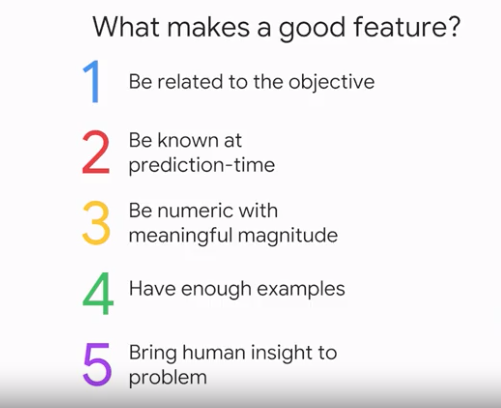
Raw Data to Features:

Creating good features, transforming them, creating synthetic features, together these three things are called preprocessing.



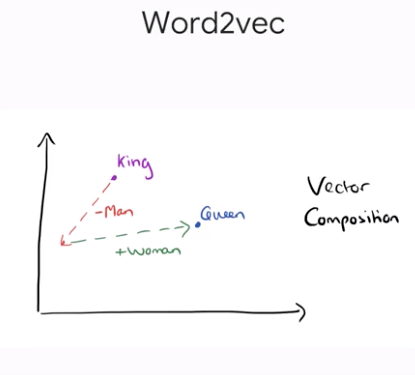
Data dredging:

Don't just throw arbitrary data in there and just hope that you can get some kind of relationship out of it. You don't want to do what's called data dredging, you don't want to dredge your large data set and find whatever spurious correlations might exist, because the larger the data set is, the more likely it is that there is a lot of these spurious correlations, and your ML model would just get confused with this mass of data you're throwing out.

Feature Engineering:

Taking categorical variable and predicting:





suppose you have words in an NLP or Natural Language Processing system, and the things that you do to the words to make them numeric is that you could typically run something like word2vec or word to vector. It's a very standard technique, and you basically take your words, and apply this technique to the word vectors, so that each word becomes a vector. And at the end of the word2vec process, when you look at these vectors, these vectors are such that if you take the vector from man and you take the vector from woman, and you actually subtract them, subtract those words, the difference that you get is going to be a very similar difference, is if you took the vector for king, and the vector for queen, and subtracted them. Interesting, right? That's exactly the Word2vec does.

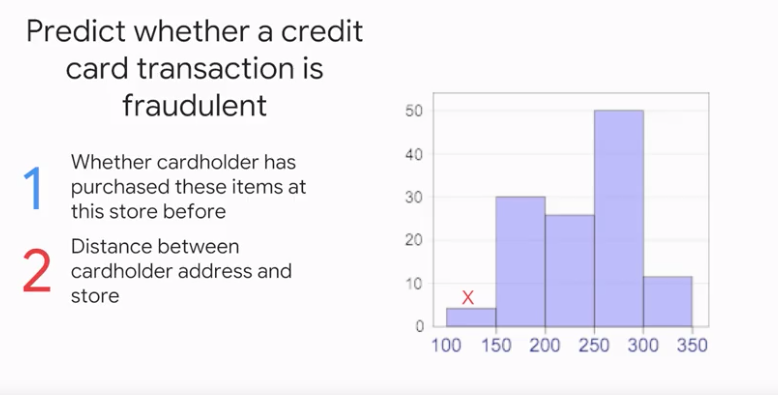
Well, you could just go ahead and throw some random encoding in there like one, two, three, four, five, but your ML model is not to be as good as if you started with a vector encoding that's nice enough to understand the context of like male, female, man, woman, king, and queen. Need to have meaningful magnitude.

You need to find a vector representations in such a way that these kinds of qualities exist for you. And one of the ways you can do these things automatically using processes called auto-encoding or embedding.

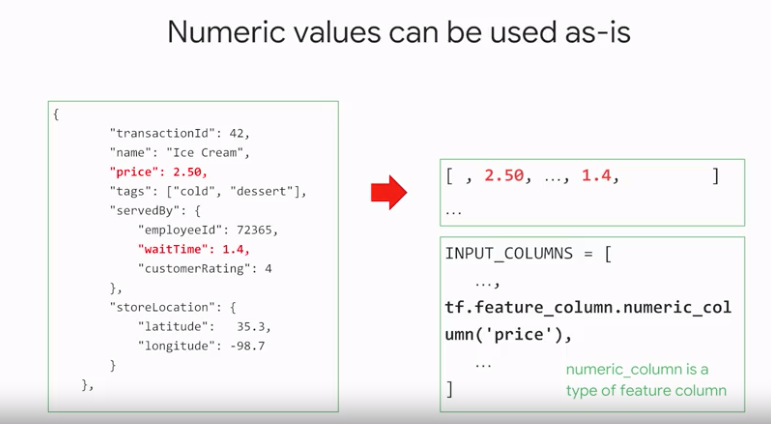
Make sure you have enough sample data in the category.

Do you have at least five examples of things that started in Q1 and Q2 and Q3 and Q4 for example. You may have to group up your values, see you have enough examples of each value.



