Machine Learning Engineer Learning Path

mardi 13 juin 2023 15:28

Google Cloud Big Data and Machine Learning Fundamentals

Big data and Machine learning on Google Cloud

Google Cloud infrastructure:
- based in five major **geographic locations: North America, South America, Europe, Asia, and Australia**
  - Having **multiple service locations** is important because choosing where to locate applications affects **qualities like availability, durability, and latency**, which measure travel from its source to its destination.
  - Each of these **locations** is divided into several different **regions** and **zones**.
    - Regions represent **independent geographic areas**, and are **composed of zones**.
    - A zone is an area where **Google Cloud resources are deployed**.
    - if a zone becomes unavailable, the resources won't be available either
  - Google Cloud lets users **specify the geographical locations to run services and resources**.
    - In many cases, you can even specify the location on a **zonal, regional, or multi-regional level**
    - This is useful for **bringing applications closer to users** around the world, and also for **protection** in case there are **issues with an entire region**, say, due to a **natu**
  - A few of Google Cloud's services support placing resources in what we call a **multi-region**.
    - For example, **Cloud Spanner multi-region configurations** allow you to replicate the database's data not just in multiple zones, but in **multiple zones across multi** configuration.
      - These **additional replicas** enable you to read data with **low latency** from multiple locations **close to or within the regions** in the configuration, like The Net
  - 103 zones in 34 regions
  - cloud.google.com/about/locations
- 3 layers: 1) Networking & security,2)compute, storage, 3) Big data ML products

Compute:
- compute power:
  - \- **Compute Engine**:
    - is an **IaaS** offering, or **infrastructure as a service**, which provides **compute, storage,and Network** resources **virtually** that are similar to **physical data centers**.
    - Compute Engine provides **maximum flexibility** for those who prefer to manage server instances themselves
  - \- Google **Kubernetes** Engine, or **GKE**:runs **containerized** applications in a **cloud environment**, as opposed to on an **individual virtual machine**, like Compute Engine. (co dependencies.)
  - \- **App** Engine,
    - a **fully managed PaaS** offering, or **platform as a service**:
    - PaaS offerings **bind code to libraries**.
  - **- Cloud Functions**
    - s often referred to as functions as a service.
    - executes code in response to events
    - serverless execution environment.
  - \- Cloud Run:
    - a **fully managed** compute platform that enables you to run request or event-driven **stateless** workloads without having to worry about servers.
    - It automatically scales up and down from zero
- Google Photos:
  - **automatic video stabilization**
    - **video** itself, which is really a large collection of individual images, along with **time series data** on the camera's **position** and **orientation** from the onboard **gyrosc**
    - : takes an unstable video,stabilizes it to minimize movement.
    - the hardware on a smartphone is not powerful enough to train sophisticated ML models.
      - That's why Google trains production machine learning models on a **vast network of data centers**, only to then **deploy smaller, trained versions** of the mod **hardware**
    - **Moore's Law**, with the required computing power used in the largest AI training runs **doubling every two years**

- **TPU**
  - introduced by Google **2016** to overcome CPU and GPU.
  - TPU is **application-specific integrated circuits (ASICs)**.
  - **domain-specific hardware (general-specific : GPU,CPU)**
  - TPUs are **generally faster** than current GPUs and CPUs for **AI applications and machine learning**.
  - They are also si**gnificantly more energy-efficient**.
  - **Cloud TPUs** have been integrated across **Google products**, making this **state-of-the-art hardware and supercomputing technology** available to Google Cloud custome

Storage
- Cloud (compute-storage decoupled) vs desktop (compute-storage coupled) computing
- fully managed database and storage services: Cloud Storage,Cloud Bigtable, Cloud SQL, Cloud Spanner, Firestore And BigQuery
- Goal=reduce the time and effort needed to store data (creating an elastic storage bucket directly in a web interface or through a command line for example on Cloud Storag
- Storage depends on data type, and business need.
- (un)Structured data = (non)tabular
  - Unstructured suited to Cloud Storage (also BigQuery),
- An object is an immutable piece of data consisting of a file of any format. You store objects in containers called buckets.All buckets are associated with a project, and you ca
- Each project, bucket, and object in Google Cloud is a resource in Google Cloud,
- Cloud Storage clases (suited for unstructured data):
  - Standard (best for frequently accessed, or hot data.stored for only brief periods of time)
  - Nearline (infrequently accessed data:data backups, long-tail multimedia content, or data archiving.)
  - Coldline (low-cost option for storing infrequently accessed data.once every 90 days)
  - Archive (lowest-cost option, used ideally for data archiving, online backup, and disaster recovery.access less than once a year,)
- Structured data:
  - transactional workloads (stem from Online Transaction Processing systems, which are used when fast data inserts and updates are required to build row-based record that impact only a few records.)

and analytical workloads (stem from Online Analytical Processing systems,which are used when entire datasets need to be read.require complex queries, for example,
Then determine wether data will be accessed using SQL or not:
- Transactional-SQL-local/regional scalability-Cloud SQL
- Transactional-SQL-global scalability-Cloud Spanner
- Transactional-NoSQL-FireStore
- Analytical-SQL-BigQuery (analyze petabyte-scale datasets)
- Analytical-NoSQL-Cloud BigTable (best for real-time, high-throughput applications that require only millisecond)

Big data:
- 2002: Google File System, GFS, was designed to handle data sharing and petabyte storage at scale.
- 2004: MapReduce
- 2008: Dremel: breaking the data into smaller chunks called shards, and then compressing them. then uses a query optimizer to share tasks between the many shards of data BigQuery.
- 2010: Colossus, in 2010, which is a cluster-level file system and successor to the Google File System. It is a Platform as a Service (PaaS) that supports querying using ANSI SQ
- 2012: Spanner, in 2012, which is a globally available and scalable relational database.
- 2015: Pub/Sub, in 2015, which is a service used for streaming analytics and data integration pipelines to ingest and distribute data.
- 2015:TensorFlow, also in 2015, which is a free and open source software library for machine learning and artificial intelligence.
- 2018: brought the release of the Tensor Processing Unit, or TPU,and AutoML, as a suite of machine learning products.
- 2021: Vertex AI, a unified ML platform
- Unified and stable platform: Cloud Storage Dataproc Cloud Bigtable BigQuery Dataflow Firestore Pub/Sub Looker Cloud Spanner AutoML, and Vertex AI,
- Product categories along the data-to-AI workflow:
  - ingestion & process:to digest both real-time and batch data (Pub/Sub, Dataflow (streaming data processing), Dataproc, Cloud Data Fusion)
  - storage: (Cloud Storage, Cloud SQL(relational), Cloud Spanner(relational), Cloud Bigtable(NoSQL), and Firestore(NoSQL))
  - analytics: (BigQuery:a fully managed data warehouse that can be used to analyze data through SQL commands; Looker, and Looker Studio)
  - ML: ML development platform and the AI solutions. (Vertex AI which includes the products and technologies: AutoML, Vertex AI Workbench, and TensorFlow)
    - AI solutions are built on the ML development platform and include state-of-the-art products to meet both horizontal and vertical market needs: Document AI, C Healthcare Data Engine

Example:
- A unicorn is a privately held startup business valued at over US$1 billion.
- For data latency: Dataflow (streaming data processing) + BigQuery (real-time business insights)
- For geospatial: Dataflow (a streaming event data pipeline).
  - driver locations ping Pub/Sub every 30 seconds, and
  - Dataflow would process the data (was able to automatically manage the number of workers processing the pipeline to meet demand)
  - The pipeline would aggregate the supply pings from the drivers against the booking requests.
  - This would connect to Gojek's notification system to alert drivers where they should go
- Conclusion:actively monitor requests to ensure that drivers are in the areas with the highest demand. This brings faster bookings for riders and more work for the drivers.

Data Engineering for streaming data

real-time data solution:
- Ingest streaming data using Pub/Sub (data ingestion is where large amounts of streaming data are received)
- Process the data with Dataflow, and
- Visualize the results with Looker and Looker Studio
- (In between data processing with Dataflow and visualization with Looker or Looker Studio, the data is normally saved and analyzed in a data warehouse such as BigQuery)
- design streaming pipelines with Apache Beam

streaming vs batch processing (media streaming vs downloading):
- Batch processing is when the processing and analysis happens on a set of stored data: Payroll and billing systems
- Streaming data is a flow of data records generated by various data sources: fraud detection or intrusion detection. data is analyzed in near real-time

4Vs: data engineers and data scientists four major challenges = variety, volume, velocity, and veracity:
- data could come in from a variety of different sources and in various formats
- Veracity: data quality (Due to several data types and sources, big data often has many data dimensions)

IoT devices challenge:
- data can be streamed from many different methods and devices
- it can be hard to distribute event messages (notification) to the right subscribers
- data can arrive quickly and at high volumes.

- ensuring services are reliable, secure, and perform as expected
- Pub/Sub:
  - a tool to handle distributed message-oriented architectures at scale
  - (Publisher/Subscriber, or publish messages to subscribers).
  - distributed messaging service
  - Pub/Sub's APIs are open, the service is global by default, and it offers end-to-end encryption
  - end-to-end encryption:
    - Pub/Sub reads, stores, broadcasts to any subscribers of this data topic that new messages are available
  - A central element of Pub/Sub is the topic.
- Designing streaming pipelines with Apache Beam:
  - streaming input sources + pipe data into a data warehouse (dataflow)
  - "Process" : extract, transform, and load data, or ETL.
  - pipeline design phase questions:
    - Will the pipeline code be compatible with both batch and streaming data? Or need to be refactored?
    - Will the pipeline code software development kit, or SDK, being used have all the transformations, mid-flight aggregations and windowing and be able to handle late data?
    - Are there existing templates or solutions that should be referenced?
  - pipeline design: Apache Beam: an open source, unified programming model to define and **execute data processing pipelines**, including ETL, batch, and stream processing
    - unified, which means it uses a single programming model for both batch and streaming data
    - It's portable, which means it can work on multiple execution environments, like Dataflow and Apache Spark
    - extensible, which means it allows you to write and share your own connectors and transformation libraries.
    - Apache Beam provides pipeline templates (Java, Python, or Go)
- Implementing streaming pipelines on Cloud Dataflow:
  - Questions:
    - How much maintenance overhead is involved?
    - Is the infrastructure reliable?
    - How is the pipeline scaling handled?
    - How can the pipeline be monitored?
    - Is the pipeline locked in to a specific service provider?
  - Dataflow is a fully managed service for executing Apache Beam pipelines within the Google Cloud
  - Dataflow is serverless and NoOps
    - NoOps: doesn't require management from an operations team, because maintenance, monitoring, and scaling are automated.
    - Serverless computing is a cloud computing execution model
  - Dataflow tasks :
    - optimizing a pipeline model's execution graph
    - schedules out distributed work
    - scaler
    - auto-heals any worker faults
    - rebalances efforts to most efficiently use its workers.
    - execution engine to process and implement data processing pipelines
  - you don't need to monitor all of the compute and storage resources that Dataflow manages,
  - Dataflow templates:
    - streaming templates (for processing continuous, or real-time, data): Pub/Sub to BigQuery, Pub/Sub to Cloud Storage, Datastream to BigQuery, Pub/Sub to M
    - batch templates (for processing bulk data, or batch load data.): BigQuery to Cloud Storage, Bigtable to Cloud Storage, Cloud Storage to BigQuery, Cloud Spar
    - utility templates (activities related to bulk compression, deletion, and conversion)

