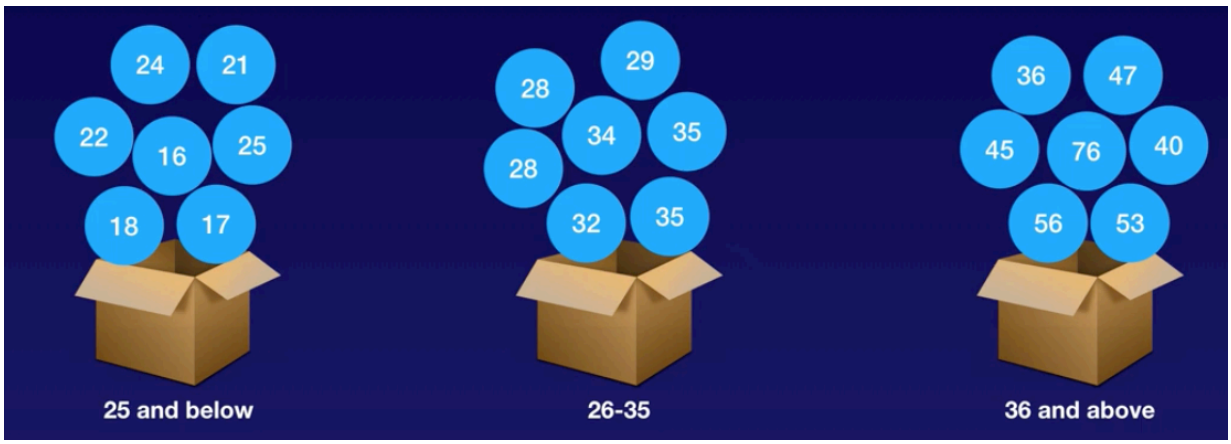


Example of age
Since we are not concerned about someone aged 50 or 51, we use bins and now we can use these as categorical variables



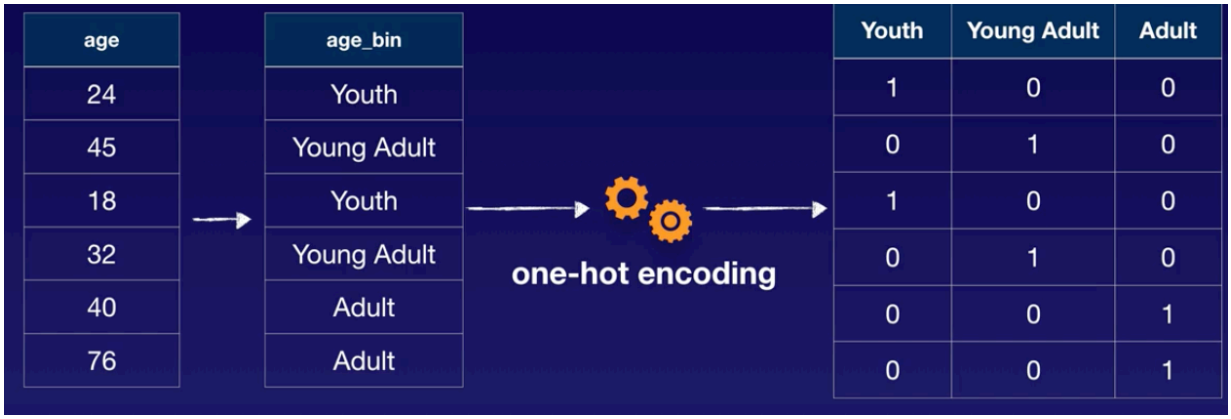
Pb - we can end up with dat beings queued

This is where we can apply:
Quantile Binning = “equals parts” + “groups”
Range of values now changes and we have an equal number of points

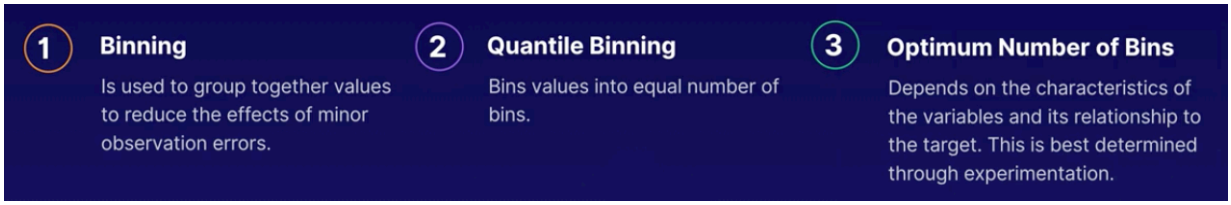


Now we can apply categorical variables to each bin: Youth, Young Adult and Adult

Then we can apply One-hot encoding



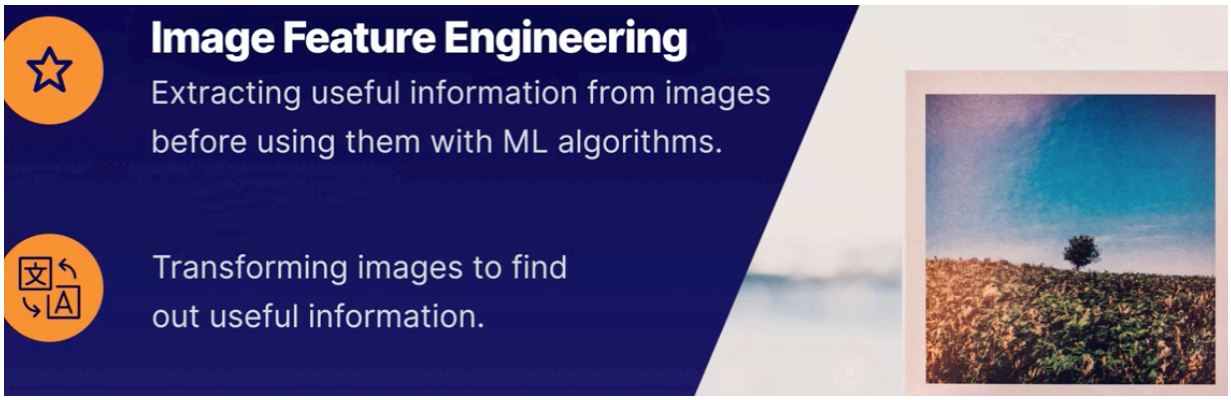
In Summary:



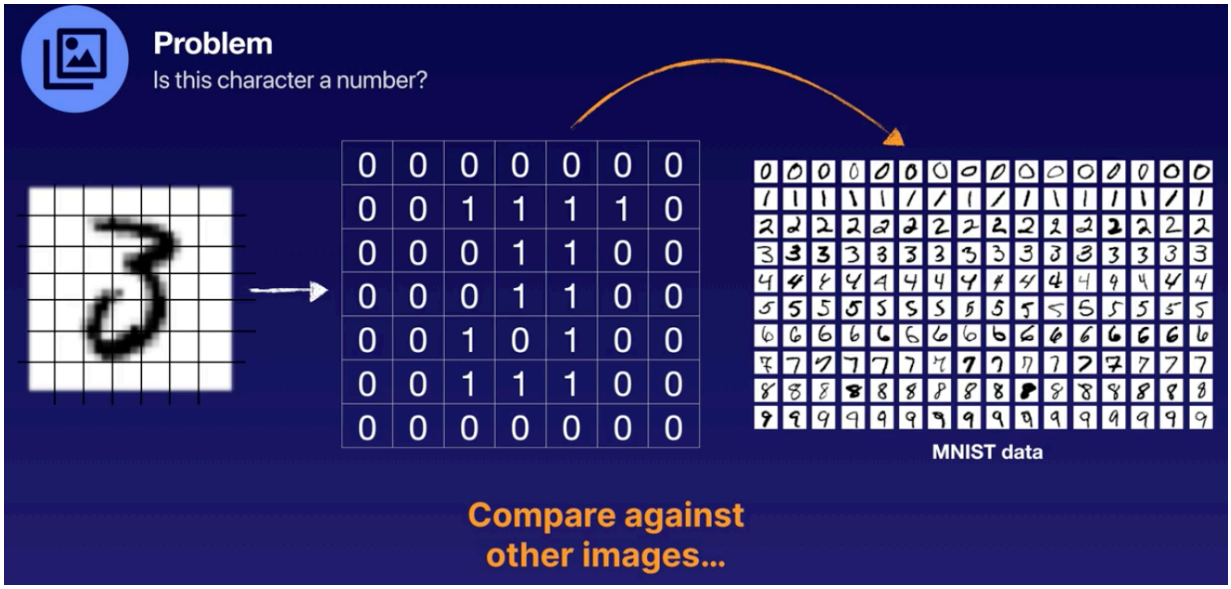
In summary:

Technique	Function
Normalization	From 0 to 1 0 - minimum value 1 - maximum value
Standardization	0 is the average value is the z-score
Quantile Binning	creates equal number of bins

Other Feature Engineering



Pb: is this character a number?
Use a grid and compare it to other grid of values from the MNISt dataset



Built-in algorithms from Amazon.