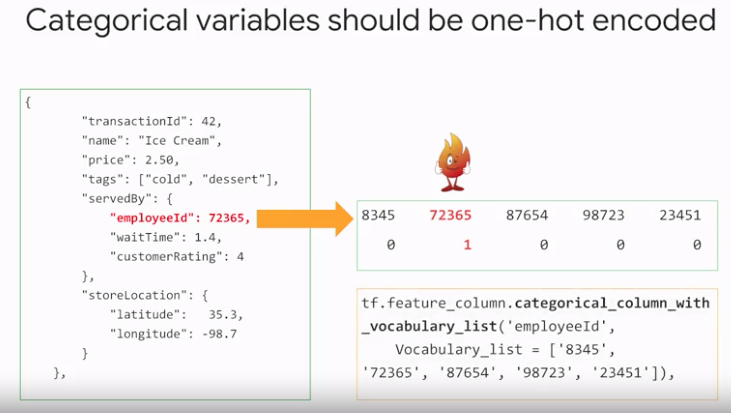


Representing Features:

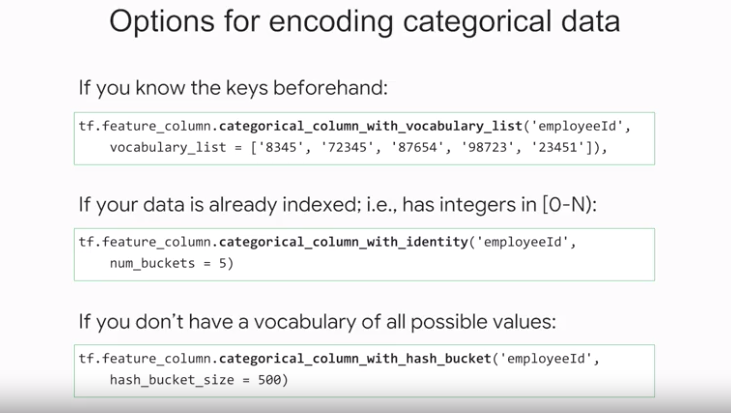




Is somebody with an employee ID as 72365's, as twice as good as an employee with an ID of 36182? No, right? So I can't use the employee ID as it is, I have to do something with them. So, let's say my ice cream shop has five employees. Employee number 8345, employee number 72365, etc. What I can do, I can say if this employee number is 72365, I'll represent this employee's ID by this vector that you see here. The vector is 01000 because I define the second column as corresponding to that employee 72365. So, essentially, I make it like a bit mask almost. You make that employee's column one and all the other columns zero. This is what's called one hot encoding, there's one column that's hot, and all the other columns are cold.



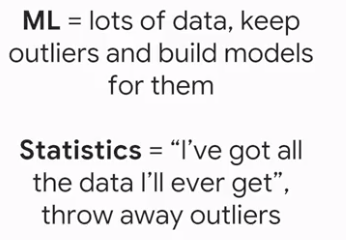




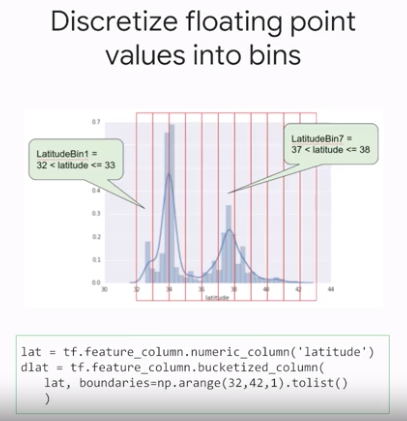




ML vs Statistics :

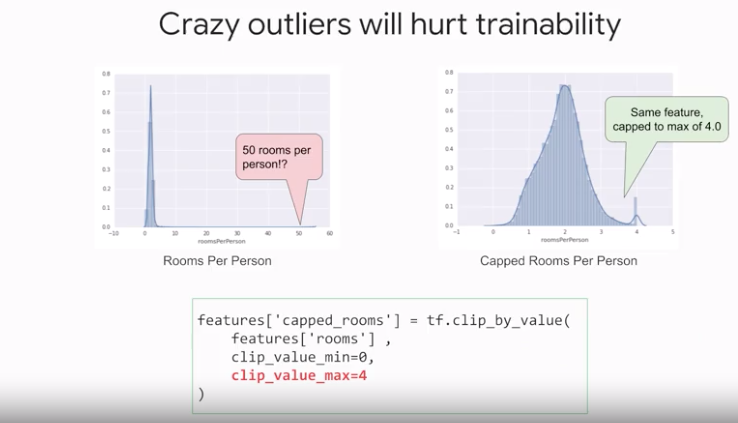


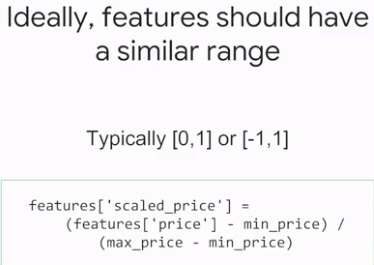
But if you take any statistics you might see if there was missing values, you would normally impute a value like the average for that column. So, that's where philosophically ML and statistics start to diverge. In ML the idea is that you build the separate model for this situation where you have the data versus when you don't.



It doesn't make sense to represent latitude as a floating point feature in our model. It's because there's no linear relationship exists between latitude and the housing values. For example, houses in latitude 35 and not 35, 34 times more expensive than houses at latitude 34. And yet individual latitudes are probably a pretty good indicator of housing values. So, what do we do with a magnitude piece?

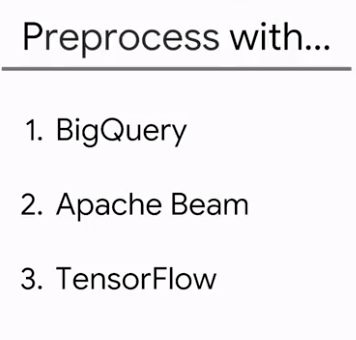
Well, what if we did this, instead of having one floating point feature let's take a look and have 11 distinct boolean features. Yes-no latitudeBin1, latitudeBin2 all the way to latitudeBin11 with yes-no boolean values. And here, we've just use fixed bin boundaries. And other options that you see commonly used between data scientist that have quantile boundaries so that the number of values in each bin is constant.