- Visualization with Looker:
  - Looker: semantic modeling layer on top of databases using Looker Modeling Language, or LookML
  - LookML defines logic and permissions independent from a specific database or a SQL language,
  - The Looker platform is 100% web-based, which makes it easy to integrate into existing workflows and share with multiple teams at an organization
  - Looker API, which can be used to embed Looker reports in other applications.
  - Looker features:
    - Dashboards: schedule its delivery through storage services like: Google Drive, Slack, Dropbox.
- Visualization with Data Studio:
  - Looker Studio: integrated into BigQuery: doesn\u2019t require support from an administrator to establish a data connection, which is a requirement with Looker
  - integrated into Google Analytics, Google Cloud billing dashboard
  - to create a Looker Studio dashboard:
    - choose a template (a pre-built template or a blank report)
    - link the dashboard to a data source.
- Lab introduction: Creating a streaming data pipeline for a Real-Time dashboard with Dataflow:
  - **gcloud** is the command-line tool for Google Cloud.
  - pre-installed on Cloud Shell and supports tab-completion
  - list the active account name: gcloud auth list
  - list the project ID: gcloud config list project
  - to create dataset: Google Cloud Shell or the Google Cloud Console.
  - [Pub/Sub]() is an asynchronous global messaging service.
    - decoupling senders and receivers, it allows for secure and highly available communication between independently written applications
    - delivers low-latency, durable messaging
    - publisher applications and subscriber applications connect with one another through the use of a shared string called a **topic**
    - Google maintains a few public Pub/Sub streaming data topics
  - [BigQuery]() is a serverless data warehouse
    - Tables in BigQuery are organized into datasets.
    - to create a new BigQuery dataset:
      - command-line tool (**Cloud Shell**):
        - Create dataset: bq --location=us-west1 mk taxirides
        - Create Table: bq --location=us-west1 mk \
        - --time_partitioning_field timestamp \
        - --schema ride_id:string,point_idx:integer,latitude:float,longitude:float,\
        - timestamp:timestamp,meter_reading:float,meter_increment:float,ride_status:string,\

passenger_count:integer -t taxirides.realtime
BigQuery Console UI:
Create a Cloud Storage bucket (to provide working space for your Dataflow pipeline)
[Cloud Storage]() allows world-wide storage and retrieval of any amount of data at any time
Applications: serving website content, storing data for archival and disaster recovery, or distributing large data objects to users via direct download.
Set up a Dataflow Pipeline
set up a streaming data pipeline to read sensor data from Pub/Sub, compute the maximum temperature within a time window, and write this out to B
Restart the connection to the Dataflow API.
Create a new streaming pipeline
Analyze the taxi data using BigQuery:
SELECT * FROM taxirides.realtime LIMIT 10
Perform aggregations on the stream for reporting
Stop the Dataflow Job (to free up resources for your project)
Create a real-time dashboard
Create a time series dashboard
Summary:
used Pub/Sub to collect streaming data messages from taxis and feed it through your Dataflow pipeline into BigQuery
Pub/Sub, which can be used to ingest a large volume of IoT data from diverse resources in various formats.
Dataflow, a serverless, NoOps service, to process the data. 'Process' here refers to ETL (extract, transform, and load).

Big data with BigQuery:
BigQuery's two main services, storage and analytics
BigQuery is a fully managed data warehouse
A data warehouse is a large store, containing terabytes and petabytes of data gathered from a wide range of sources within an organization, that's used to **g**
A **data lake** is just a pool of **raw, unorganized, and unclassified** data, which has **no specified purpose**. A data warehouse on the other hand, contains structu
used for **advanced querying**
Being **fully managed** means that BigQuery **takes care of the underlying infrastructure**, so you can focus on using SQL queries to answer business questions, scalability, and security.
BigQuery Storage: to store petabytes of data. **1 petabyte** is equivalent to **11,000 movies at 4k** quality.
BigQuery analytics: **machine learning (**using SQL**), geospatial analysis, and business intelligence**,
BigQuery is a fully managed serverless solution, meaning that you don't need to worry about provisioning any resources or managing servers in the backend to answer your organization's questions in the frontend.
By **encryption at rest**, we mean encryption used to protect data that is stored on a **disk**, including solid-state drives, or backup media.
data warehouse solution architecture:
input data can be either real-time or batch data
streaming data, which can be either structured or unstructured, high speed, and large volume, Pub/Sub is needed to digest the data.
If it's batch data, it can be directly uploaded to Cloud Storage.
both pipelines (real-time or batch data) lead to Dataflow to process the data (ETL)
(real-time+Pub/Sub)(batch data+Cloud Storage)+Dataflow (ETL)+BigQuery(analytics,AI, and ML)+(business intelligence(BI) tools)(AI/ML tools)
The job of the analytics engine of BigQuery at the end of a data pipeline is to ingest all the processed data after ETL, store and analyze it, and possibly c visualization and machine learning.
business analyst, BI developers: If you prefer to work in spreadsheets, you can query both small or large BigQuery datasets directly from **Google Sheet** operations like pivot tables.
data scientist or machine learning engineer: you can directly call the data from BigQuery through **AutoML** or **Workbench (**parts of Vertex AI**)**
BigQuery is like a common staging area for data analytics workloads

Storage and analytics
BigQuery :**fully managed storage facility to load and store datasets**, and also a **fast SQL-based analytical engine**
The two services are connected by **Google's high-speed internal network**.
BigQuery can ingest datasets from : **internal** data(data saved directly in BigQuery), **external** data, **multi-cloud** data (data stored in **multiple cloud services**, s datasets.
After the data is stored in BigQuery, it's fully managed by BigQuery and is **automatically replicated**, **backed up**, and **set up to autoscale**
BigQuery also offers the option to **query external data sources**–like data stored in other **Google Cloud storage** services like **Cloud Storage**, or in other **Googl**
Spanner or Cloud SQL
That means a raw CSV file in Cloud Storage or a Google Sheet can be used to write a query **without being ingested by BigQuery first**
inconsistency might result from **saving and processing data separately**. To avoid that risk, consider using **Dataflow** to build a **streaming data pipeline into** I
patterns to load data into BigQuery:
batch load, where source data is loaded into a BigQuery table in a single batch operation. (**one-time** operation or it can be **automated to occur on a sc**
A batch load operation can create a **new** table or **append** data into an existing table.
streaming, where **smaller batches of data are streamed continuously** so that the data is available for **querying in near-real time**
generated data, where SQL statements are used to insert rows into an existing table or to write the results of a query to a table.
BigQuery is optimized for running analytic queries over large datasets.
It can perform queries on **terabytes of data in seconds and on petabytes in minutes.**
BigQuery analytics features:
Ad hoc analysis using Standard SQL, the **BigQuery SQL** dialect.
Geospatial analytics using geography data types and Standard SQL geography functions.
Building **machine learning models** using **BigQuery ML**.
Building rich, interactive **business intelligence dashboards** using **BigQuery BI Engine**.
Query types**:**
BigQuery runs **interactive** queries, which means that the **queries are executed as needed**
batch queries, where **each query is queued** on your behalf and the query starts **when idle resources are available**, usually within a few minutes.

BigQuery demo - San Francisco bike share
- If no **hyphen** in the project name, no need to **back ticks** around the project name
- hold down the command key or the **Windows** key it'll highlight all the data sets in your query, so you can click on it
- Click on Query Table. Click on field names from Schema to add them into Query editor (no need to write names)
- More>Format
- Run

SELECT, FROM, AS, GROUP BY, ORDER BY, WHERE, DESC, LIMIT
Explore In Data Studio

data definition language (DDL): I want a creation statement inside of the actual code itself
Create dataset
CREATE OR REPLACE **TABLE** <dataset name>.<table name> AS
CREATE OR REPLACE **VIEW** <dataset name>.<view name> AS : If the original public dataset updates, the derived table will update

Introduction to BigQuery ML:

Traditional:

export data from **datastore** into an **IDE** (integrated development environment) such as **Jupyter Notebook or Google Colab**
transform the data and perform all your **feature engineering** steps **before you can feed it into a training mode**l.
build the model in TensorFlow, or a similar library, and train it **locally** on your computer or on a **virtual machine**.
To **improve the model performance**, you also need to go back and forth to **get more data and create new features**.

BigQuery:

- **Create a model** with a SQL statement
- Write a **SQL prediction query** and invoke ml.Predict
- evaluating the model,

Hyperparameters are the settings applied to a model **before the training starts**, like the learning rate
Choosing which type of ML model depends on your **business goal and the datasets**.
Supervised models are **task-driven and identify a goal**.
Goal: classification: logistic regression
goal :predict a number (regression): linear regression
unsupervised models are **data-driven and identify a pattern**
goal: identify patterns or clusters: cluster analysis
We recommend that you start with these options (logistic/linear regression), and use the results to **benchmark** to compare against more comple
networks)(XGBoost,AutoML Tables, wide and deep DNNs), which may take more time and computing resources to train and deploy
machine learning operations (**ML Ops**): BigQuery ML supports features to **deploy, monitor, and manage the ML production**
Importing TensorFlow models for **batch prediction**
Exporting models from BigQuery ML for **online prediction**
hyperparameter tuning using **Vertex AI Vizier**
ML development: upload data, engineer feature, train model, evaluate result

Using BigQuery ML to predict customer lifetime value (LTV):

LTV: to estimate how much revenue or profit you can expect from a customer given their history and customers with similar patterns.
a **record or row** in the dataset is called an **example, an observation, or an instance**.
A **label** is a correct answer, and you know it's correct because it comes from historical data.
Depending on what you want to predict, a label can be either a **numeric** variable, which requires a **linear regression** model, or a **categorical** variable, w
model.
Those columns are called **features**, or at least potential features.
Understanding the quality of the data in each column and working with teams to **get more features or more history** is often the hardest part of any ML pro
feature engineering: **combine or transform feature** columns
BigQuery ML **automatically does one-hot encoding** for categorical values.
BigQuery ML **automatically splits the dataset into training data and evaluation data**

BigQuery ML project phases:

phase 1: you **extract, transform, and load** data into BigQuery, if it isn't there already.
If you're already using other Google products, like YouTube for example, look out for easy **connectors** to get that data into BigQuery before you build y
You can **enrich your existing data warehouse with other data sources** by using **SQL joins**.
phase 2: you **select** and **preprocess** features.
You can use **SQL** to create the training dataset for the model to learn from
phase 3: you **create the model inside BigQuery**.
This is done by using the "**CREATE MODEL**" command.
Give it a **name**, specify the **model type**, and pass in a SQL query with your training dataset.
phase 4: after your model is trained, you can execute an **ML.EVALUATE** query to evaluate the performance of the trained model on your evaluation dataset.
phase 5: use it to **make predictions**.
invoke the **ml.PREDICT** command on your newly trained model to return with predictions and the model's confidence in those predictions.