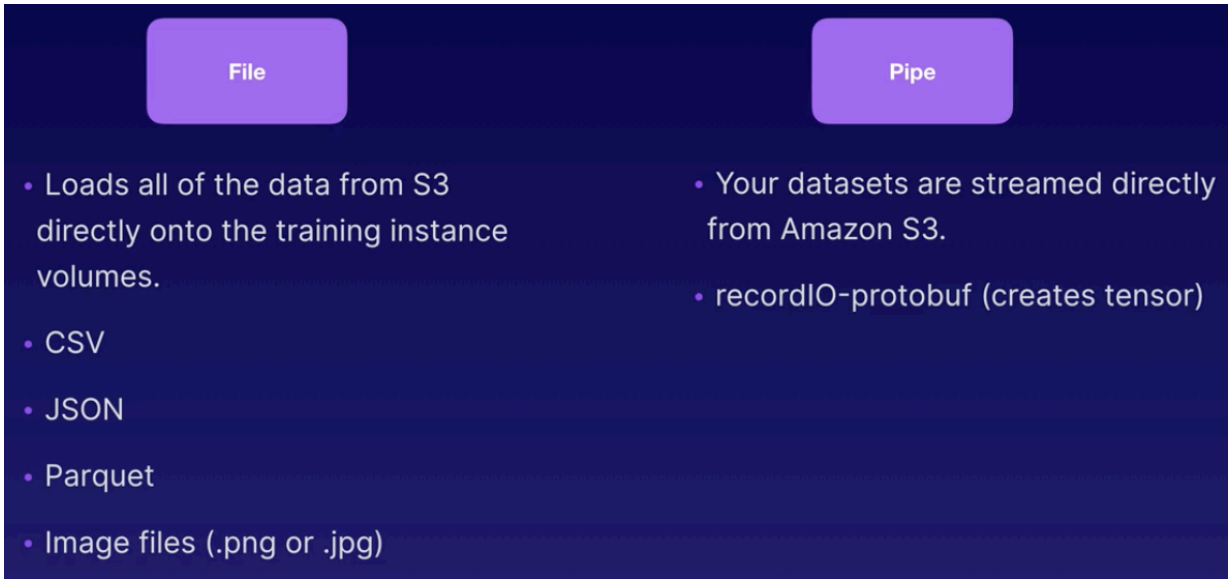
Input:

- File
- Pipe

File: AWS will copy the data from S3 to the EBS volume or whoever the algorithm runs

Pipe: stream from s3 to the instance => quicker startup => create tensor

Tensor is a multi dimensional array



Sample python code to transform data from s3 to a tensor (protobuf)

```
from sagemaker.amazon.common import write_numpy_to_dense_tensor
import io
import boto3

bucket = 'bucket-name' # Use the name of your s3 bucket here
data_key = 'kmeans_lowlevel_example/data'
data_location = 's3://{}/{}'.format(bucket, data_key)

# Convert the training data into the format required by the algorithm
buf = io.BytesIO()
write_numpy_to_dense_tensor(buf, train_set[0], train_set[1])
buf.seek(0)

# location to upload to recordIO-protobuf data
boto3.resource('s3').Bucket(bucket).Object(data_key).upload_fileobj(buf)
```

Imports

Where we want to store recrodIO-protobuf

Data format conversion

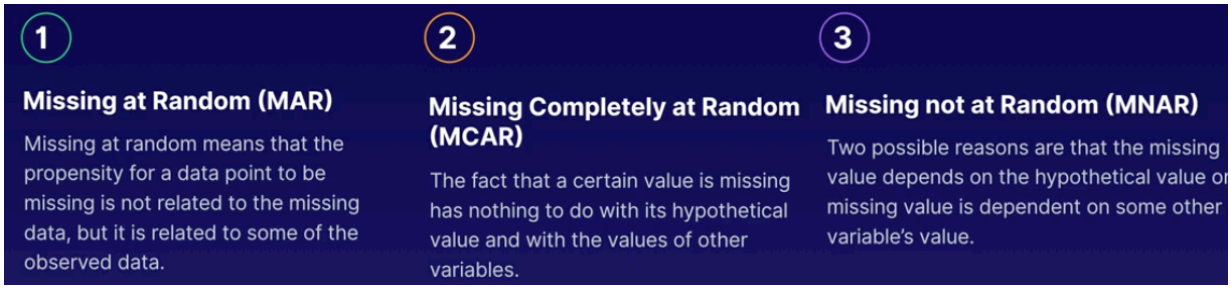
Upload to S3

Handling Missing values

Before filling in null values, need to understand why are values missing in the first place?

Missingness Mechanisms:

- **MAR**: Missing at Random
- **MCAR**: Missing completely at Random
- **MNAR**: Missing not at Random



How to handle missing values?

- Supervised Learning with an algorithm like K-nearest neigh for of fuzzy-kmeans
- Mean
- Median
- Mode: most common value
- Drop the rows

Technique	Why this works	Ease of Use
Supervised learning	Predicts missing values based on the values of other features	Most difficult, can yield best results
Mean	The average value	Quick and easy, results can vary
Median	Orders values then choses value in the middle	Quick and easy, results can vary
Mode	Most common value	Quick and easy, results can vary
Dropping rows	Removes missing values	Easiest but can dramatically change datasets

Replacing data is known as **imputation**

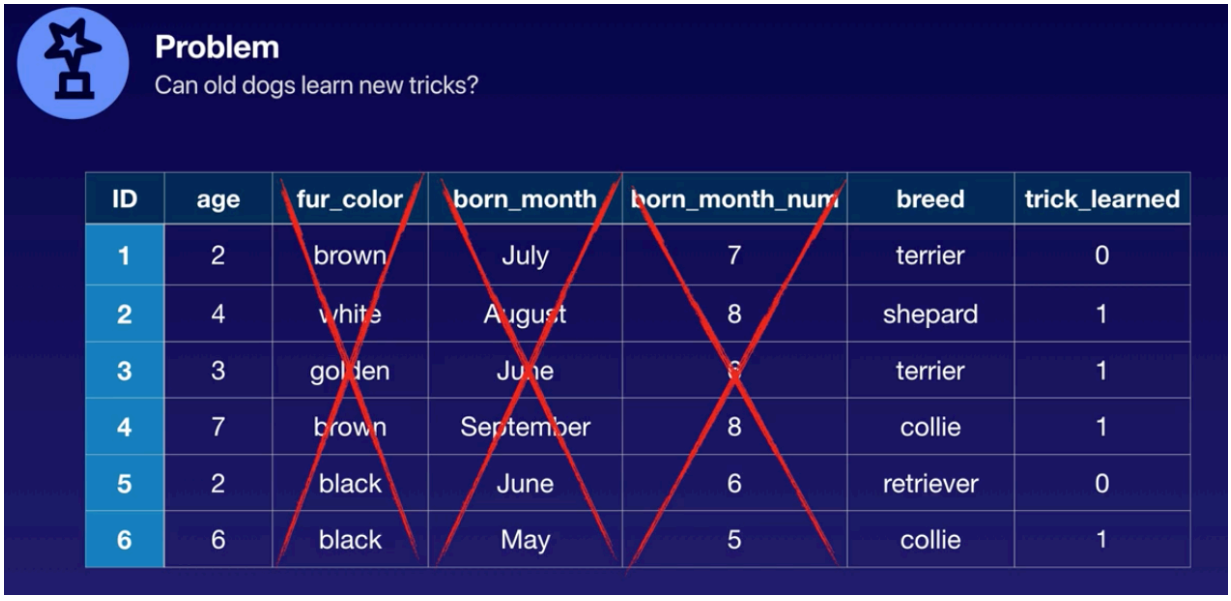
Technique	Why this works	Ease of Use
Supervised learning	Predicts missing values based on the values of other features	Most difficult, can yield best results
Mean	The average value	Quick and easy, results can vary
Median	Orders values then choses value in the middle	Quick and easy, results can vary
Mode	Most common value	Quick and easy, results can vary
Dropping rows	Removes missing values	Easiest but can dramatically change datasets

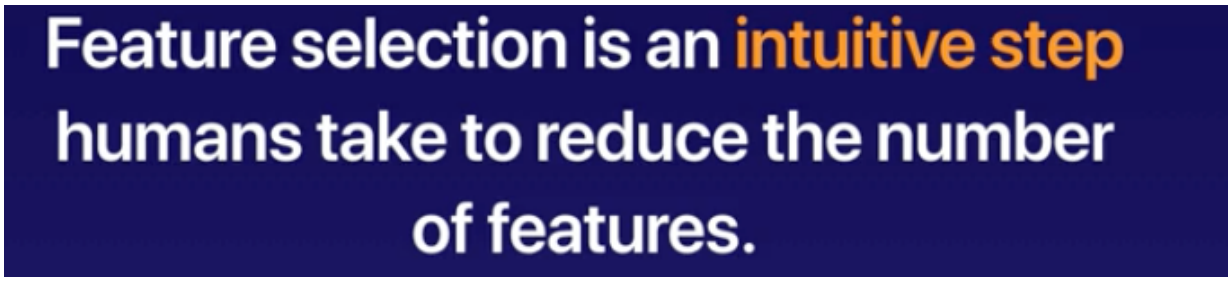
Replacing data is known as imputation

Feature Selection

=> Select the most relevant features from dataset to prevent over-complicating the analysis, resolving potential inaccuracies, and removes irrelevant features or repeated information.
In other words, it's deciding what to keep and what to get rid of

Example



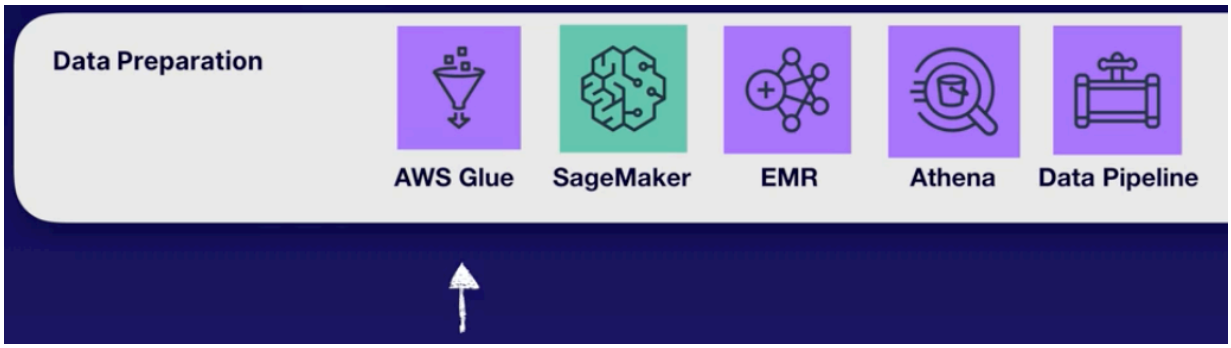


PCA - Principal Component Analysis
Similar to feature selection but it is an unsupervised ML technique that reduces the number of features while still retaining as much information as possible.
In other words, it reduces the total number of features in a dataset.