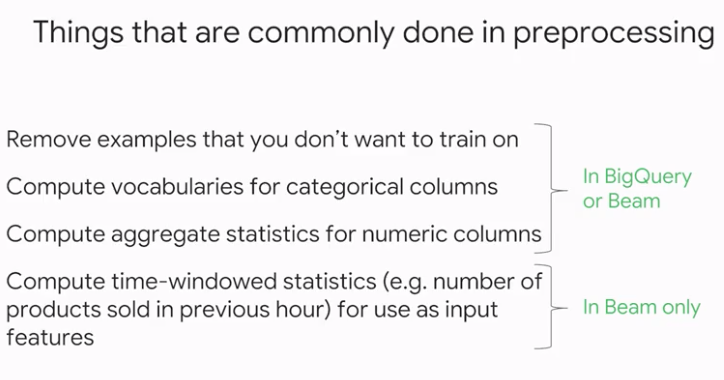


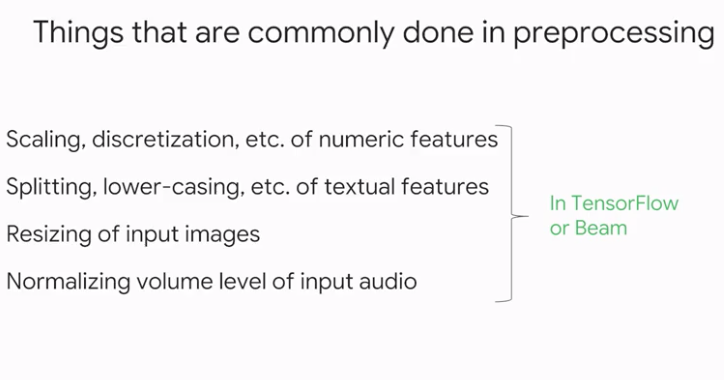


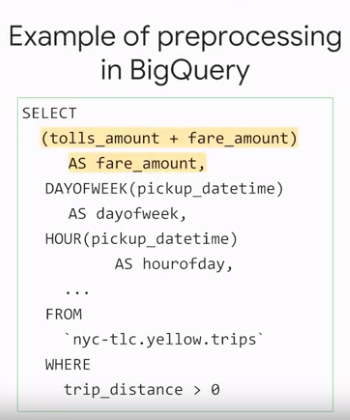
The price ends up dominating your gradient. Now, modern architectures for ML end up taking a variable magnitudes into account because of what's called batch normalization.