BigQuery ML key commands:

CREATE MODEL: create a model. Models have OPTIONS (model type), which you can specify.
CREATE OR REPLACE MODEL: to overwrite an existing model
ML.WEIGHTS: inspect what a model learned
That value indicates **how important the feature is for predicting the result**, or label.
ML.EVALUATE: To evaluate the model's performance
ML.PREDICT: to make batch predictions
BigQuery ML commands for supervised models:
Labels: you need a field in your training dataset titled **LABEL**, or you need to specify which field or fields are your labels using the **input_label_cols** in yo
Feature: features are the data columns that are part of your **SELECT** statement after your CREATE MODEL statement.
After a model is trained, you can use the **ML.FEATURE_INFO** command to get **statistics and metrics** about that column for additional analysis.
model object: created in BigQuery that **resides in your BigQuery dataset**. You train many different models, which will all be objects stored under your
tables and views. Model objects can display information for **when it was last updated** or **how many training runs** it completed
Model types (regression/classification): Creating a new model is as easy as writing CREATE MODEL, choosing a type, and passing in a training dataset.
Training progress: While the model is running, and even after it's complete, you can view training progress with **ML.TRAINING_INFO**.
inspect weights (ML.WEIGHTS): to see what the model learned about the importance of each feature as it relates to the label you're predicting.
ML.EVALUATE: how well the model performed against its evaluation dataset.
ML.PREDICT: getting predictions (through referencing your model name and prediction dataset)

Predicting Visitor Purchases with a Classification Model with BigQuery ML:

BQML: a feature in BigQuery where data analysts can **create, train, evaluate, and predict** with machine learning models with **minimal coding**

Create a BigQuery dataset to store models

the web analytics dataset has nested and repeated fields like [ARRAYS] which **need to be broken apart into separate rows** in your dataset. This is accomplish

Machine learning options on google cloud:

Smart Reply: Gmail automatically suggests three responses to a received message. it uses artificial intelligence (natural language processing) to predict how you m
The goal is to enable every company to be an AI company by **reducing the challenges of AI model creation** to only the steps that require **human judgment or crea**
Options to build ML models:

a. **BigQuery ML**: uses SQL queries to create and execute machine learning models in BigQuery
b. **pre-built APIs**: lets you **leverage machine learning models that have already been built and trained by Google**, so you don't have to build your own machir enough training data or sufficient machine learning expertise in-house.
c. **AutoML**: is a **no-code solution**, so you can **build your own machine learning models** on **Vertex AI** through a **point-and-click interface.**
d. **custom training:** through which you can **code your very own** machine learning **environment, the training, and the deployment**, which gives you flexibility a pipeline.

Differences between options:

Data type: BigQuery ML only supports tabular data while the other three support tabular, image, text, and video

Training data size: Pre-built APIs do not require any training data, while BigQuery ML and custom training require a large amount of data

Machine learning and coding expertise: Pre-Built APIs and AutoML are user friendly with low requirements, while Custom training has the highest req you to understand SQL.

Flexibility to tune the hyperparameters: At the moment, you can't tune the hyperparameters with Pre-built APIs or AutoML, however, you can experi BigQueryML and custom training.

Time to train the model: Pre-built APIs require no time to train a model because they directly use pre-built models from Google. The time to train a m depends on the specific project.

Normally, custom training takes the longest time because it builds the ML model from scratch, unlike AutoML and BigQuery ML.

Which option:

- **BigQuery ML:** data engineers, data scientists, and data analysts are familiar with SQL and already have your data in BigQuery: **BigQuery ML** lets you de
- pre-built APIs: business users or developers with little ML experience: pre-built APIs (vision, video, and natural language)
- AutoML: If your developers and data scientists want to **build custom models** with your **own training data** while spending **minimal time coding**, then A provides a code-less solution to enable you to focus on business problems instead of the underlying model architecture and ML provisioning.
- custom training: If your ML engineers and data scientists want **full control of ML workflow**, **Vertex AI** custom training lets you train and serve custom **Workbench**.

Pre-built APIs:

Good Machine Learning models require **lots of high-quality training data**. You should aim for **hundreds of thousands** of records to train a custom model. If built APIs are a great place to start.

Pre-built APIs are offered as services.

They save the time and effort of **building, curating, and training** a new dataset so you can just jump right ahead to predictions

API examples:

The Speech-to-Text API: converts audio to text for data processing.

The Cloud Natural Language API: recognizes **parts of speech** called entities and sentiment.

The Cloud Translation API: converts text from one language to another.

The Text-to-Speech API: converts text into high quality voice audio.

The Vision API: works with and recognizes **content** in static images.

cloud.google.com/vision

The Video Intelligence API: recognizes **motion and action** in video.

You can actually **experiment** with each of the ML APIs in a **browser**.

When you're ready to **build a production model**, you'll need to **pass a JSON object request to the API and parse what it returns**.

AutoML:

training and deploying ML models can be extremely time consuming, because to achieve the best result you need to

repeatedly add new data and features,

try different models, and

tune parameters .

AutoML was first announced in January of 2018, the goal was to **automate machine learning pipelines** to **save data scientists from manual work**

For AutoML, two technologies are vital:

- **transfer learning**. is a powerful technique that lets people with **smaller datasets, or less computational power**, achieve state-of-the-art results by takir that have been trained on **similar, larger data sets**.

  Because the model learns via transfer learning, **it doesn't have to learn from scratch**, so it can generally reach higher accuracy with much less da that don't use transfer learning.
- **neural architecture search**. to find the **optimal model** for the relevant project.

AutoML platform actually trains and evaluates **multiple models** and compares them to each other

This neural architecture search produces an **ensemble of ML models** and chooses the best one.

It is no-code solution: minimal effort and requires little machine learning expertise

a tool to **quickly prototype models** and explore new datasets before investing in development.

How AutoML works:

For each data type (image, tabular, text, and video), AutoML solves different types of problems, called **objectives**.

upload your data (from Cloud Storage, BigQuery, or even your local machine) into AutoML.

inform AutoML of the problems you want to solve.

Some problems may sound similar to those mentioned in pre-built APIs.However the major difference is that pre-built APIs use **pre-built machin custom-built models**.

In AutoML, you use your **own data** to train the machine learning mode

Objectives:

- For image data:
  - You can use a **classification model** to analyze image data and return a list of **content categories** that apply to the image.
  - You can also use an **object detection** model to analyze your image data and return **annotations** that consist of a **label** and **bounding b** an image.
- For tabular data:
  - You can use a **regression model** to analyze tabular data and return a numeric value.
  - You can use a **classification model** to analyze tabular data and return a list of categories.
  - And a **forecasting model** can use multiple rows of **time-dependent** tabular data from the past to predict a series of numeric values in
- For text data:
  - You can use a **classification model** to analyze text data and return a list of categories that apply to the text found in the data.
  - An **entity extraction model** can be used to inspect text data for known entities referenced in the data and label those entities in the te **hashtag**, but created by machine).
  - a **sentiment analysis model** can be used to inspect text data and identify the prevailing emotional opinion within it, especially to deter negative, or neutral
- for video data:
  - You can use a **classification model** to analyze video data and return a list of c**ategorized shots and segments**.

- You can use a classification model to analyze video data and return a list of categorized shots and segments.
- You can use an **object tracking model** to analyze video data and return a list of **shots and segments where these objects were detect**
- And an **action recognition model** can be used to analyze video data and return a **list of categorized actions with the moments the act**

Custom training
Using **Vertex AI Workbench**
Workbench is a **single development environment for the entire data science workflow**, from exploring, to training, and then deploying a machine lea
environment you want your ML training code to use: a **pre-built container** (includes dependencies and libraries) **or a custom container (empty: no libraries**

Vertex AI
Some traditional challenges
determining how to handle **large quantities of data**,
Determining the **right machine learning model** to train the data,
harnessing the required amount of **computing power**.
Production challenges:
scalability,
monitoring,
continuous integration and continuous delivery or deployment
ease-of-use challenges.
Many tools on the market require advanced coding skills,
which can take a data scientist's focus away from model configuration.
no unified workflow.
Google's solution to the above challenges: Vertex AI: a **unified platform** that brings all the components of the **machine learning ecosystem and workflow** to
Vertex AI, a tool that **combines the functionality of AutoML**, which is codeless, and **custom training**, which is code-based, **to solve production and ease-of-**
unified platform: one digital experience to **create, deploy, and manage** models **over time, and at scale:**

1. **data readiness** stage: users can **upload** data from wherever it's stored– **Cloud Storage, BigQuery, or a local machine**.
2. during the **feature readiness** stage: users can **create features**, which are the **processed data** that will be put into the model, and then share them
3. **Training and Hyperparameter tuning**. This means that when the data is ready, users can **experiment** with different models and **adjust** hyperpara
4. **deployment and model monitoring:** users can set up the pipeline to **transform the model into production** by **automatically monitoring** and perf

Vertex AI allows users to **build** machine learning models with either **AutoML**, or **Custom Training**
AutoML is **easy to use** and lets data scientists spend more time **turning business problems into ML solutions**, while custom training enables data scien
development environment and process.
Vertex AI properties:
seamless: Vertex AI provides a **smooth** user experience from **uploading and preparing data all the way to model training and production**.
Scalable: The Machine Learning Operations (**MLOps**) provided by Vertex AI helps to **monitor and manage** the ML **production** and therefore **scale the s**
automatically.
sustainable. All of the artifacts and features created using Vertex AI can be **reused and shared**.
speedy: Vertex AI produces models that have **80% fewer lines of code** than competitors.