Problem	Technique	Why
Data is too large due to the large number of features	Principle Component Analysis (PCA)	Algorithm that reduces total number of features
Useless features that do not help solve ML problem	Feature Selection	Remove features that do not help solve the problem

Data Preparation Tools
- **AWS Glue** - most commonly used to transform data. Fully managed ETL
https://docs.aws.amazon.com/glue/latest/dg/how-it-works.html
AWS Glue uses other AWS services to orchestrate your ETL (extract, transform, and load) jobs to build data warehouses and data lakes and generate output streams. AWS Glue calls API operations to transform your data, create runtime logs, store your job logic, and create notifications to help you monitor your job runs. The AWS Glue console connects these services into a managed application, so you can focus on creating and monitoring your ETL work
You don't need to create the infrastructure for an ETL tool because AWS Glue does it for you. When resources are required, to reduce startup time, AWS Glue uses an instance from its warm pool of instances to run your workload.
With AWS Glue, you create jobs using table definitions in your Data Catalog. Jobs consist of scripts that contain the programming logic that performs the

transformation. You use triggers to initiate jobs either on a schedule or as a result of a specified event. You determine where your target data resides and which source data populates your target. With your input, AWS Glue generates the code that's required to transform your data from source to target. You can also provide scripts in the AWS Glue console or API to process your data.



Data Sources

AWS Glue supports the following data sources:

- Data stores
  - Amazon S3
  - Amazon Relational Database Service (Amazon RDS)
  - Third-party JDBC-accessible databases
  - Amazon DynamoDB
- Data streams
  - Amazon Kinesis Data Streams
  - Apache Kafka

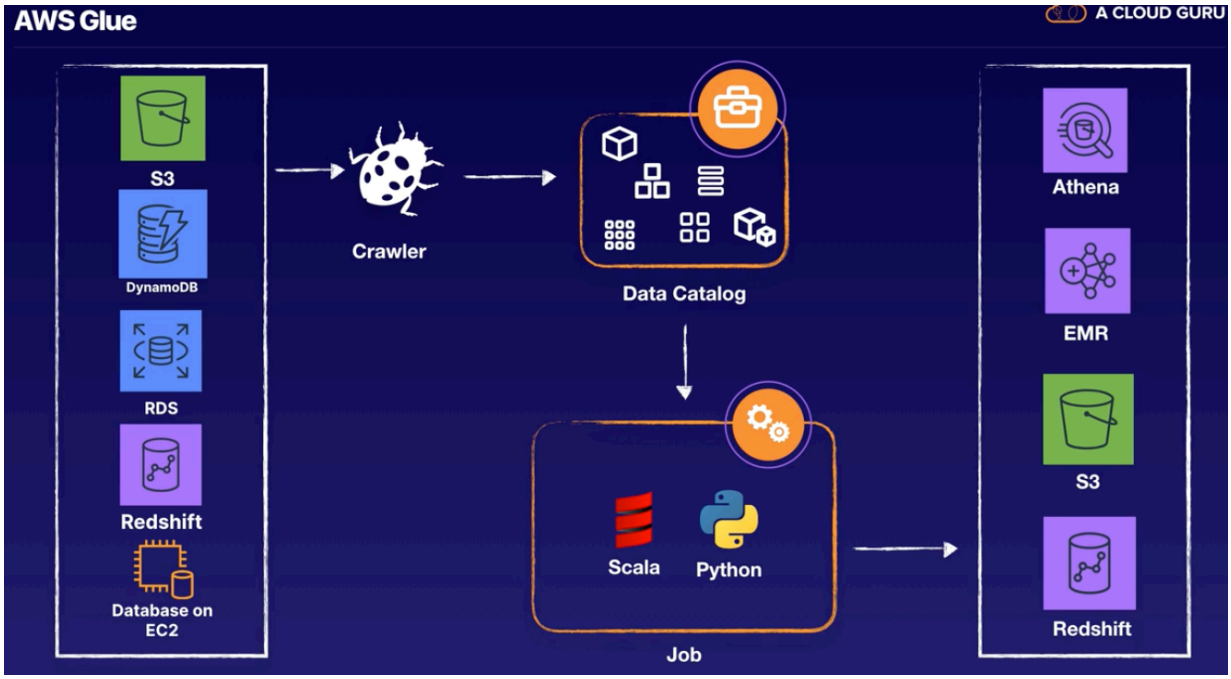
Data Targets

AWS Glue supports the following data targets:

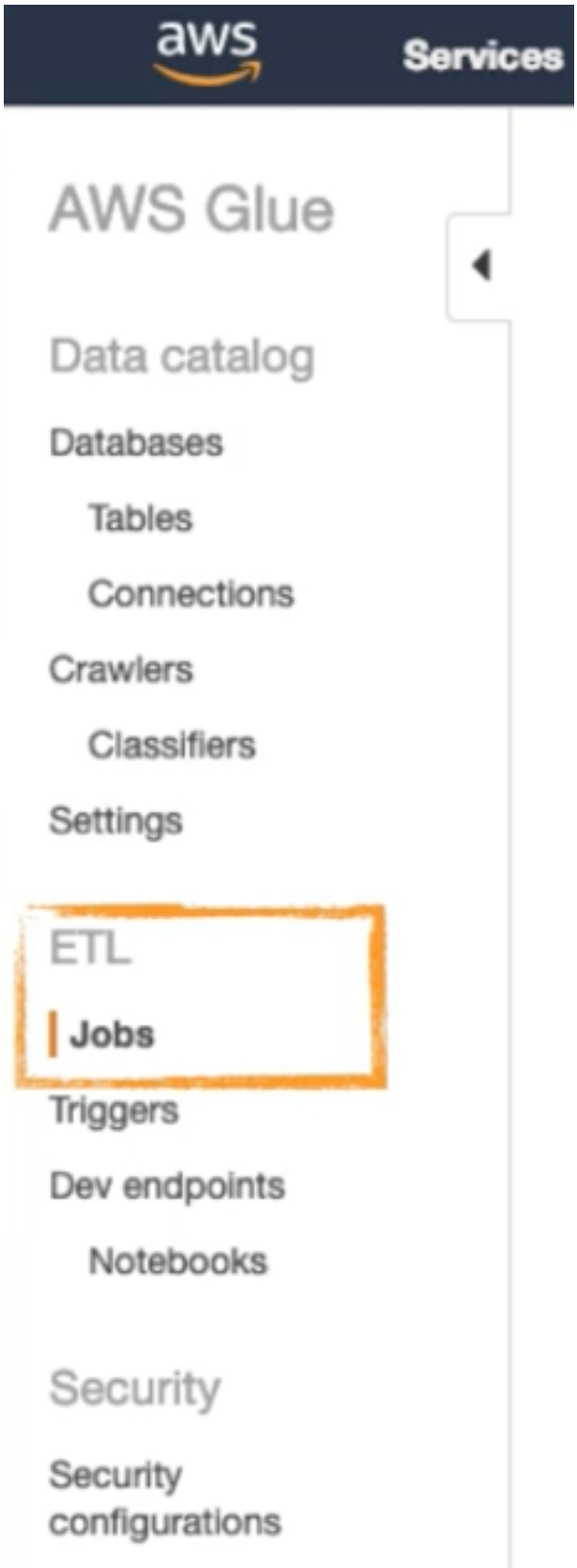
- Data stores
  - Amazon S3
  - Amazon Relational Database Service (Amazon RDS)
  - Third-party JDBC-accessible databases

Use one of the services built into AWS Glue: S3, DynamoDB, RDS, RedShift
Set up a crawler, determines structure/schema. It then creates a data catalog (metadata of our data)

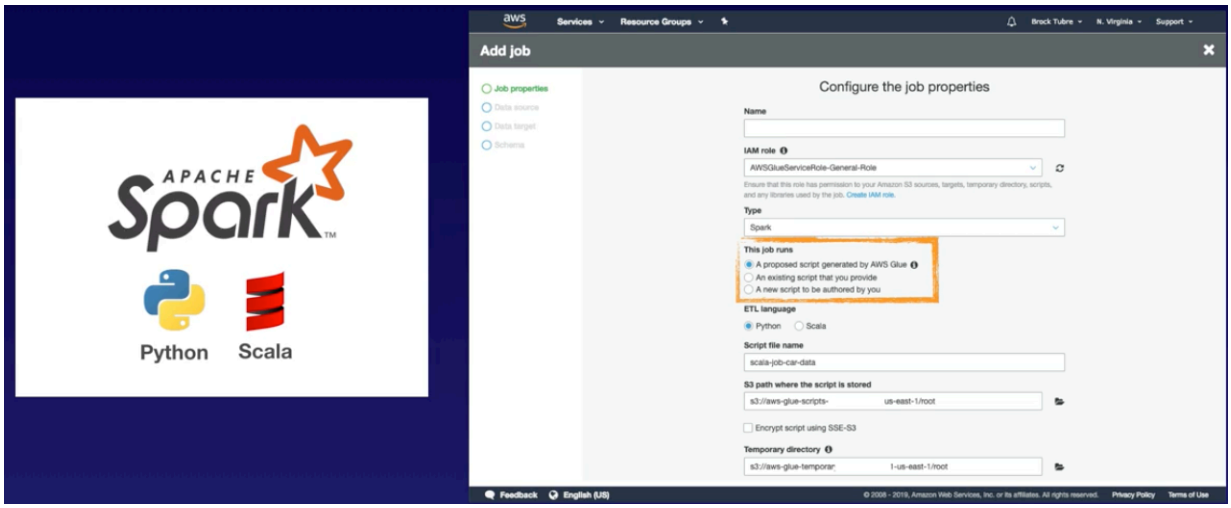
Once data catalog is created, we can set up jobs and run Scala or python jobs => categorical encoding, feature engineering, selection,...
Jobs can run on demand, on schedule or when an AWS service is triggered (like S3 trigger)
Once data is transformed, it can then be put in a data target: S3, EMR, RedShift, Athena and use within SageMaker