## Preprocess the data:

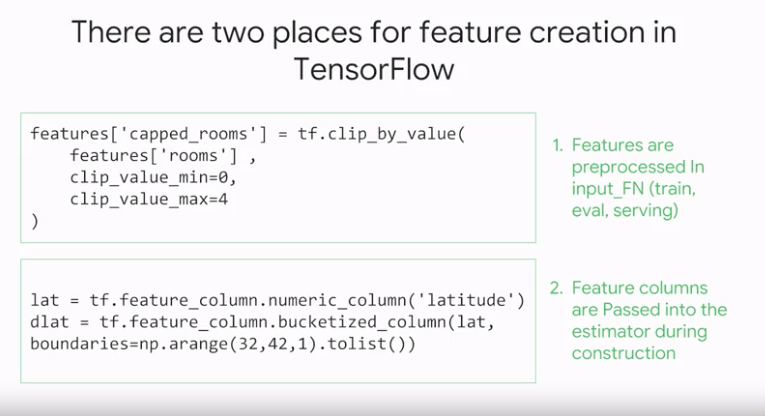


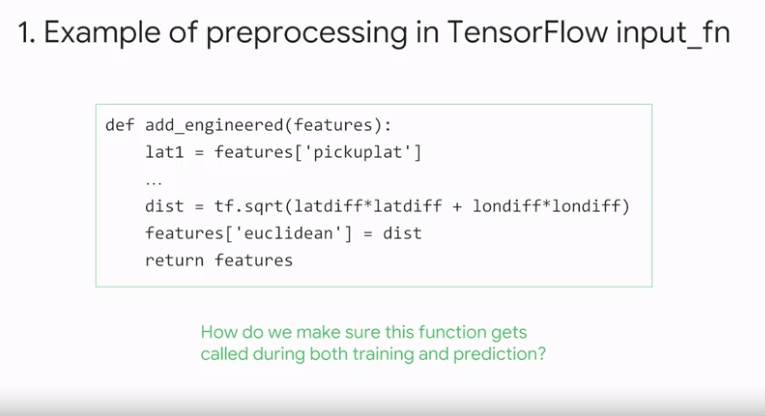


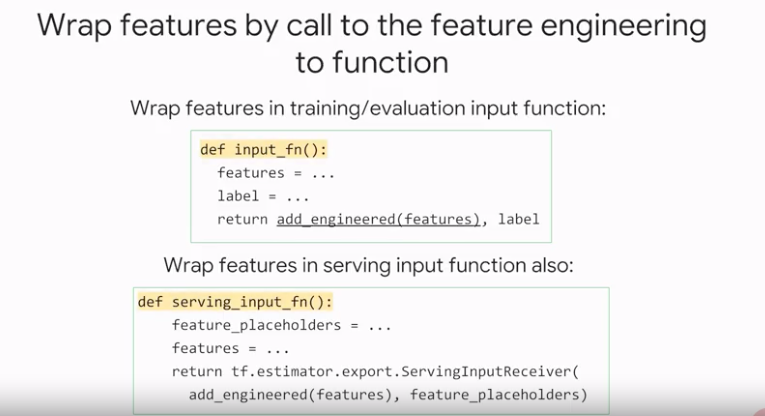


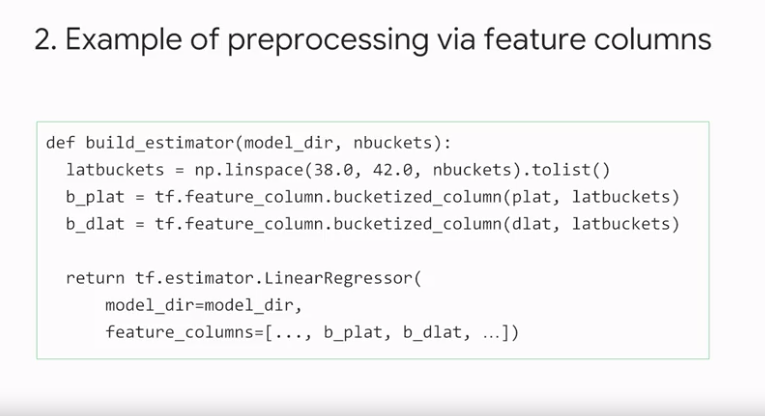


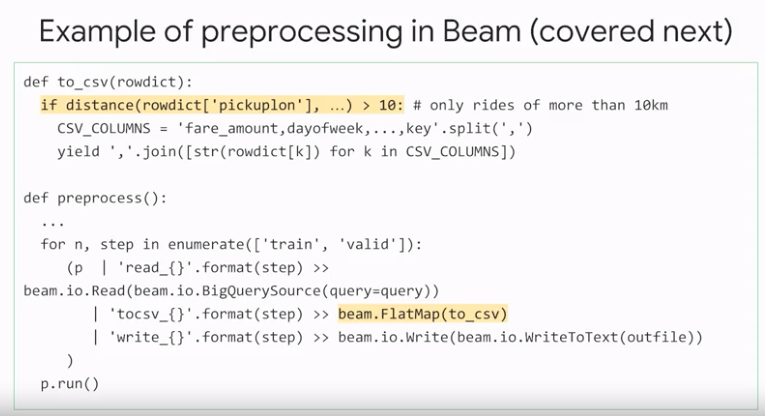
If you do decide to use sequel to pre-process training examples, it is absolutely critical that you take care to implement exactly the same preprocessing logic in TensorFlow.





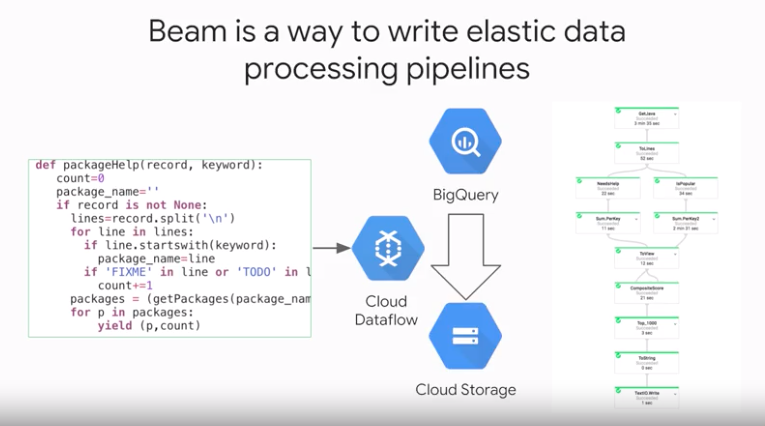


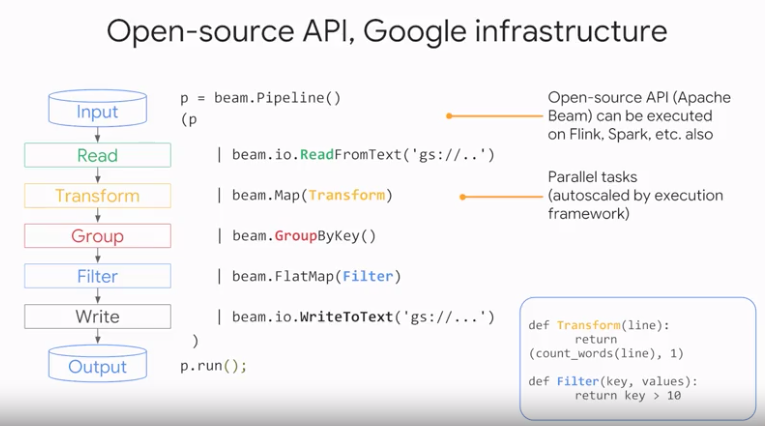




Google Cloud Dataflow:

Google Cloud Dataflow, which is a complimentary technology to Apache Beam. And both of them can help you build and run pre-processing and feature engineering.