AI Solutions:
Google Cloud's artificial intelligence solution portfolio (three layers):
the AI foundation: Google Cloud infrastructure (compute, storage, networking) and data (BQ, Dataflow, Looker).
the AI development platform: AutoML and custom training, which are offered through Vertex AI, and pre-built APIs and BigQuery ML.
AI solutions:
horizontal solutions:
usually apply to any industry that would like to solve the same problem.
Document AI and **CCAI**
Document AI uses **computer vision** and **optical character recognition**, along with **natural language processing**, to create pretrained models
documents
The goal is to **increase the speed and accuracy** of document processing to help organizations make better decisions faster, while **reducing c**
Contact Center AI, or CCAI. The goal of CCAI is to improve **customer service** in contact centers through the use of artificial intelligence.
It can help **automate simple interactions, assist human agents, unlock caller insights, and provide information to answer customer questi**
vertical/industry solutions:
relevant to specific industries.
Retail Product Discovery, which gives retailers the ability to provide **Google-quality search and recommendations on their own digital pro**
conversions and reduce search abandonment
Halthcare Data Engine, which generates **healthcare insights and analytics** with one **end-to-end solution**
Lending DocAI, which aims to transform the **home loan** experience for borrowers and lenders by **automating mortgage document processi**
cloud.google.com/solutions/ai
Summary

The machine learning **Workflow** with **Vertex AI:**
Introduction:
basic differences between machine learning and traditional programmin:
Traditional: Data, plus rules–otherwise known as algorithms–lead to answers: **Data+rules(algorithms)=answers**
computer can only follow the algorithms that a human has set up
ML: the algorithms are too complex to figure out
we feed a machine a **large amount of data, along with answers** that we would expect a model to conclude from that data
For machine learning to be successful:
lots of storage, like what's available with **Cloud Storage**,
the ability to make **fast calculations**, like with **cloud computing**.
key stages to this learning process:

- data preparation: data uploading and feature engineering
  Data types:
  real-time streaming data or batch data,
  structured (numbers and text normally saved in tables), or unstructured (images and videos).
- model training: A model needs a tremendous amount of **iterative** training. This is when **training and evaluation** form a cycle to train a model, then eva
  model some more.
- model serving. move an ML model into production. the machine learning model is **deployed, monitored, and managed**.

ML workflow isn't linear, it's iterative:
during model training, you may need to **return** to dig into the raw data and **generate more useful features** to feed the model.

When **monitoring** the model during **model serving**, you might find **data drifting**, or the accuracy of your prediction might suddenly drop. You might ne
adjust the model parameters.
Fortunately, these steps can be **automated with machine learning operations, or MLOps**.
Vertex AI provides many features to support the ML workflow:
- Feature Store: provides a **centralized repository for organizing, storing, and serving** features to feed to training models,
- **Vizier**: helps you **tune hyperparameters** in complex machine learning models,
- Explainable AI: helps **interpret training performance and model behaviors**,
- Pipelines: help you **automate and monitor the ML production**

Data preparation:
AutoML workflow
upload data, feature engineering
When you upload a dataset in the **Vertex AI user interface**, you'll need to provide a **meaningful name** for the data and then select the **data type (**image, tab
tagged—or labeled
A label can be **manually** added, or it can be added by using **Google's paid label service via the Vertex console**.
These **human labellers** will manually generate accurate labels for you.
Data can be uploaded from a local source, BigQuery, or Cloud Storage.
After your data is uploaded to AutoML, the next step is preparing the data for model training with feature engineering.
feature
a factor that contributes to the prediction.
an independent variable in statistics
a column in a table.
Preparing features can be both challenging and tedious.To help, Vertex AI has a function called F**eature Store**.
F**eature Store: centralized repository** to **organize, store, and serve** machine learning features.
Feature Store **aggregates all the different features from different sources** and updates them to make them available from a central repository
Vertex AI automates the feature aggregation to scale the process.
benefits of Vertex AI Feature Store:
features are **shareable** for training or serving tasks.
Features are **managed and served from a central repository**, which helps **maintain consistency** across your organization.
features are **reusable**.This helps save time and reduces duplicative efforts, especially for high-value features.
features are **scalable**.Features automatically scale to provide **low-latency serving**, so you can focus on **developing the logic to create the feature**
deployment.
features are **easy to use**.

Model training:
model training, and model evaluation. This process might be iterative.
Artificial intelligence (AI):**anything related to computers mimicking human intelligence**.
Machine learning is a subset of AI that mainly refers to supervised and unsupervised learning.
deep learning, or deep neural networks: a subset of ML that **adds layers in between input data and output** results to make a machine learn at more depth
Supervised learning is **task-driven and identifies a goal**.
classification: predicts a categorical variable
regression model: predicts a continuous number,
Unsupervised learning, however, is **data-driven and identifies a pattern:**
clustering: groups together data points with similar characteristics and assigns them to "clusters,"
association: identifies underlying relationships
dimensionality reduction: reduces the number of dimensions, or features, in a dataset to **improve the efficiency** of a model.
with AutoML and pre-built APIs you **don't need to specify a machine learning model**.
you'll **define your objective**, such as text translation or image detection. Then on the **backend**, **Google will select the best model to meet your busine**
With the other two options, BigQuery ML and custom training, you'll **need to specify which model** you want to train your data on and assign somethin
AutoML, you **don't need to worry about adjusting these hyperparameter** knobs because the tuning happens automatically in the **back end**.
This is largely done by a **neural architect search**, which finds the best fit model by comparing the performance against thousands of other models

Model evaluation:
Vertex AI provides extensive evaluation metrics:
confusion matrix (recall and precision): specific performance measurement for machine learning classification problems.
false positive/negative : Type 1/2 Error
Precision and recall are often a trade-off:
If the goal is to **catch (as many potential)(all) spam emails as possible (FN->0)**, Gmail may want to prioritize **recall**.
In contrast, if the goal is to **only catch the messages that are definitely spam** (FP->0) without blocking other emails, Gmail may want to pric
based on feature importance:
is displayed through a **bar chart** to illustrate **how each feature contributes to a prediction**.
feature importance is one example of Vertex AI's comprehensive machine learning functionality called **Explainable AI**
Explainable AI is a set of tools and frameworks to help **understand and interpret predictions** made by machine learning models.

Model deployment and monitoring:
model serving: **model deployment, model monitoring**
model management exists throughout this whole workflow to **manage the underlying machine learning infrastructure**
This lets data scientists focus on **what to do, rather than how to do** it.
MLOps combines **machine learning development** with **operations** and and applies similar principles from **DevOps** to machine learning models, which i
operations
MLOps aims (both data and code are constantly evolving in machine learning):
to solve **production challenges related to machine learning**.
building an **integrated machine learning system**
operating it i**n production**
This means adopting a process to enable **continuous integration, continuous training, and continuous delivery**.
three options to deploy a machine learning model:
to deploy to an **endpoint**: is best when **immediate results with low latency** are needed, such as **making instant recommendations** based on a user's b
online.
A model must be **deployed to an endpoint** before that model can be used to serve real-time predictions
to deploy using **batch prediction:**
This option is best when **no immediate response is required**, and **accumulated data** should be processed with a single request.

            - For example, **sending out new ads** every other week based on the user's recent purchasing behavior and what's currently popular on the market
        - to deploy using **offline prediction:**
            - This option is best when the model should be deployed in a **specific environment off the cloud**.
    - The backbone of MLOps on Vertex AI is a tool called **Vertex AI Pipelines:**
        - It **automates, monitors, and governs** machine learning systems by **orchestrating the workflow** in a **serverless** manner.
        - With **Vertex AI Workbench**, which is a **notebook tool**, you can **define your own pipeline**
- Lab introduction: Predicting loan risk with AutoML:
    - AutoML requires **at least 1,000 data points** in a dataset.
- Vertex AI: Predicting Loan Risk with AutoML:
    - [Vertex AI](), the **unified AI platform** on Google Cloud to **train and deploy** a ML model:
    - Vertex AI offers two options on one platform to build a ML model:
        - a codeless solution with **AutoML**
        - a code-based solution with **Custom Training** using Vertex **Workbench**
    - There are three options to import data in Vertex AI:
        - Upload a local file from your computer.
        - Select files from Cloud Storage.
        - Select data from BigQuery.
    - For **Budget**, which represents **the number of node hours for training**, enter **1**.
        - Training your AutoML model for 1 compute hour is **typically a good start for understanding whether there is a relationship between the features anc**
    - The **confidence threshold** determines how a ML model counts the positive cases.
        - A **higher threshold increases the precision, but decreases recall**. A lower threshold decreases the precision, but increases recall.
    - You can improve the percentage by **adding more examples (more data), engineering new features, and changing the training method**, etc.
    - These **feature importance** values could be used to help you improve your model and have more confidence in its predictions:
        - You might decide to **remove the least important features** next time you train a model or
        - to **combine two of the more significant features** into a [**feature cross**]() to see if this improves model performance.
            - Feature crosses are mostly used with **linear models** and are rarely used with neural networks.
    - Now that you have a trained model, the next step is to **create an endpoint in Vertex**.
        - A model resource in Vertex can have **multiple endpoints** associated with it, and you can **split traffic between endpoints**.
    - To allow the pipeline to authenticate, and be authorized to **call the endpoint to get the predictions**, you will need to provide your **Bearer Token**.
    - To use the trained model (get predictions), you will need to **create some environment variables**.
    - The smlproxy application is used to communicate with the backend.
- Lab recap: Predicting loan risk with AutoML:
    - how high (TP, TN) or low (FP, FN) they need to be really depends on the **business goals** you're looking to achieve
    - to improve the performance of a model:
    - using a more accurate data source,
    - using a larger dataset,
    - choosing a different type of ML model,
    - tuning the hyperparameters.

Course Summary:
- cloud.google.com/training/data-engineering-and-analytics
- cloud.google.com/training/machinelearning-ai
- cloud.google.com/certifications

How Google Does Machine Learning

Introduction to course and series:
- Course series preview
    - Roles:
        - machine learning scientists,
        - machine learning engineers
        - data scientists,
        - data engineers and
        - data analysts
    - Tools:
        - Vertex AI platform,
        - BigQuery ML,
        - TensorFlow
        - Keras
    - machine-learning products and concepts:
        - Analytics Hub,
        - Dataplex,
        - Data Catalog,
        - predictions
        - model monitoring,
        - data quality improvement
        - exploratory data analysis

what it means to be AI-first :