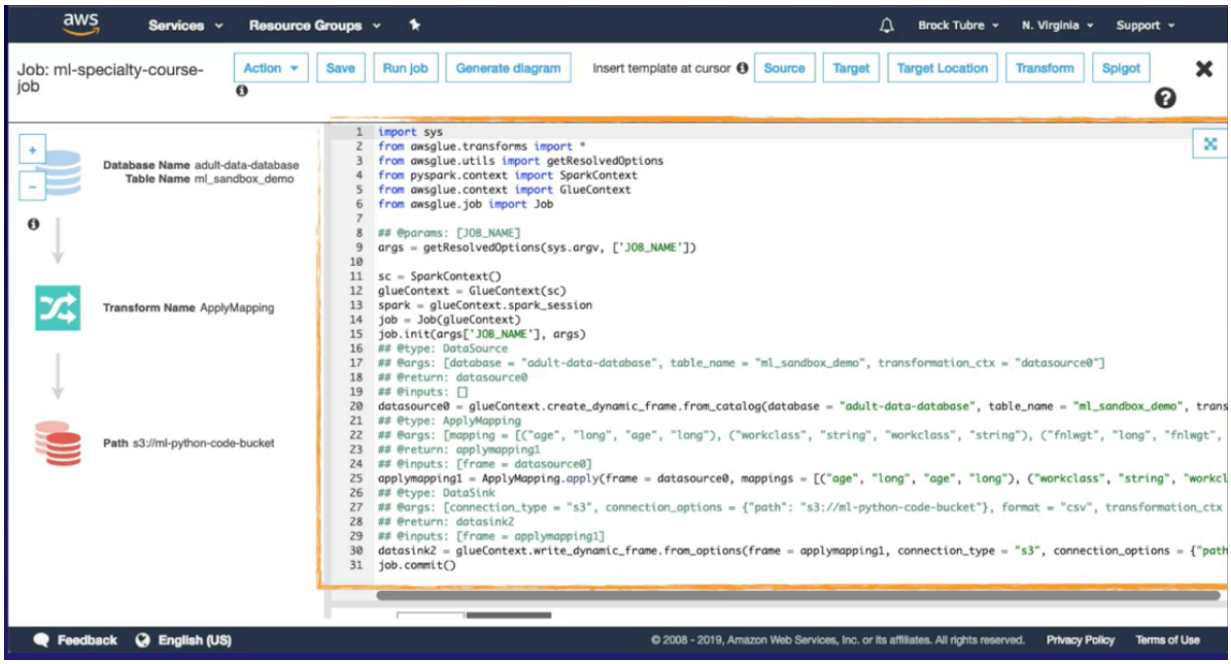
AWS Glue in console:
 - Data Catalog: Database + crawler (classify the data source, infer the schema from the source,...). From the crawler, it creates tables in the DB
 - ETL: jobs run to populate the tables



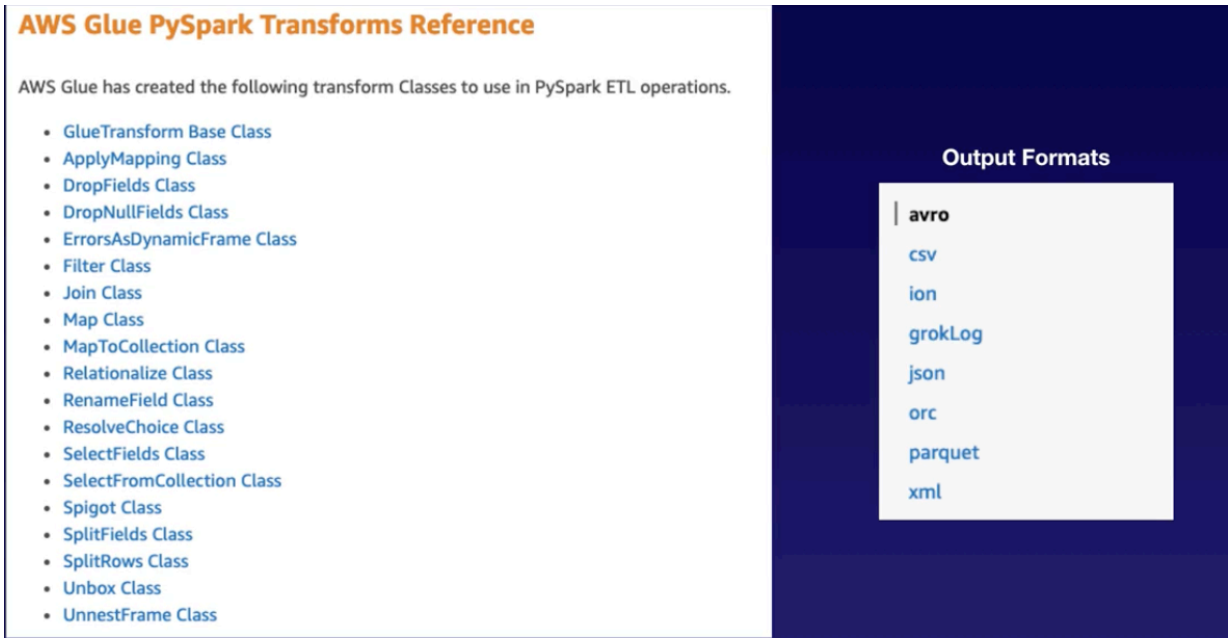
A job can be in Spark (default)
We can then use Python or Scala for ETL language



If we let AWS Glue generate the code for us, it will look like this:

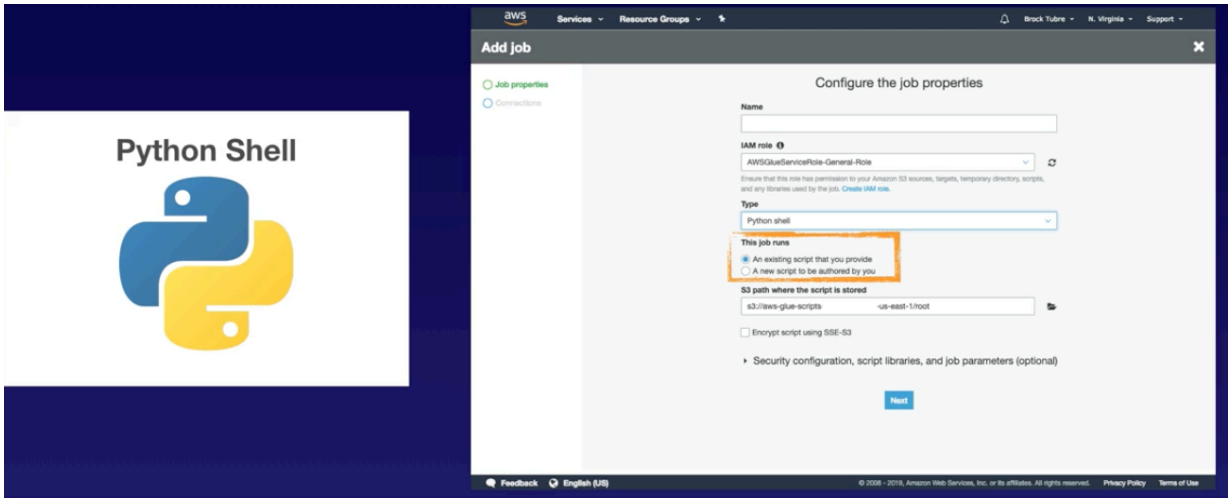


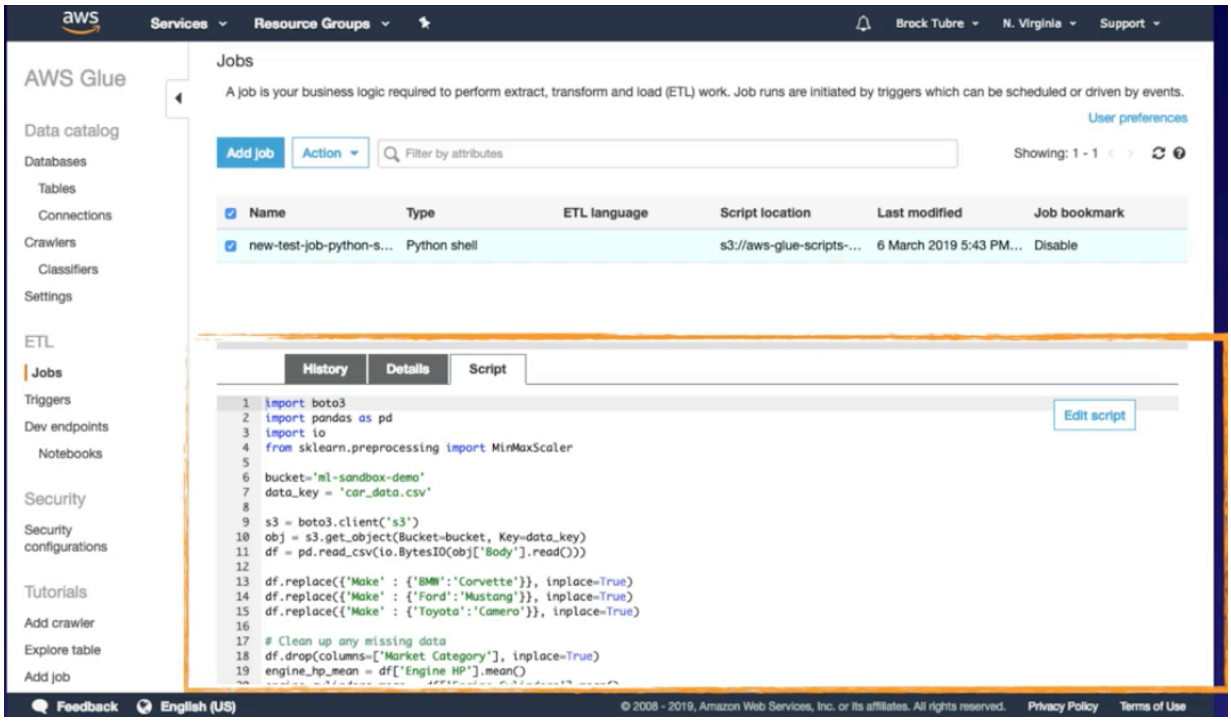
To note we can use built in Transforms to add to our scripts



The job will create “py-spark” score for us.