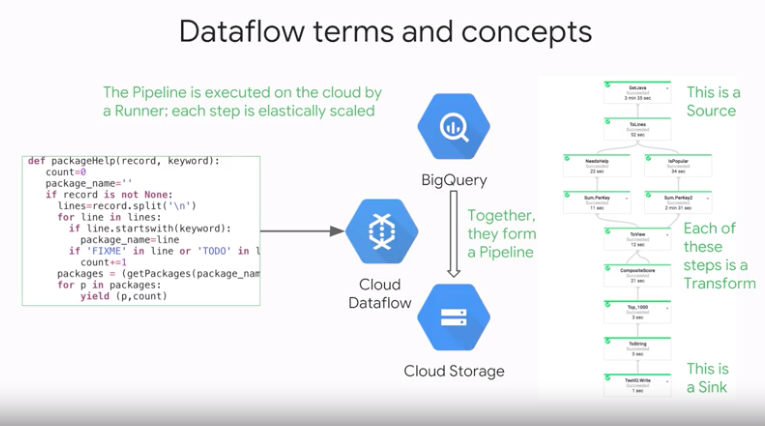
Source

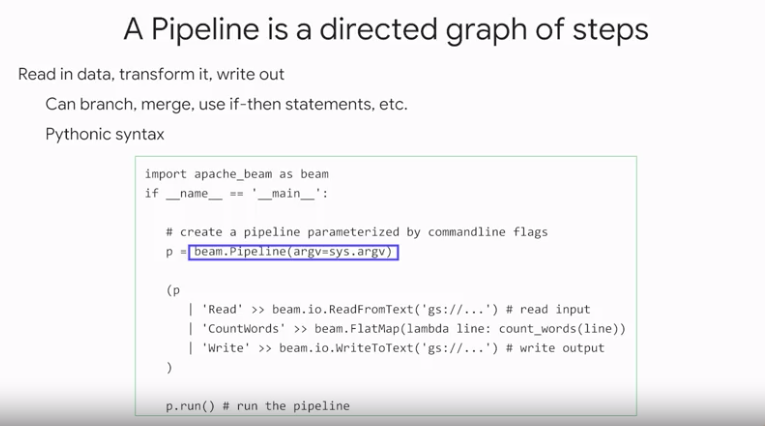
Source – Transform – Sink

Let's explore the Apache Beam pipelines in more detail. The pipeline must have a source, which is where the pipeline gets the input data.

The pipeline has a series of steps, each of the steps in Beam is called a transform.

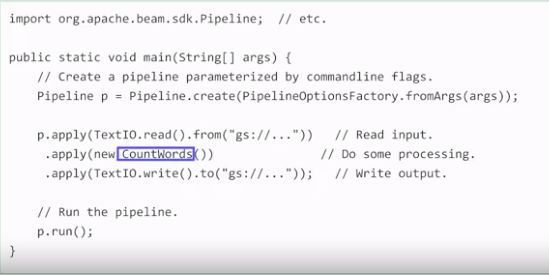
Each transform works on a data structure called PCollection. I'll return to a detailed explanation of PCollections shortly. For now, just remember that every transform gets a PCollection as input and outputs the result to another PCollection.





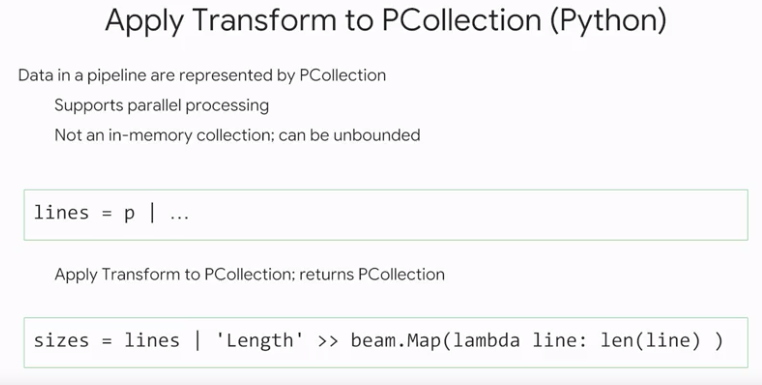
To run a pipeline, you need something called a runner. A runner takes the pipeline code and executes it. Runners are platform-specific, meaning that there's a data flow runner for executing a pipeline on Cloud Dataflow. There's another runner if you want to use Apache Spark to run your pipeline. There's also a direct router that will execute a pipeline on your local computer.

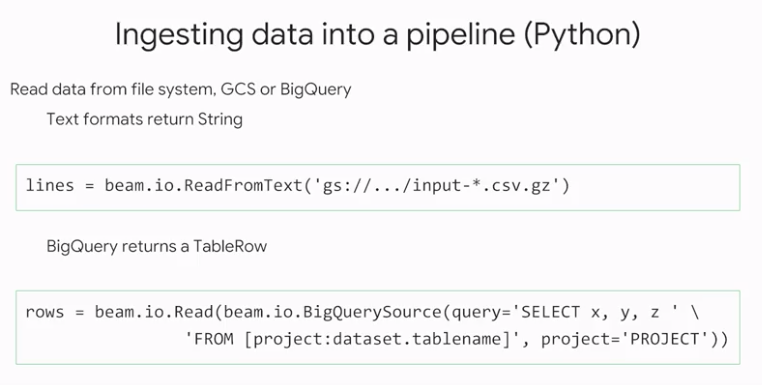
Every time you use the pipe operator, you provide a PCollection data structure as input and return a PCollection as output. An important thing to know about PCollections is that unlike many data structures, PCollection does not store all of its data in memory. Remember, the Dataflow is elastic and can use a cluster of servers through a pipeline. So PCollection is like a data structure with pointers to where the data flow cluster stores your data.