- Introduction:
    - build a data strategy around ML,
    - identify and solve ML problems,
    - infuse your applications with ML.
    - What kinds of problems can ML solve?
- What is ML?
    - ML
        - Machine learning is a way to use standard algorithms to **derive repeated predictive insights from data** and **make repeated decisions**
        - So machine learning is about **making many predictive decisions** from data
        - It's about scaling up business intelligence and decision-making
    - backward-looking use of data: looking at historical data to create reports and dashboards.
    - AI vs ML: AI is a discipline, ML is a specific way of solving AI problems
    - The ML model is a **mathematical function**
    - ML has two stages: **training and inference (prediction)**.
    - each of the layers of the Neural networks are a simple mathematical function. The entire model therefore consists of a **function of a function of a function** and so on.
    - Training **deep neural networks**, neural networks with lots of layers, takes a lot of computing power
    - Google has more than **10,000** plus deep learning models
    - In practice, you have to build **many ML models to solve the problem**
        - Avoid the trap of thinking of building **monolithic** "**one model solves the whole problem**" solutions
    - Google Photos:
        - This is the Google product where you can **upload photos from your camera to the cloud**. You don't need to tag it. The **ML software tags images** for you so that **you can then find images.**
    - Google Translate: lets you point a phone camera at a street sign, and it translates the sign for you:
        - One model to find the sign,
        - another model to read the sign through OCR, optical character recognition,
        - a third model to detect the language,
        - a fourth model to translate the sign,
        - a fifth model to superimpose translated text,
        - perhaps even a sixth model to select the font to use and so on
    - Gmail's Smart Reply:
        - This is arguably **the most sophisticated** ML model in production today.
        - It's a **sequence-to-sequence** model.
- What problems can it solve?
    - ML scales beter than the Hand-coded rules, because it's all automated
    - RankBrain, a deep neural network for search ranking
        - the system could continually improve itself based on what users actually preferred.
    - Replacing **heuristic rules** by ML, that's what ML is about
    - What kinds of problems can you solve with ML?
        - Anything for which you're writing rules today.
    - Google is an **AI-first** company: Google thinks of ML as the way to **scale, to automate, to personalise**
    - Rule-based vs model-based application
        - You don't think about **coding up rules**,you think about **training models based on data**.
        - You don't think about **fixing bug reports by adding new rule**s, you think in terms of **continuously training the model as you get new data**.
        - And when you think of **applying rules to inputs**, you think in terms of **deploying models at scale to make predictions**.
    - Machine learning is about collecting the appropriate data, and then finding the right balance of good learning, trusting the examples
- Activity intro: Framing a machine learning problem
    - Cloud ML use cases:
        - Manufacturing
        - Retail
        - healthcare and life sciences
        - travel and hospitality
        - financial services
        - Energy, feedstock and utilities.
- Activity solutions: Framing a machine learning problem:
    - ML problem:
        - what is being predicted?
        - what data do we need?
    - software problem:
        - What is the API of the service?
        - what does it take?
        - who's going to use this service?
        - How are they doing it today?
    - data problem:
        - what kind of data do we need to collect?
        - what data do we need to analyze?
        - what is our reaction? How do we react to a prediction?
- Infuse your apps with ML:
    - An easy way to add ML to your apps is to take advantage of pre-trained models.
    - **Aucnet**:
        - the new machine learning system can **detect the model number of the car** at high accuracy
        - It can also show the estimated price range for each model
        - and recognize what part of the car is being photographed
        - With this system, the car dealers just drag and drop a bunch of unclassified photos and check if the model and parts are classified by the system correctly.
        - Aucnet built a **custom** image model on Google Cloud platform using TensorFlow
        - you don't have to do that. There are a variety of domains where Google exposes ML services trained with their own data (Pre-trained (vs custom) ML models):
            - speech API: transcribe speech

- Vision API
- Translation API
- Natural Language API
- Jobs API
- Video Intelligence API

Ocado
is the world's largest **online-only grocery** based in the UK:
They were able to get **sentiment** entities and **pausing** syntax.
The computational technology helps Ocado **parse** through the body of e-mails, **tagging** and **routing** to help contact center reps determine the **priority and context**
Ocado use **parsed results from the NLP API** to **route** customer e-mails.
They don't want to send you an e-mail. They want to **talk to you interactively** to get their questions or concerns answered.
we'll be spending more on **conversation interfaces** than even on **mobile apps**

Dialogflow
high level conversational agent tool

Giphy
uses the Vision API to **find the text in memes** using **optical character recognition**.
The social media company used the Vision API to reject inappropriate uploads.

Uniqlo
designed a shopping chatbot using Dialogflow.

Build a data strategy around ML:
Google Maps
Where the roads are,
the traffic on each road, bridge closures,
Traffic and bridge closures are a little more difficult in that you have to work with a bunch of smaller government entitiesthe algorithm itself,
routing algorithms between A and B subject to a set of constraints.
You're in a subway station called Roppongi, and Maps tells you that you're on floor number two
Personalization of the map service is possible only with machine learning.
Machine learning is about scaling beyond handwritten rules.
asking for **user feedback to keep improving the model**
Given the choice between **more data and more complex models**, spend your energy collecting **more data**
That means collecting **not just more quantity, but also more variety**
An ML strategy is first and foremost a **data strategy**.
several reasons why you want to go through manual data analysis to machine learning:
if you're doing manual data analysis, you probably **have the data already**.
Collecting data is often the **longest and hardest** part of an ML project and the one most likely to fail.
Manual analysis helps you **fail fast and try new ideas in a more agile way**, so don't skip the analysis step.
to build a good ML model, you have to **know your data**.
Since that's the first step, go through the process of doing manual data analysis, don't **jump straight** to ML.
ML is a journey towards **automation and scale**.
training-serving skew:
This is where you had a certain system (**batch processing system**) for processing **historical** data so that you could **train** it
Then you have a different system that needs to **use the ML model during prediction**.
The system that serves these predictions is probably written in something that your **production engineering team** writes and maintains. Perhaps it's written in **Java** using **Web Frameworks**.
training-serving skew: The problem is that unless the model sees the exact same data in **serving** as was used to do **training**, the model predictions are going to be off
One way to reduce the chances of this is to take **the same code that was used to process historical data during training** and **reuse** it during **predictions**, but for that to happen, your **data pipelines have to process both batch and stream**.
This is a key insight behind **cloud dataflow**
a way to offer **data pipelines in Python, Java** or even visually with Cloud Dataprep.
It's **open-sourced as Apache Beam**, where the B stands for **batch** and -eam stands for **stream**.
A single system to do both batch and stream
in machine learning, it's helpful to use **the same system in both training and prediction**
During training, the key performance aspect you care about is **scaling to a lot of data** (**distributed training)**
During prediction, though, the key performance aspect is **speed of response**, high **QPS**.
This is a key insight behind **TensorFlow**.
the magic of ML comes with **quantity not complexity**.
If you're building many ML models and planning for many more that you may never build, you want to have an environment where you **fail fast**. The idea is that **if you're failing fast, you get the ability to iterate**.

Quiz:
ML training phase:
Evaluating the models
Data management
Create the models
~~Connecting Neural Networks~~
to replace user input by machine learning?
~~Neural networks.~~
~~All options are correct.~~
~~Labeled data.~~
Pre-trained models.
best practices for Data preparation
Avoid target leakage
Provide a time signal
Avoid training-serving skew

How Google Does ML:
machine learning,
we mean the process by which one computer writes a computer program (rule) to accomplish a task. which the best program is by only looking at a set of examples.
normal software engineering,

We have a human who analyzes the problem, writes a bunch of code, and then this code becomes a program that can translate inputs to outputs.

Fully end-to-end ML effort allocation:
- defining the **KPI**,
  - what you should even be trying to accomplish,
- collecting data,
- building the infrastructure,
- **optimizing the ML algorithm** itself (the most),
- integrating with the rest of the pre-existing systems at your organization.

The secret sauce
- Top 10 ML pitfalls:
  - ML requires just **as much software infrastructure**
    - you thought training your own ML algorithm would be faster than writing the software.
    - And the reason is that to make a great ML system, beyond just the algorithm, you're going to need lots of things around the algorithm. like a whole software stack to serve, to make sure that it's robust, and it's scalable, and has great uptime.
    - But then, if you try to use an algorithm, you put in additional complexities around **data collection, training**, and all of that just gets a little bit more complicated.
  - **No data** collected yet
  - If there's not someone in your organization who's **regularly reviewing that data** or generating reports or new insights, if that data isn't generating value already, likely, the effort to maintain it isn't being put in.
  - keep **humans in the loop**:
    - they're reviewing the data,
    - handling cases the ML didn't handle very well,
    - curating its training inputs.
  - You launched a product whose initial value prop was its ML algorithm instead of some other feature.
  - ML system, and it just happens to **optimize for the wrong thing**
  - if you forget to measure if your ML algorithm is actually **improving** things in the real world?
  - you confuse the ease of use and the value add of somebody else's **pre trained** ML algorithm with **building your own**
  - ML algorithms are going to be **retrained many times**
  - And in fact, all the algorithms we have to address these are **very highly tuned from decades of academic research**. And you should almost always take one off the shelf already made, or already kind of defined, instead of trying to do your own research, as that is very expensive.

ML and business processes:
- business processes: any set of activities a company must do directly or indirectly to serve customers.
  - almost every one of them has a feedback loop.
- General feedback loop:
  - Input-process-outputs-insight generation-tuning-process

The path to ML
- Input-process(individual contributor,delegation,digitization)-outputs-insight generation(big data and analytics)-tuning(ML)-process
- individual contributor
  - A task or business process that's in the individual contributor phase is performed by a single person.
  - The task is not parallelized or scaled at all and is usually very informal.
- delegation
  - multiple people who are all performing the same task in parallel
  - there's some repeatability in the task
- Digitization
  - we take the **core repeatable** part of the task or a business process, and we **automate** it with computers.
  - We want to give our users a higher quality of service
  - it involves so much upfront investment,
- Big Data and analytics
  - we're going to use a lot of data to **build operational and user insights**.
- ML:
  - Here we use all the data from the previous step.
  - We'll automatically start to improve these computer processes.

A closer look at the path to ML
- individual contributor
  - Dangers of Skipping:
    - Inability to scale
    - incorrect assumptions that are hard to change later.
  - Dangers of Lingering
    - Imagine that you've got one person who's very skilled at their job but then leaves the company or retires.
      - All that organizational knowledges leaves with them. It's a problem when no one else can perform that process.
    - Also in this phase you can't scale up the process to meet a sudden increase in demand.
- delegation
  - Dangers of Skipping:
    - you're never forced to formalize the business process and to define success.
    - Human responses have an inherent diversity
    - great ML systems will need humans in the loop
      - If your ML system is very important, it should be reviewed by humans
  - Dangers of Lingering
    - you're paying a very high marginal cost to serve each user
    - The more voices you have in your organization, automation is less possible.
    - This creates a kind of of **organizational lock-in** because you have too many **stakeholders**
- Digitization
  - Dangers of Skipping:
    - you'll need all the infrastructure of this step to be able to serve your ML at scale
    - you might start to untangle an IT project, which we may call software, with an ML project. If either one of them fails, the whole project fails.
  - Dangers of Lingering
    - the other members of your industry are collecting data and tuning their offers and operations from these new insights
- Big Data and analytics