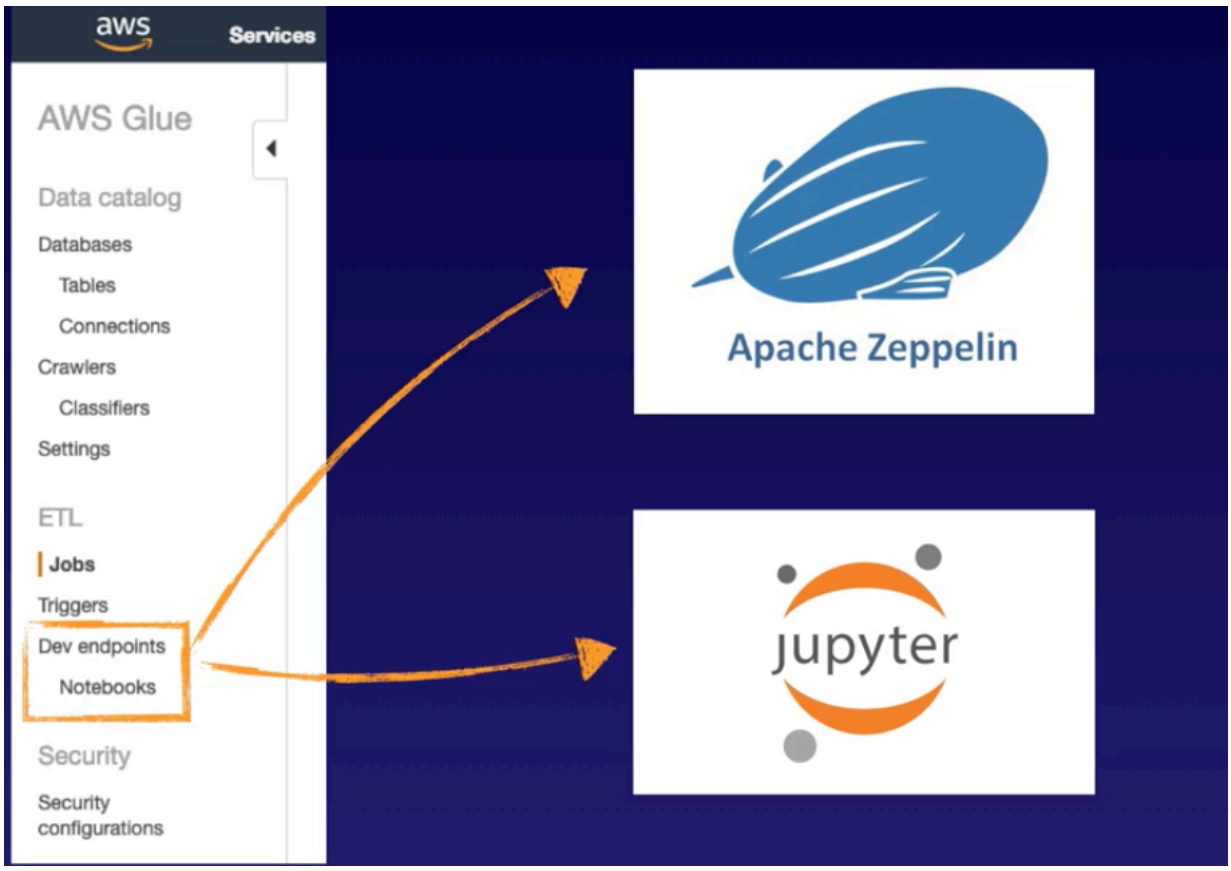
If we are more comfortable with traditional Python and libraries like numpy, pandas, scikit learn, we can use “Python Shell” as a job



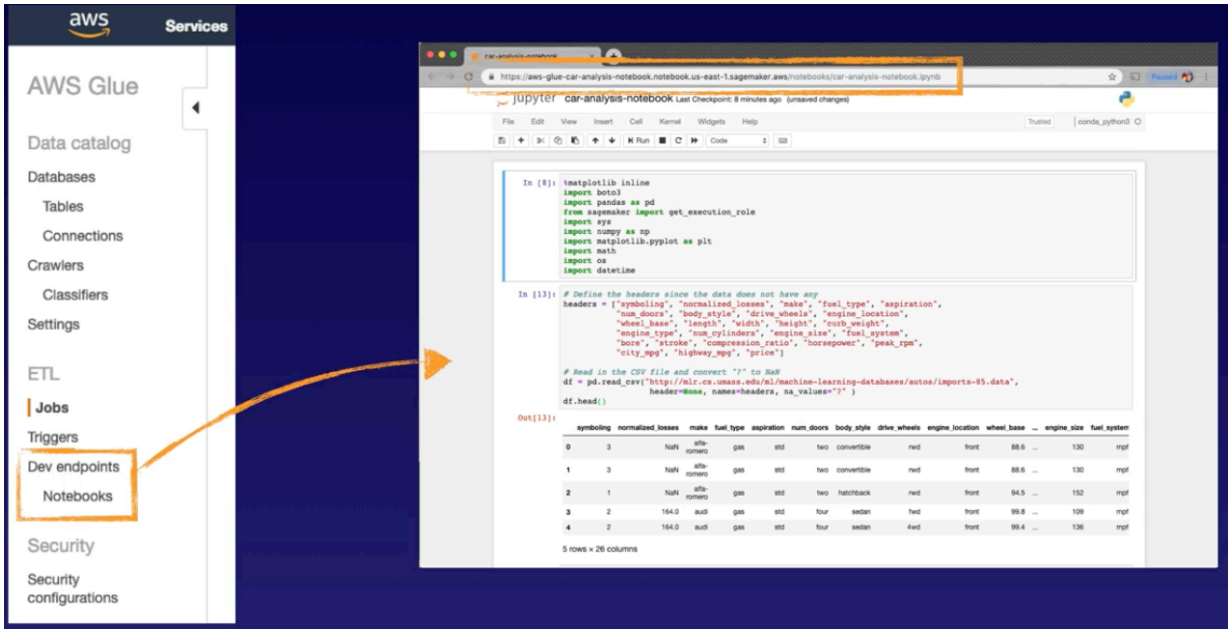


Built-in libraries to transform our data:

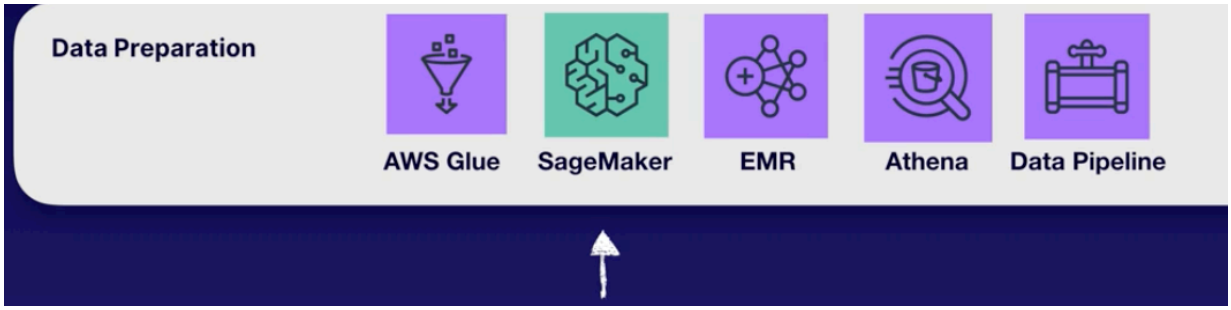
Lastly, AWS Glue allows us to create Zeppelin and Jupiter notebooks to do ad-hoc transformations