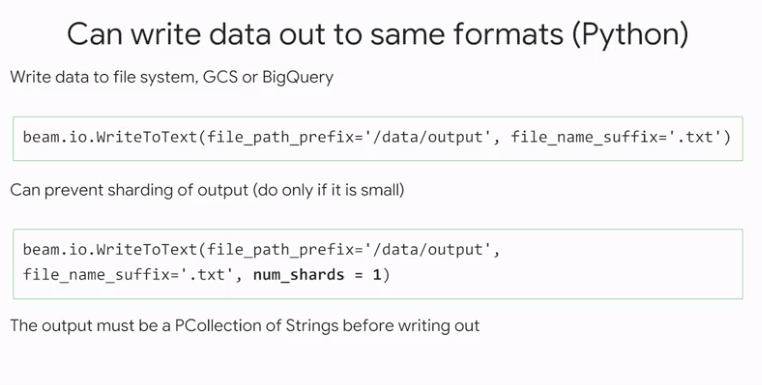


Let's say we have a PCollection of lines. For example, the lines could come from a file in Google Cloud storage. One way to implement the transformation is to take a PCollection of strings,

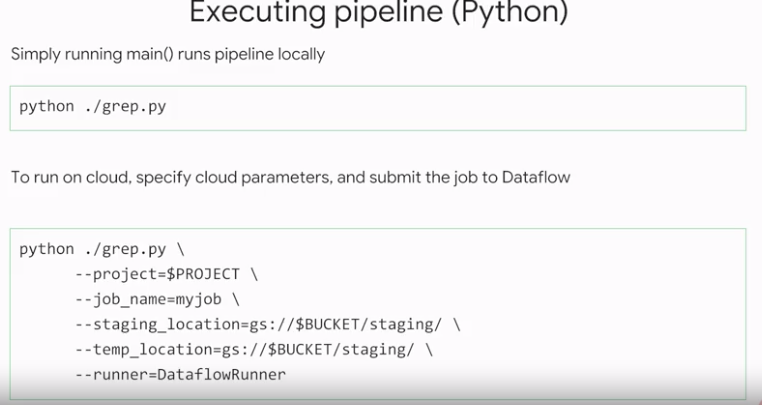
which are called lines in the code, and return a PCollection of integers.







An important thing to keep in mind when writing to a file system is that data flow can distribute execution of your pipeline across a cluster of servers. This means that there can be multiple servers trying to write results to the file system. In order to avoid contention issues where multiple servers are trying to get a file lock to the same file concurrently, by default, the text I/O connector will the output, writing the results across multiple files in the file system. For example, here, the pipeline is writing the result to a file with the prefix output in the data connector.

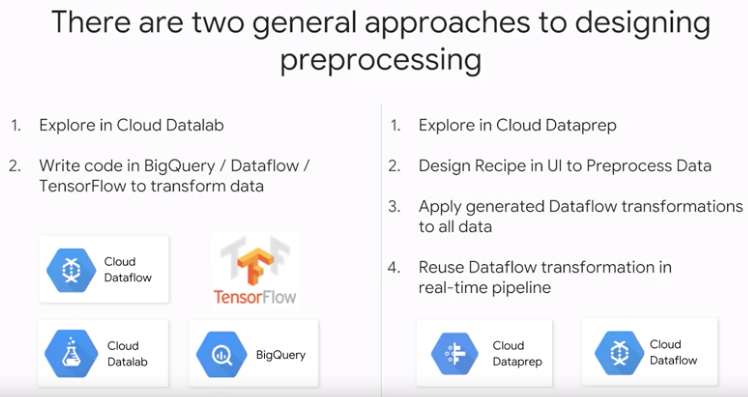


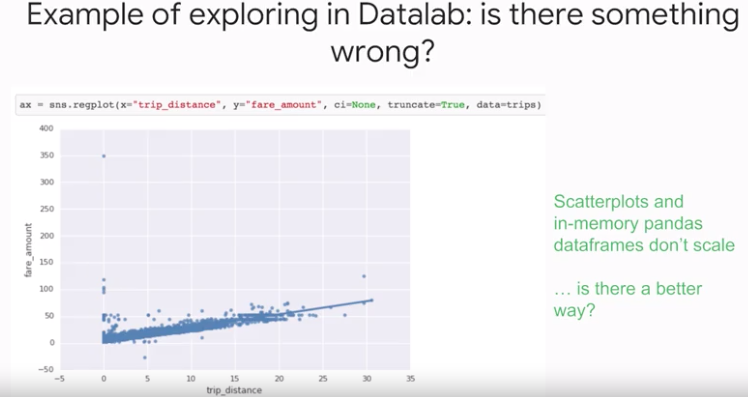
To submit the pipeline as a job to execute in Dataflow on GCP, you need to provide some additional information. You need to include arguments with the name of the GCP project, location in Google Cloud Storage Bucket where Dataflow will keep some staging and temporary data. And you also need to specify the name of the runner, which in this case is the DataFlowRunner.