Dangers of Skipping:
you won't be able to train your ML algorithm because your data isn't clean and organized
you can't build a measure of success
Dangers of Lingering
limiting the **complexity** of problems you can solve and the **speed** at which you can solve them
End of phases deep dive
Almost every single one has a team of people reviewing the algorithms, reviewing their responses and doing random sub-samples and it generates a lot of value for the organization, for customers and for end users.
facets that differentiate deep learning networks in multilayer networks?
More complex ways of connecting layers
Cambrian explosion of computing power to train
Automatic feature extraction
ML development with Vertex AI
Introduction
To build an ML for production
identifying a goal,
acquiring, exploring, and preparing data,
building,
training,
evaluating the model.
Deployment (**web client** that can **request** predictions from the model)
monitored
maintained.
the proof of concept, or **experimentation phase**: the process of determining whether the model is **ready for production**
Moving from experimentation to production
ML development during the experimentation phase:
framing the problem
you identify your **use case**,
questions typically asked are what you're trying to do? What are the minimum requirements of your business application?
preparing the data,
you might use a subset of a larger data set.
You would also perform **EDA** or exploratory data analysis and seek to improve data quality
You'd also consider combining features to create a new feature (**Feature engineering**)
experimenting,
you experiment with **different models** to compare performance metrics.
evaluating the model
recall, precision, an F1 score, or cross-entropy
ML practitioners train models using
different architectures,
CNNs, or convolutional neural networks are used for image classification, object detection, and recommender systems.
RNNs or recurrent neural networks are used for sequence modeling,next word prediction, translating sounds to words and human language translation.
Sorting and clustering architectures are used for anomaly detection, and pattern recognition.
GANS, or generative adversarial networks are used for anomaly detection,pattern recognition, cybersecurity, self-driving cars, and reinforced learning.
different input datasets,
numerical data sets,
bivariate data sets,
multivariate data sets,
categorical data sets,
correlation data sets
different hyperparameters
learning rate,
number of layers,
num_estimators,
max_depth
different hardware
CPUs,
GPUs,
TPUs
Moving from experimentation to production requires:
packaging,
As you package your code for production, you need to **build and install a Python package** out of your predictive model in Python.
deploying,
A trained module can be deployed in various ways, such as on a **user's mobile device**, or as a **web service in a container running on a cluster**.
Deploying the model also requires you to **set up endpoints** which allow your app to serve predictions.
a machine learning model can have a **web app front end**. a **request** is sent by a **web client** using a **REST API**, a prediction made by the model is returned.
monitoring your model
model stale: the underlying data distribution may have shifted over time
the model may have been **misconfigured** in its production deployment
Then, a **model retraining deployment pipeline** can be triggered
Monitoring measures key model performance metrics, and includes
model drift,

model performance,
model outliers
data quality

to build a custom model:
- you need to know an **ML framework** such as **TensorFlow and Keras**.
- You need to know how to **upload** the model to **Google storage** using code,
- how to **host** a model on an **AI platform**.
- You also need to know how to create a **service account** to **access** the model,
- how to **wrap the app in a Docker container**,
- how to **push the Docker container to the Google Cloud registry (GCR)**.
- You then need to know how to **deploy** the cloud registry container to **App Engine** to serve predictions.
- you need to enable **monitoring** of the app to assure **model stability**.

You could build above components **separately or in a pipeline**. But a **unified platform** could offer a more efficient way to achieve your objective or goal

What is there to unify:
- Dataset: We **create** datasets by **ingesting** data, **analyzing** the data and **cleaning** it **up (ETL, ELT)**.
- Model:
  - model **training**. This includes experimentation, hypothesis testing, and hyperparameter tuning.
  - The trained model is **versioned** and **rebuilt** when there's new data on a schedule, or when the code changes (**ML Ops**).
  - The model is **evaluated** and compared to existing model versions.
  - The model is **deployed** and used for **online and batch** predictions.

Vertex AI provides unified definitions and implementations (unified set of APIs for the ML lifecycle) of **four** concepts.
- A **data set** can be structured or unstructured, it has **managed metadata**, including **annotations**, and can be stored anywhere on Google Cloud.
- A **training pipeline is a series of containerized steps** that can be used to train an ML model using a data set, the containerization helps with **generalization, reproducibility, and auditability**.
- A **model** is an ML model with **metadata** that was built with a **training pipeline, or directly loade**d only if it's in a compatible format.
- And an **endpoint** can be invoked by users for online predictions and explanations.

Vertex AI to manage the following stages in the ML workflow:
- Create a data set
- upload data.
- Train an ML model on your data, train the model,
- evaluate model accuracy,
- tune hyperparameters and custom training only.
- **Upload and store your model in Vertex AI**.
- **Deploy** your trained model to an **endpoint** for serving predictions.
- Send **prediction requests** to your endpoint,
- specify **prediction traffic split** in your endpoint,
- manage your models and endpoints.

choose a training method:
- AutoML
  - AutoML lets you create and train a model with **minimal technical effort**.
  - Even if you want the flexibility of a custom training application, you can use AutoML to **quickly prototype** models, and **explore new datasets** before investing in development.
- Custom training
  - lets you create a training application **optimized for your targeted outcome**.
  - You have **complete control over training** application functionality.
  - You can target any **objective**, use any **algorithm**, develop your own **loss functions** or **metrics** or do any other customization

Vertex AI offers
- fast experimentation,
- accelerated deployment
- simplified model management

Components of Vertex AI
- Vertex AI dashboard:
  - Datasets:
    - After you load data into Vertex AI, whether it's from cloud storage or BigQuery, it's **managed** by Vertex AI.
    - This means **it can be linked to a mode**l.
  - Features:
    - **Vertex AI Feature Store** is a **fully managed repository** where you can **ingest, serve, and share** ML feature values within your organization.
    - Vertex AI Feature Store **manages all of the underlying infrastructure** for you.
    - For example, it provides **storage and compute resources** for you and can easily scan as needed
  - labeling tasks
    - let you request **human labeling** for a dataset that you plan to use to train the custom machine learning model.
  - Workbench
    - is a **Jupyter notebook based development environment** for the entire **data science workflow**.
    - Vertex AI Workbench lets you **access data, process data in a Dataproc cluster, train a model, share** your results, and more.
    - All without leaving the Jupyter lab interface.
  - Pipelines
    - helps you to **automate, monitor, and govern your ML system**s by orchestrating your ML workflow in a **serverless** manner, and storing your **workflow's artifacts** using Vertex ML **metadata**.
    - By storing the artifacts of your ML workflow in Vertex ML metadata, you can analyze the **lineage** of your workflow's artifacts.
    - For example, an ML model's lineage may include the **training data, hyperparameters, and code** that were used to create the model.
    - The key takeaways are that, pipelines allow you to automate, monitor, and **experiment with interdependent parts of an ML workflow**.
    - ML pipelines are **portable, scalable, and based on containers** and each individual part of your pipeline workflow for example creating a dataset or training a model, **is defined by code**.
    - This code is referred to as a **component**.
    - Each instance of a component is called a **step**
  - Training

You can train models on Vertex AI, using **AutoML**, or use **custom training** if you need the wider range of **customization** options available in AI platform training.
In custom training you can select many different machine types to power your training jobs, enable **distributed training**, use **hyperparameter tuning**, and **accelerate with GPUs**.

Experiments
includes Vertex **Vizier**, which is an **optimization service** that helps you **tune hyperparameters** in complex machine-learning models.
You can **run different studies and compare them using TensorBoard**.

Models
built from the **dataset or unmanaged data source**s.
Many different types of machine learning models are available in Vertex AI, depending on your **use case** and **level of experience with machine learning**.
Managing and deploying models manually can involve writing an **application or framework** to load the model and serve the inferences.
The application might also need to handle pre or post-processing steps and the incoming traffic.
Vertex AI's model resources help to **manage your model on Google Cloud** including **deploying, generating predictions and hyperparameter tuning**.
Vertex AI models can handle both AutoML models and custom trained models.

Endpoints:
Vertex AI lets you **deploy a trained model to an endpoint** for **serving predictions**.
Models can be deployed in Vertex AI, **whether or not the model was trained on Vertex AI**.

Batch prediction
intakes a **group of prediction requests** and outputs the results to a specified location.
Use batch prediction when you **don't require an immediate response**, and want to **process accumulated data with a single request**.

Metadata
stores **artifact** and **metadata for pipelines run** using **Vertex AI pipelines**.
Each pipeline run produces metadata and ML artifacts, such as the **training, test, and evaluation data** used to create the model.
The **hyperparameters** used during model training, and the **code** that was used to train the model.
It also includes the metadata recorded from the training and evaluation process, such as the model's **accuracy** and artifacts that descend from this model, such as the results of **batch predictions**.

Lab intro: Using an image dataset to train an AutoML model
create an image classification dataset
import images,
train an AutoML classification model,
deploy a model to an end point
send a prediction.

Lab demo: Using an image dataset to train an AutoML model
be sure that we have the Vertex AI API enabled
create our first managed data set

Using an Image Dataset to Train an AutoML Model
Enable the APIs
In the **Google Cloud Console**, on the **Navigation menu**, click **Vertex AI > Dashboard**.
Click **Enable all recommended API**.
Previously, models trained with **AutoML** and **custom models** were **accessible via separate services**.
The new offering **combines both into a single API**, along with other new products.
You can also **migrate existing projects to Vertex AI**.
Vertex AI includes many different products to support **end-to-end ML workflows**.
Dataset:
These input images are stored in a **public Cloud Storage bucket**.
This publicly accessible bucket also contains a **CSV** file you use for data **import**.
This file has **two columns**: the first column lists an image's **URI** in **Cloud Storage**, and the second column contains the image's **label**.
Create an image classification dataset and import data
on the **Vertex AI** page, in the navigation pane, click **Dashboard**
In the central pane, click **Create dataset**.
Optional: Specify a name for this dataset.
For **Select a data type and objective**, on the **Image tab**, select **Image classification (Single-label)**.
For **Region**, select **us-central1**.
To create the **empty dataset**, click **Create**.
The **Data import** page opens.
Select **Select import files from Cloud Storage**, and specify the **Cloud Storage URI of the CS**V file with the image location and label data.
When import process is complete, the next page shows all of the images, both labeled and unlabeled, identified for your dataset.
Review imported images
After your dataset is **created** and data is **imported**, use the **Cloud Console to review the training images** and begin model training.
After the dataset is imported, the **Browse** tab opens.
You can also access this tab by selecting **Datasets** from the side menu, and then selecting the ***annotation*** *set* (set of single-label image annotations) associated with your new dataset.
Train an AutoML image classification model
On the **Browse tab**, you can choose **Train new model** to begin training.
You can also start training by selecting **Models** from the side menu, then selecting **Create**.

1. On the **Vertex AI** page, in the navigation pane, click **Model Registry**.
2. To open the **Train new model** page, click **Create**.
3. Under **Training method**, select the target **Dataset** and **Annotation set** if they are not automatically selected.
4. Select **AutoML**, and then click **Continue**.
5. Optional: Under **Model details**, type a **Model name**.

6. Click **Continue**.
7. Leave the **Explainability** section as default and click **Continue**.
8. Under **Compute and pricing**, for **Budget**, enter **8** maximum node hours.
9. Click **Start training**.