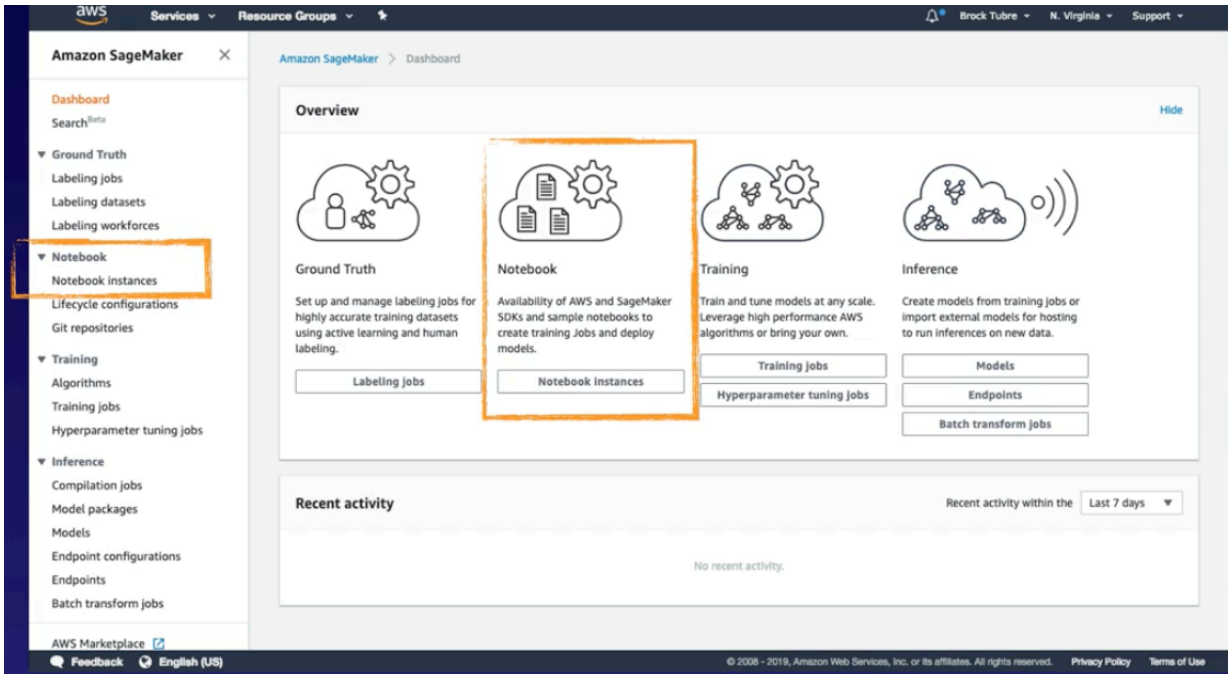
The notebook will be hosted as part of SageMaker:



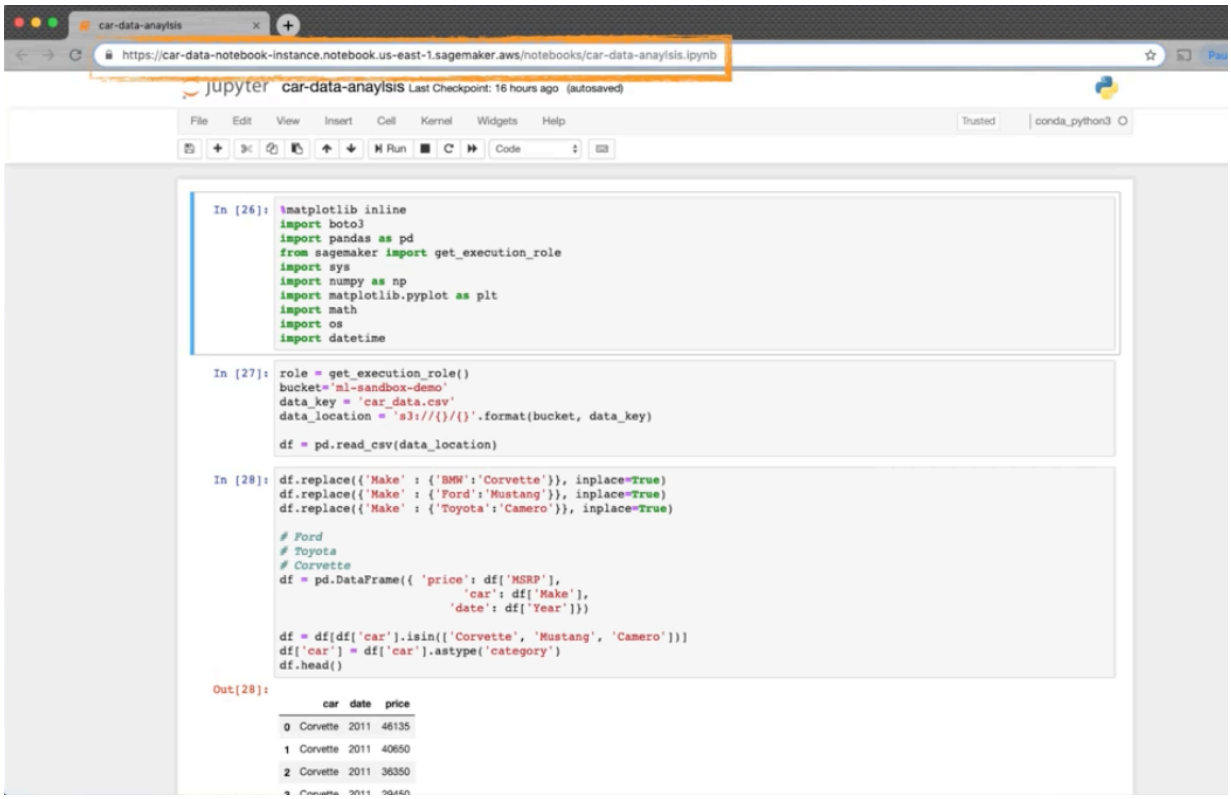
Other Data Preparation tool (and more than that): SageMaker



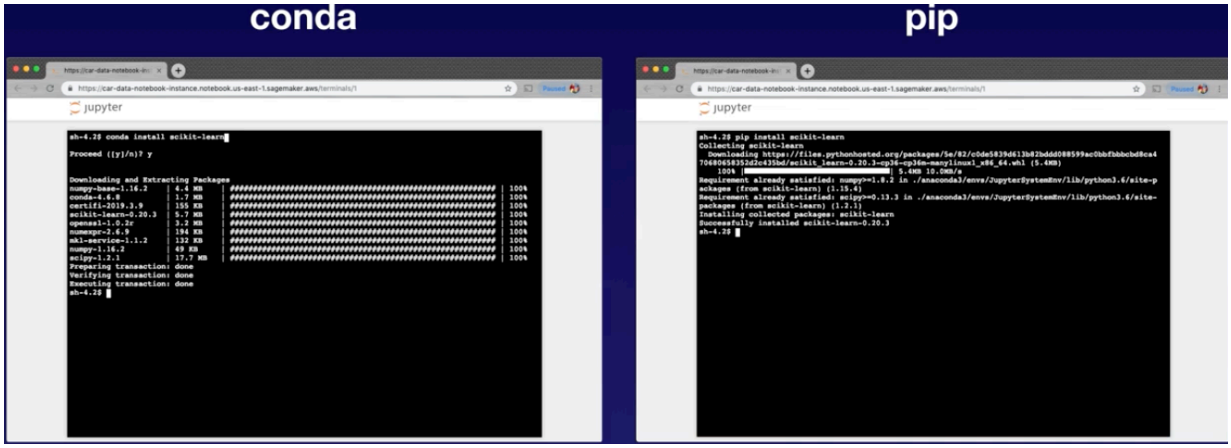
SageMaker allows us to create Jupiter notebooks directly integrated with the service.



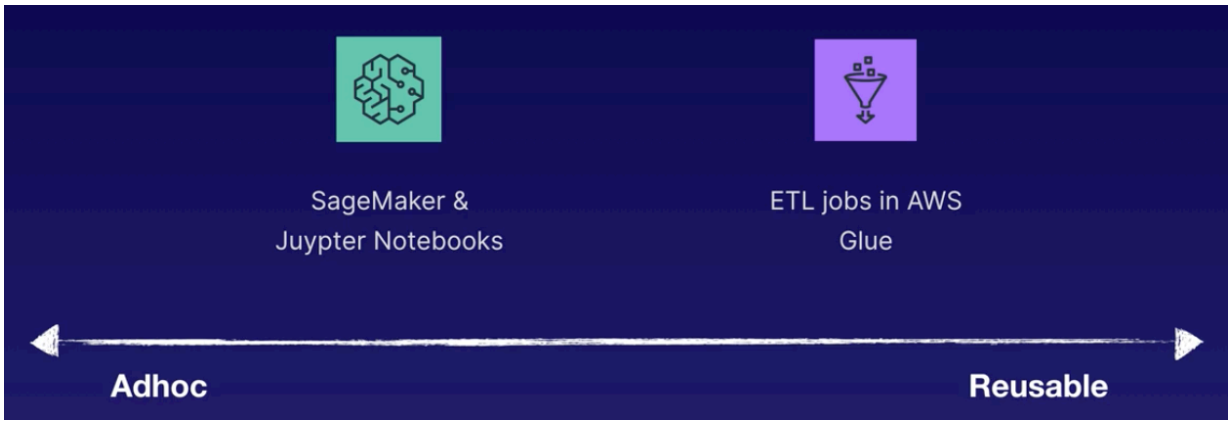
From the notebooks, we can use many of the python libraries most common



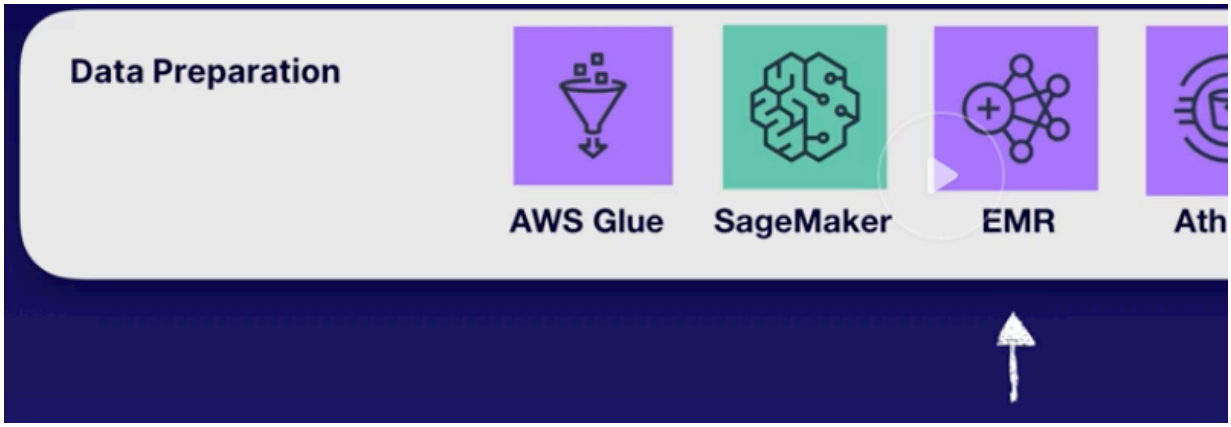
We can also install other modules with package managers like condo or pip



So use AWS Glue for long standing or repetitive transformations to do on schedule or when actions are required
Use Jupiter notebooks for quick transformation or ad-hoc transformation on the data before it feeds the model.