Preprocessing with Dataprep:

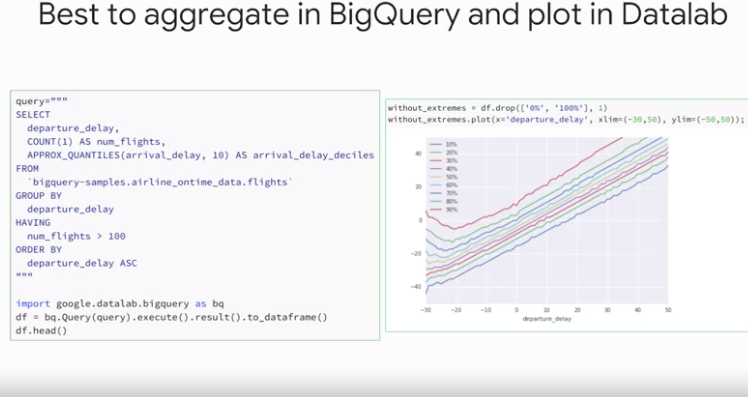
EDA:

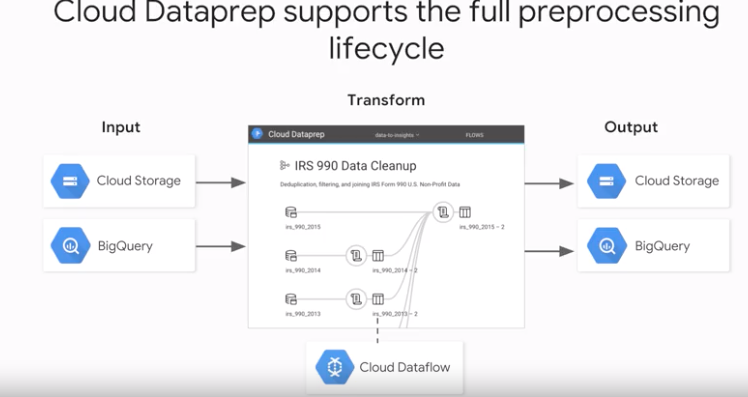




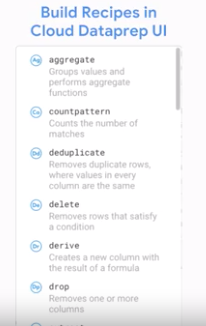


However, remember that the default Datalab environment is running in a single virtual server with a limited amount of memory. In case of the taxi fare dataset, there are billions of data points. So, it will be impractical or too expensive, to plot and analyze all of them using just a single no datalab environment. Instead of loading the billions of records of the entire taxi fare data set in the data lab environment, you can use SQL and calculate summary statistics using BigQuery.

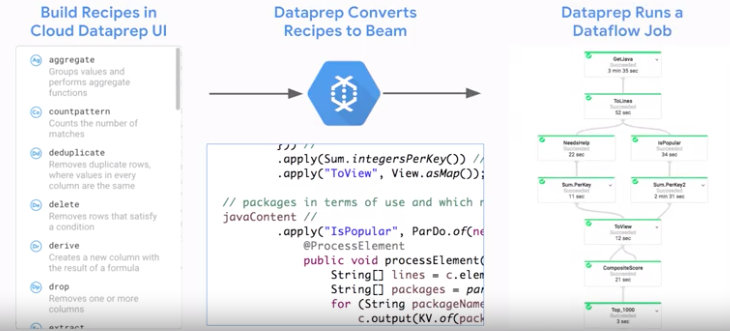




fter you have explored and understood your dataset, you can use Dataprep to compute flows of data transformations. The flows are similar to the pipelines that you have seen in dataflow. In fact, the flows are compatible with dataflow. You can take a Dataprep flow, and run it as a pipeline on the data flow platform. In Dataprep, the flows are implemented as a sequence of recipes, the recipes are data processing steps built from a library of so called wranglers. Dataprep has Wranglers for many common data processing tasks shown on the left.

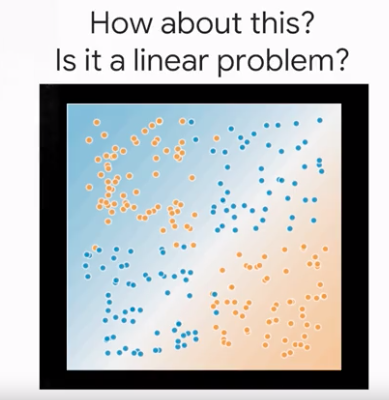


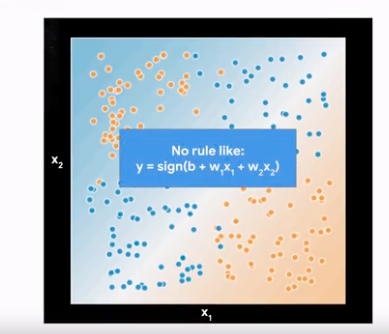
if you use the wranglers, Dataprep can take your flow and its recipes, and convert them to a dataflow pipeline.



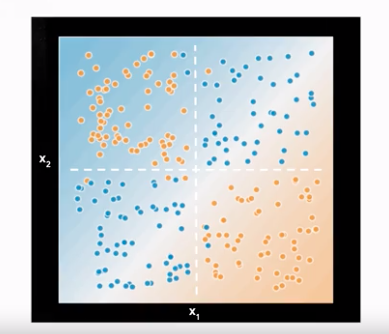
Feature Crosses:



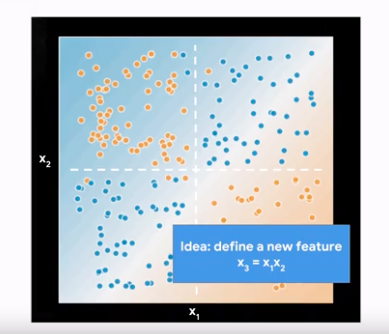


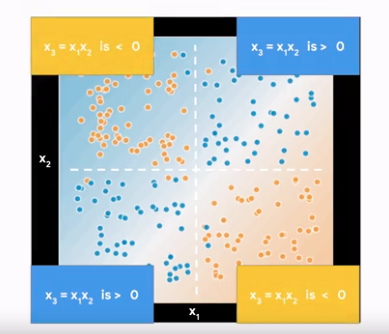


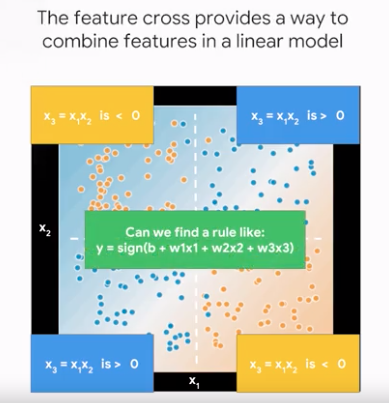
When we say we cannot use a linear model is that there is no way to linearly combine x1 and x2 to get a single decision boundary that would fit the data well.