Training takes about **2 hours**. When the model finishes training, it is displayed with a **green checkmark status icon**.

Deploy a model to an endpoint and send a prediction

After your AutoML image classification model training is complete, **use the Google Cloud Console to create an endpoint and deploy your model to the endpoint**.

After your model is deployed to this new endpoint, send an image to the model for label prediction.

Access your trained model to deploy it to **a new or existing endpoint** from the **Models** page.

1. On the **Vertex AI** page, in the navigation pane, click **Model Registry**.
2. Select your trained AutoML model and then click on **Version ID**.
   The **Evaluate** tab opens, where you can view model performance metrics.
3. On the **Deploy & Test** tab, click **Deploy to endpoint**.
   The **Endpoint options** page opens.
4. Under **Define your endpoint**, select **Create new endpoint**, and for **Endpoint name**, type **hello_automl_image**.
5. Click **Continue**.
6. Under **Model settings**, accept the **Traffic split** of **100%**, and set the **Number of compute nodes** to **1**.
7. Click **Deploy**.

Send a prediction to your model

After the endpoint creation process finishes, you can **send a single image annotation (prediction) request in the Cloud Console**.

In the **Test your model** section of the same **Deploy & test** tab you used to create an endpoint in the previous subtask, click **Upload image**, choose a local image for prediction, and view its predicted label.

Training an AutoML Video Classification Model

Task 4. Deploy a model to make batch predictions

i. On the **Batch Predict** tab, click **Create Batch Prediction**.
ii. Provide a batch prediction name.
iii. For the **Source path**, use **automl-video-demo-data/hmdb_split1_predict.jsonl**
iv. For the **Destination path** to your bucket, click **Browse**.
v. Click **Create new bucket**, type **Project_ID**.
vi. Click **Create**.
vii. Click **Create new folder**, type **predict_results**.
viii. Click **Create**, and select the destination path.
ix. Click **Create**.
x. You can navigate to **Cloud Storage** to find your bucket name. Results are added to the **predict_results** folder.

View results

When the job is complete, your prediction is displayed on the **Batch prediction**s tab.

i. Click on the prediction in the **Batch prediction** view.
   Details of the batch prediction job appear.
ii. Click the **Export location** link to view the results in your storage bucket.
iii. To see your results in the UI, click **View results**.
   A video appears. From the dropdown menu at the top of the page, you can select other videos you want to see the results for.

Understanding the results

In the results for your video annotation, Vertex AI provides three types of information:

- Labels for the video: This information is on the **Segment** tab below the video on the results page.
- Labels for shots within the video: This information is on the **Shot** tab below the video on the results page.
- Labels for each 1-second interval within the video: This information is on the **Interval** tab below the video on the results page.

If the prediction fails, the results in the list show a red icon on the **Recent Predictions** list.

If only one video in the prediction attempt failed, the results show a green icon in the **Recent Predictions** list. On the results page for that prediction, you can view the results for the videos that Vertex AI has annotated.

Tools to interact with Vertex AI

You can deploy models to the Cloud and **manage** your **datasets, models, endpoints and jobs** on the Cloud Console.

This option gives you a **user interface** for working with your machine-learning resources.

As part of Google Cloud, your **Vertex AI resources** are connected to useful tools like **Cloud Logging and Cloud Monitoring**.

Tools:

- **client library**: for some languages to help you **make calls to the Vertex AI API**.
  The client libraries provide an **optimized developer experience** by using each supported language's **natural conventions and styles**.
  Alternatively, you can use the **Google API Client Libraries** to access the Vertex AI API by using other languages such as [Indistinct].
  When using the Google API Client Libraries, **you build representations of the resources and objects used by the API**.
  This is easier and requires less code than **working directly with HTTP requests**.
  For example, Cloud client libraries include **Python, Node.js and Java**.
- The **Vertex AI REST API:** provides **RESTful services for managing jobs, models and endpoints, and for making predictions with hosted models on Google Cloud**
- **Deep Learning VM Images:** is a set of virtual machine images **optimized for data science and machine-learning tasks**.
  All images come with key ML frameworks and tools preinstalled.
  You can use them out of the box on instances with GPUs to accelerate your data processing tasks.
  Deep Learning VM Images are available to support many combinations of **framework and processor**.
  There are currently images supporting **TensorFlow Enterprise, TensorFlow, PyTorch** and generic high-performance computing with versions for both **CPU-only and GPU-enabled workflows**.
- **Deep Learning Containers** are a set of **Docker containers** with key data science frameworks, libraries and tools preinstalled.

These containers provide you with **performance-optimized consistent environments** that can help you **prototype** and **implement workflows** quickly.

Quiz: Machine Learning Development with Vertex AI

managed dataset in Vertex AI?

Data loaded into Vertex AI - whether it be from **Google Cloud Storage or BigQuery**. This means, for example, that it can be **linked to a model**.

Machine Learning Development with Vertex Notebooks

Machine Learning Development with Vertex Notebooks

Vertex AI Workbench provides **two Jupyter notebook-based options** for your **ML workflow**.

i. managed notebooks

Managed notebooks instances are **Google-managed environments** with **integrations and features** that help you set up and work in an **end-to-end notebook-based production environment**.

are usually a good choice if you want to use a notebook for **data exploration, analysis or modeling, or as part of an end-to-end data science workflow**.

Managed notebooks instances lets you perform **workflow-oriented tasks without leaving the JupyterLab interface**.

They also have many **integrations and features** for implementing your data science workflow.

a. **Control your hardware and framework** from JupyterLab.

In a managed notebooks instance, your JupyterLab interface is where you determine **what compute resources** -- for example, how many **VCPUs or GPUs** and how much **RAM** -- your code will run on and what **framework** you want to run the code in.

This means you can write your code first and then choose how to run it **without having to leave JupyterLab or restart your instance**.

This makes it easy to **scale your hardware down** for **quick tests** of your code and then **scale it back up** when you need to run your code on more data.

b. **Custom containers**.

Your managed notebooks instance includes many common data science frameworks to choose from such as **TensorFlow** and **PyTorch**, (PySpark, R) but you can also add **custom Docker container images** to your **instance**. Your custom containers are available to use **directly** from the JupyterLab interface alongside the pre-installed frameworks.

Training: Dockerfile, Cloud Build, Container Registry, Vertex Training

c. **Access to data**. Managed notebooks lets you access your data **without leaving the JupyterLab interface**.

In JupyterLab's left sidebar, use the **Cloud Storage extension** to browse data and other files that you have access to.

Also in the left sidebar, use the **BigQuery extension** to browse tables that you have access to, write queries, preview results and load data into your notebook.

d. **Dataproc integration**.

You can **process data quickly** by **running** a notebook on a **Dataproc cluster**.

After your **cluster is set up**, you can run a notebook file on it **without leaving the JupyterLab interface**.

Automated shutdown for idle instances. To help **manage costs**, you can set your managed notebooks instance to **shut down after being idle for a specific time period**.

ii. **user-managed notebooks**:

Deep Learning VM Images instances:

User-managed notebooks are **Deep Learning VM Images instances** that are **heavily customizable** and are ideal if you need **a lot of control over your environment**.

you select your **machine type and the framework** for your instance when you create it.

You **can change your instance's machine type after creation**, although this requires a **restart** of your instance.

You **can't easily change the framework on your instance**, but you can still make manual modifications like updating software and package versions.

Additionally, because user-managed notebooks instances are exposed as **Compute Engine instances**, you can customize them in the same way that you can customize Compute Engine instances.

Health status monitoring.

User-managed notebooks instances provide several methods for monitoring the health of your notebooks, including a **built-in diagnostic tool**.

For example, the diagnostic tool verifies the status of **core services**, including **Docker and Jupyter**.

It checks whether the **disk space for boot and data disks is used beyond an 85 percent threshold**, and collects instance logs on network information, Docker, Jupyter and proxy service status

Networking and security.

For users who have specific networking and security needs, user-managed notebooks can be the best option.

You can use **VPC Service Controls** to set up a user-managed notebooks instance within a service parameter and implement other built-in networking and security features.

You can also configure user-managed notebooks instances manually to satisfy some specific networking and security needs.

Both options are **pre-packaged with JupyterLab** and have a **pre-installed suite of Deep Learning packages**, including support for the TensorFlow and PyTorch frameworks.

Both options support **GPU accelerators** and the ability to **sync with a GitHub repository**.

And both options are **protected by Google Cloud authentication and authorization**.

Vertex AI Model Builder SDK: Training and Making Predictions on an AutoML Mode

to train and make predictions:

Vertex AI Python client library

`gcloud` command-line tool

online Cloud Console.

Set up your environment

Enable the Notebooks API

1. In the Google Cloud Console, on the **Navigation menu**, click **APIs & Services** > **Library**.
2. Search for **Notebooks API** and press enter. Click on the `Notebooks API` result.
3. If the API is not enabled, you'll see the **Enable** button. Click **Enable** to enable the API.

Enable the Vertex AI API

In the Google Cloud Console, on the **Navigation menu**, click **Vertex AI > Dashboard**, and then click **Enable Vertex AI API**.

Launch a Vertex AI Notebooks instance

i. In the Google Cloud Console, on the **Navigation Menu**, click **Vertex AI > Workbench**. Select **User-Managed Notebooks**.
ii. On the Notebook instances page, click **New Notebook > TensorFlow Enterprise > TensorFlow Enterprise 2.6 (with LTS) > Without GPUs**.
iii. In the **New notebook** instance dialog, confirm the name of the deep learning VM, if you don't want to change the region and zone, leave all settings as they are and then click **Create**. The new VM will take 2-3 minutes to start.
iv. Click **Open JupyterLab**.
    A JupyterLab window will open in a new tab.
v. You will see "Build recommended" pop up, click **Build**. If you see the build failed, ignore it.

Clone a course repo within your Vertex AI Notebooks instance

To clone the training-data-analyst notebook in your JupyterLab instance:

1. In JupyterLab, to open a new terminal, click the **Terminal** icon.
2. At the command-line prompt, run the following command: `git clone URL`
3. To confirm that you have cloned the repository, double-click on the training-data-analyst directory and ensure that you can see its contents.