Other datA transformation tool: EMR

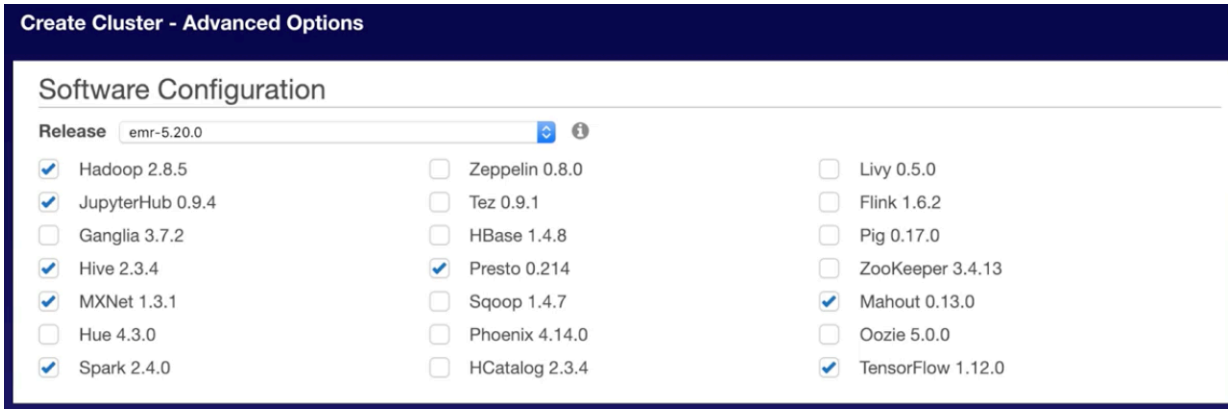


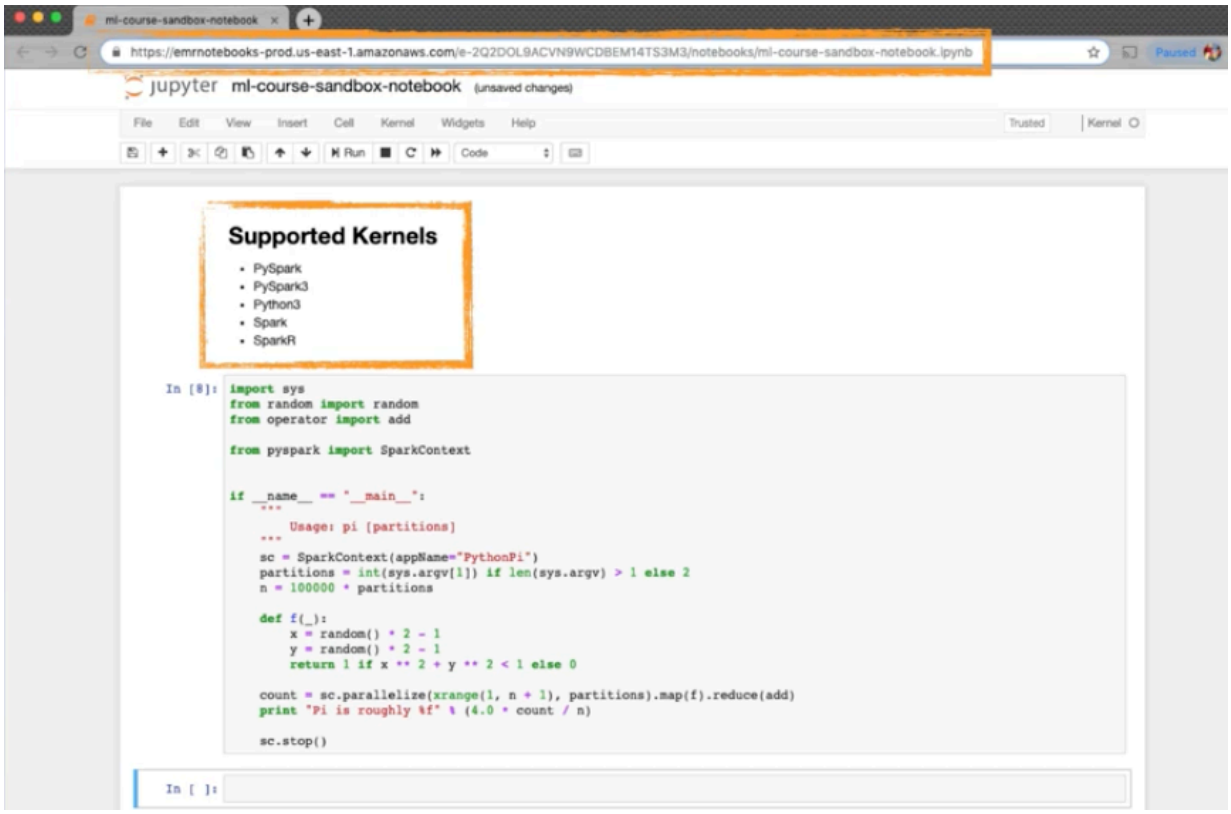
EMR is a fully managed Hadoop cluster that runs on multiple EC2 instances
It allows us to use different frameworks like Tensorflow, spark, flume, Jupiter, Monet,...

It’s more work to set up transformation than SageMaker.
It is great at scaling petabytes of data.

To note sageMaker and EMR can integrate together. We can use SageMaker SDK to connect it to an existing EMR cluster.
So if data is set up in EMR cluster, we can use Apache Spark to integrate directly with SageMaker.
Least amount of effort = AWS Glue. Since it is a fully managed service, we don’t need to set it up like we would have to do with EMR.

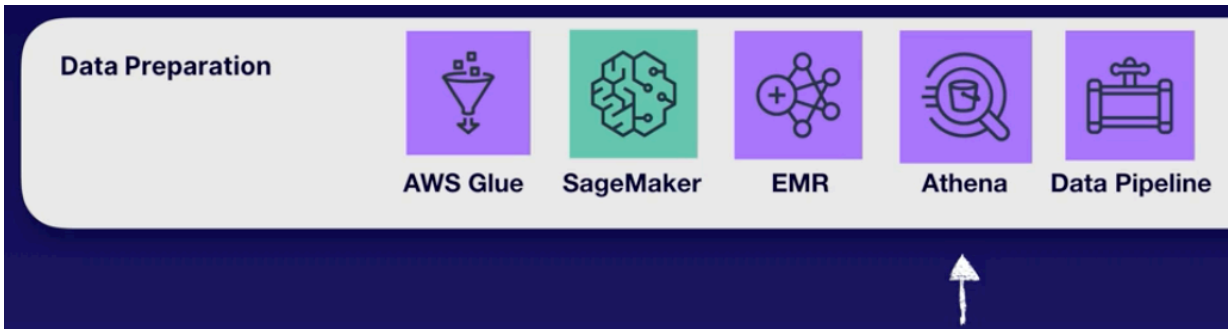
When creating cluster within EMR, we can use frameworks lthat we want to include in the cluster
And since Jupiter is built into EMR, we can use the notebooks to transform the data and use services like Hive or Apache Spark to run the ETL jobs