For simplicity, let’s put that two axes in the center of the diagram so that the origin (0,0) is at the center of the diagram.



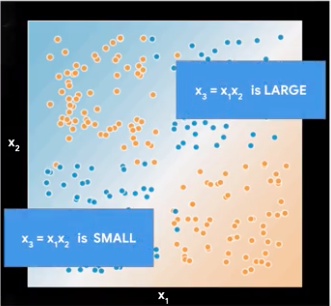


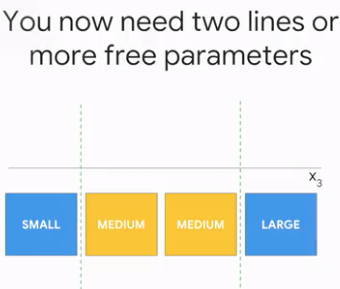


Can you see how the addition of x3 makes this solvable via a linear model? So, now we can find a rule such that the sine of x3 essentially gives us y. Of course that's just what we did. W1 and zero, w2 and zero, and w3 is one. Essentially, y is a sine of x3.

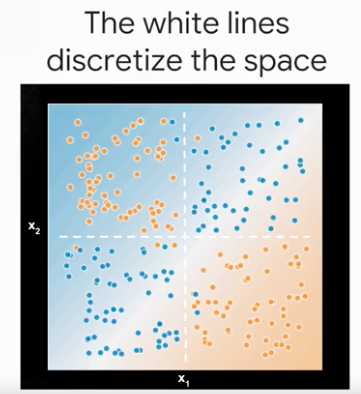
So, in traditional machine learning, feature crosses don't play much of a role, but that's because traditional ML methods were developed for relatively small datasets, and once you have datasets with hundreds of thousands to millions and billions of examples, feature crosses become an extremely useful technique to have in your tool chest.

Deep neural networks let you have many layers, and since each layer combines the layers before it, DNNs can model complex multidimensional spaces.

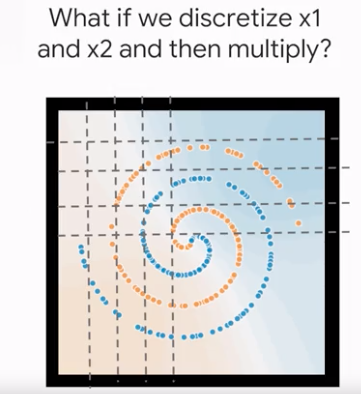




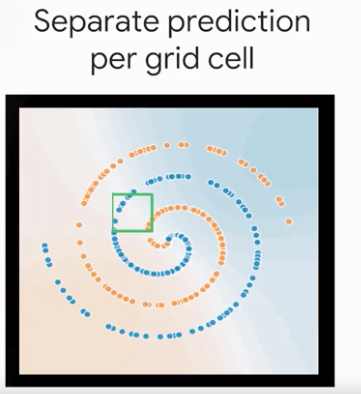
Just in values of X3 the feature cross, you have two linear separation boundaries. To make it just one, you need to translate x1 by one number and x2 by another number. This would be the parameters to the model, that it should learn.



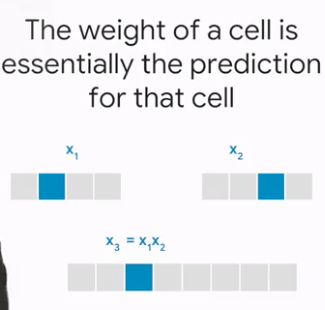
Two lines into four quadrants.



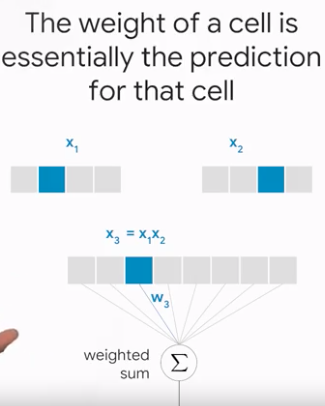
Now, let's consider what this looks like when we discretize x1 and x2 and then multiply.



Let's count the number of blue points and the number of yellow points and call it 85 percent blue. You see now how the probabilities are coming in.



When you one hot and cold the first set of values, and then you one hot and cold the second set of values, and then you feature cross them, you're essentially left with one node that fires for points that fall into that bucket. So think about it, the x3 will be one only if x1 equals one and x2 equals one. So for any point in the input space, only one bucket fires.



Now, if you take these feature crossed values and feed them into a linear regression, what does the wait w3 have to be? Yup, the ratio of blue dots to yellow dots in the grid cell corresponding to x1 and x2.

So that's why a feature cross is so powerful. You essentially discretize the input space and memorize the training data set. But can you see how this could be problematic? What if you don't have enough data?

1# Deployment Manager.