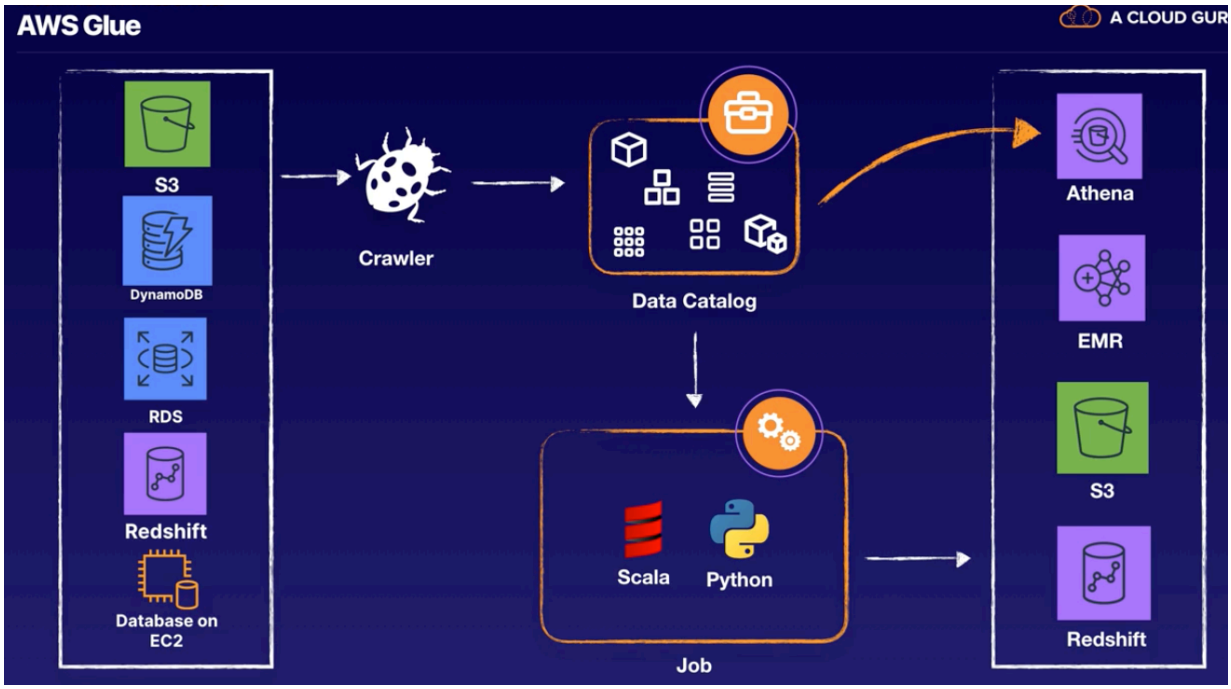


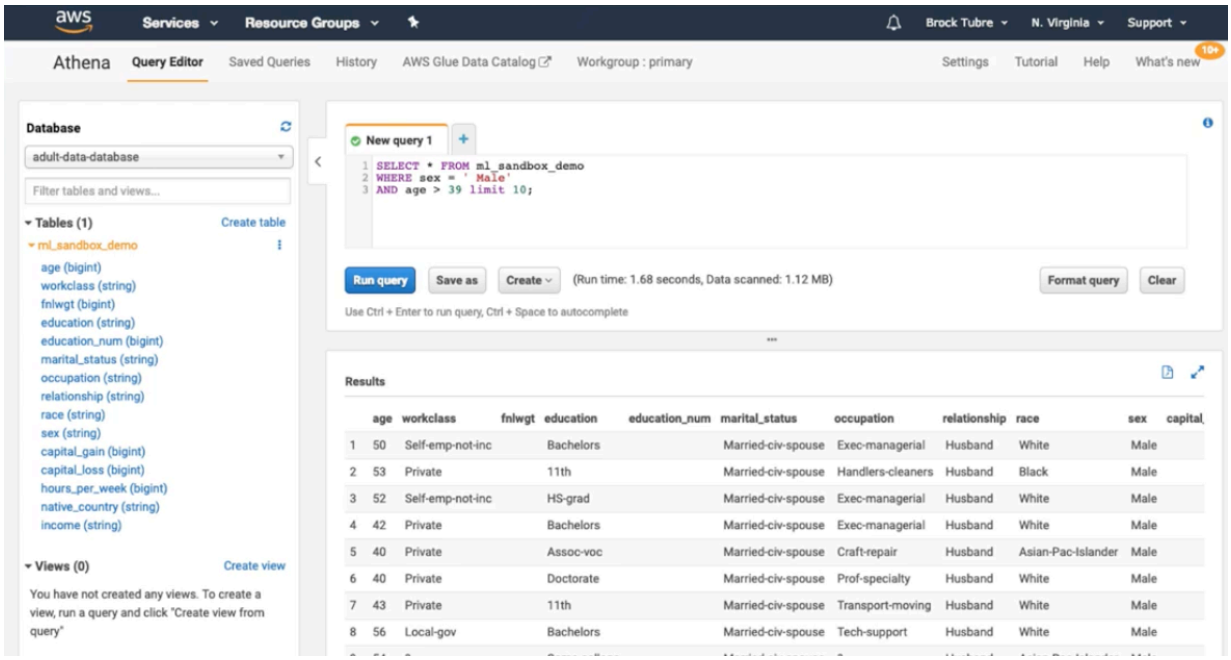
Other Data preparation tool: Athena

To run SQL queries on S3 data. Fully managed server less service.



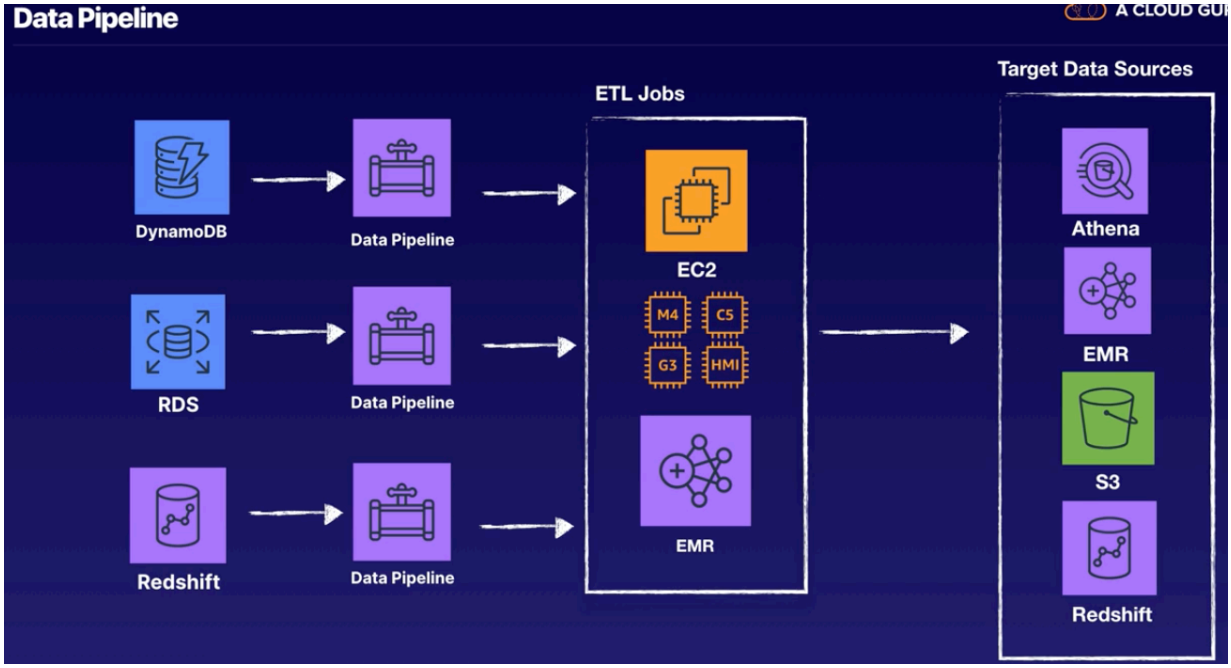
We can set a data catalog with AWS Glue, then query it with Athena



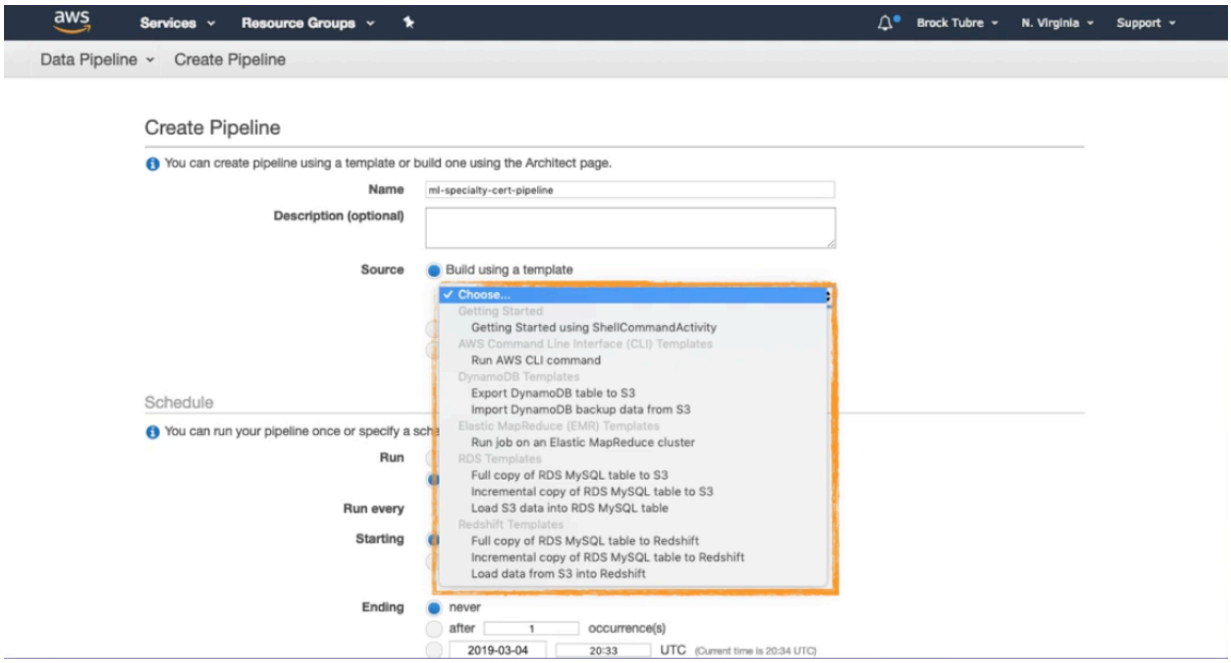


Last data transform: Data Pipeline

Process and move data between different AWS compute services : from Dynamo, RDS, Redshift to Athena, EMR, S3,...



When we want to use Java or Java script, we can load that into EC2 and use the dAta Pipeline to feed these algorithms



Which service to use?

Most cases: AWS Glue with python or Scala

We can also use Athena for simple queries against S3 if we know the data

If we want to use a language outside of python or Scala, use the data pipeline

Datasource	Data Preparation Tool	Why
S3, Redshift, RDS, DynamoDB, On Premise DB	AWS Glue	Use Python or Scala to transform data and output data into S3
S3	Athena	Query data and output results into S3
EMR	PySpark/Hive in EMR	Transform petabytes of distributed data and output data into S3
RDS, EMR, DynamoDB, Redshift	Data Pipeline	Setup EC2 instances to transform data and output data into S3