Best practices for implementing machine learning on Vertex AI

Best practices for machine learning development

- best practices for **preparing and storing your data**
  - tabular/structured data:
    - store all data in BigQuery,
    - you can also store **intermediate processed** data in BigQuery.
    - For **maximum speed**, it's better to store **materialized** data instead of using **views or sub-queries** for training data.
    - Use Vertex AI **Feature Store** with structured data.
    - When you're training a model with **structured** data, irrespective of way you're **training that model**, follow these steps.
      a. Search Vertex AI Feature Store
         i. to determine whether **existing features satisfy your requirements**.
            Open Vertex AI Feature Store and search to see whether **a feature already exists** that relates to your use case or covers the signal that you're interested in passing to the model.
         ii. If Vertex AI Feature Store contains features that you want to use, **fetch those features** for your training labels using Vertex AI **Feature Store's batch serving** capability.
      b. Create a new feature.
         i. If Vertex AI Feature Store doesn't have the features you need, create a new feature using data from your **data lake (Cloud storage bucket or BigQuery)**.
         ii. **Fetch raw data from your data lake** and write your **scripts** to perform the necessary **feature processing and engineering**.
      c. **Join** the feature values you **fetch from Vertex AI Feature Store** and the **new feature values** that you created from the data lake.
         Merging those feature values produces the training dataset.
      d. Set up a **periodic job** to compute **updated values of the new feature**.
         When you determine that a feature is **useful** when you want to put it into **production**, set up a **regularly scheduled job** with the required cadence to compute updated values of that feature and **ingest** it into Vertex AI Feature Store.
         By **adding your new feature to Vertex AI Feature Store**, you automatically have a solution to **online serving** of the features for online prediction use cases, and you can **share** your features with **others in the organization** that may get value from it for their own ML models.
  - image, video, audio and unstructured
    - Store data in **large container formats in Cloud Storage**.
    - This applies to **sharded TFRecord** files if you're using **TensorFlow** or **AVRO** files if you're using any other framework.
    - **Combine** many individual images, videos or audio clips **into large files**.
      - This would **improve your read and write throughput to Cloud Storage**.
    - Aim for files of at least **100 megabytes** and between 110,000 shards.
    - To enable **data management**, use **Cloud Storage buckets and directories** to group the shards.
    - Use **Vertex Data Labeling** for unstructured data.
      - You might need humans to provide labels to your data, especially with unstructured data.
      - You can hire your own labels and use Good Cloud's software for managing their work, or you can use Google's **in-house** labels for the task.
  - Avoid storing data in **block storage** like **network file systems or in virtual machine hard disk**s.
    - Those tools are **harder to manage than Cloud Storage or BigQuery** and often present challenges in shooting performance.
  - Similarly, avoid reading data directly from **databases** like **Cloud SQL**.
    - Instead, store data in **BigQuery in Cloud Storage**
- best practices for training a model with Vertex AI:
  - Training a model within the **Notebooks** instance my be sufficient for **small datasets or for subsets of a larger dataset**.
  - It may be helpful to use the **training service** for **larger** datasets or for **distributed training**.
  - Using the Vertex Training service is also recommended to **productionized training**, even on small datasets if the training is carried out on a **schedule** or **in response to the arrival of additional data**
  - Vertex AI Training provides a set of **prebuilt algorithms** that allows users to **bring their custom codes to train models**.
  - This is a **fully managed training service** for users needing **greater flexibility and customization**, or for users running training on premises or another cloud environment.
  - The workflow presented here shows
    - **training with prebuilt containers** using Vertex Training.
    - The **training application** with your model is **packaged** and then **pushed** to a **Cloud Storage bucket**.
    - You can then **pull** the package from the bucket and run the **job** on Vertex Training using **prebuilt containers**, which is **simpler** than creating your own **custom Docker image container** for training
- best practices for Explainable AI

Vertex Explainable AI is an integral part of the ML implementation process, offering **feature attributions** to provide insights into why models generate predictions.
By **detailing the importance of each features** that the model uses as input to make a prediction, Vertex Explainable AI helps you **better understand your model's behavior and build trust in your models**.
Vertex Explainable AI supports custom-trained models based on tabular and image data.
hyperparameter tuning with Vertex Training
To maximize your model's predictive accuracy, use hyperparameter tuning.
This is an **automated model enhancer** provided by Vertex Training that takes advantage of the **processing infrastructure of Google Cloud** to test different hyperparameter configurations when training your model.
Hyperparameter tuning **removes the need to manually adjust hyperparameters** over the course of numerous training runs to arrive at the **optimal values**
best practices for using Workbench Notebooks
Use Workbench Notebooks to **evaluate and understand your models**.
In addition to built-in common libraries like scikit-learn, Notebooks offers **What-if-Tool, WIT**, and **Language Interpretability Tool, LIT**.
WIT lets you **interactively analyze your models** for **bias** using multiple techniques, and LIT enables you to **understand natural language processing model behavior through a visual, interactive and extensible tool**.
best practices for using **Vertex AI TensorBoard**
When developing models, use Vertex AI TensorBoard **to find and compare specific experiments**.
For example, based on hyperparameters, Vertex AI TensorBoard is an **enterprise-ready managed service**.
It provides a **cost effective, secure** solution that lets **data scientists and ML researchers** collaborate easily by making it seamless to track, compare and share their experiments.
Vertex AI TensorBoard lets you **track experiment metrics, such as loss and accuracy, over time, visualize a model graph, project embeddings to a lower dimensional space** and much more.
Data preprocessing best practices
best practices for data preprocessing
BigQuery: tabular data,
Dataflow: unstructured data
TensorFlow Extended: managed datasets (data linked to your models)
If you're using tabular data, use BigQuery for data **processing and transformation** steps.
When you're working with ML, use **BigQuery ML** in BigQuery.
After your data is **preprocessed** for ML, you may want to consider using a **managed dataset** in Vertex AI.
Managed datasets enable you to create **a clear link between your data and custom-trained models** and provide **descriptive statistics and automatic or manual splitting into train, test, and validation sets**.
Managed datasets **aren't required**.
You may choose **not** to use them if you want **more control over splitting** your data in your training code or if **lineage** between your data and model isn't critical to your application.
With **large volumes** of **unstructured** data, consider using **Dataflow**, which uses the **Apache beam** programming model.
You can use Dataflow to convert the unstructured data into **binary data formats like TFRecord**, which can improve performance of data **ingestion** during the training process.
If you need to perform transformations that are **not expressible in Cloud SQL**, or are for **streaming**, you can use a **combination of Dataflow and the pandas library**
If you're using TensorFlow for model development, use **TensorFlow Extended** to **prepare your data from training**.
TensorFlow Transform is the TensorFlow component that enables **defining the executing the preprocessing function to transform your data**.

Best practices for machine learning environment setup
- **Workbench Notebooks**
Use notebooks for **experimentation and development** including **writing code, starting jobs, running queries and checking status**.
It's a common practice to **customize Google Cloud properties** like **network, Cloud Identity, Access Management and software for a container associated with a notebook**
Create notebooks instance for **each team member**.
If a team member is involved in multiple projects, especially projects that have **different dependencies**, we recommend using multiple notebooks instances and treating each notebooks instance as a **virtual workspace**
And use Vertex **SDK for Python**
- **security**
Help secure **PII (**personally identifiable information**)** in notebooks.
take a look at the **notebooks security blueprint protecting PII data guide**, which provides guidance about applying data governance and security policies to help protect your notebooks.
See also the accompanying deployable blueprint in GitHub
- **Data & model**
Store prepared data and your model in the same project.
You can store prepared data in the **Google Cloud project** where your model is stored.
This will give your AI project **access to all of the datasets required** for modeling and will help to ensure that there are no breaks in reproducibility.
However, different parts of your organization might store their data in different projects, and ML models may need to rely on raw data from different projects.
- Optimize performance and costs.

responsible AI development
inclusive machine learning
ML fairness (biases that ML can amplify), explainability, privacy and security,
equality of opportunity: (there is an equal chance of a machine learning system correctly classifying an outcome **regardless of sensitive attributes**)
getting the best results out of a machine learning system requires that you truly understand your data
Facets: open-source visualization tool
Human biases lead to biases in ML models
just because something is based on data doesn't automatically make it neutral
Biases:
interaction bias:
Like this recent game where **people were asked** to draw shoes for the computer, **most people drew ones** like this so as more people interacted with the game the computer didn't even recognize these.
Latent bias:

For example, if you were training a computer on what a physicist looks like and you're using pictures of **past** physicists, your algorithm will end up with a latent bias skewing towards men.
selection bias:
Say you're training a model to recognize faces, whether you grab images from the internet or your own photo library, are you making sure to select photos that **represent everyone**?
Examples:
from tackling **offensive or clearly misleading information from appearing at the top of your search results page**, to adding a **feedback tool on the search bar** so people can **flag hateful or inappropriate auto complete suggestions**.
Biases in data
Unconscious biases exist in our data
human biases that exist in **data:** because data found in the world has existing biases with properties like gender, race and sexual orientation.
reporting bias
Subjects only choose to reveal certain aspects about themselves or their opinions.
selection bias
subjects that get into our samples only represent a **privileged** type of user.
human biases in **data collection and labeling** procedures:
Confirmation bias
only looking for data which may confirm our hypotheses.
Automation bias
appear when the data we use is only the data we can easily automate.
Google has announced seven AI principles (not theoretical concepts but concrete standards)
The challenge we run into when creating a system that is **fair** and **inclusive** to all is that ML models learn from **existing data collected from the real world**, so an accurate model may **learn** or even **amplify problematic preexisting biases** in the data based on **race, gender, religion** or other characteristics.
a quick checklist for situations where you should watch out for bias-related issues.
Does your use case or product specifically use any of the following data, **biometrics, race, skin color, religion, sexual orientation, socioeconomic status, income, country, location, health, language or dialec**t?
Does your use case or product use data that is likely to be **highly correlated with any of these personal characteristic**s? For example, **zip code or other geospatial data** is often correlated with **socioeconomic status** and **income**.
Image and video data can **reveal information about race, gender and age**.
Could your use case or product negatively individuals economic or other important life opportunities?
tools to diagnose **fairness issues in your data, in your labels and in the effects of predictions**:
What-If Tool
access from within **TensorBoard**
visualize inference results,
edit a data point
see how your model performs,
explore the **effects of a single feature**,
arrange examples for similarity,
view confusion matrices
test algorithmic fairness constraints.
Evaluating metrics with **inclusion** for your ML system
A confusion matrix helps in understanding inclusion and how to introduce inclusion across different types of groups in your data
This is only for classification problems
FPR=FP/(FP+TN)
FNR=FN/(FN+TP)
TPR=TP/(TP+FN)
Precision=TP/(TP+FP)
FP=Type I error
FN=Type II error
missing a lot of stuff in exchange for **high precision** or of the limited about of stuff the ML classifies **it's all correct**.
For email spam: FP should be minimised
For bluring identity for privacy: FN should be minimised
Criterion value: thr
equality of opportunity
To **evaluate inclusion** as you're developing and testing your machine learning model
the true positive rate is identical between groups

How to find errors in your dataset using Facets
So there's two parts to Facets, **Overview** and **Dive**
Common data issues that can hamper machine-learning
unexpected feature values,
features with high percentages of missing values,
features with unbalanced distributions,
features distribution skew between datasets.
Summary
Summary
Machine learning at Google is about providing a **unied plaorm** for **managed datasets, a feature store, a way to build, train, and deploy** machine learning models **without writing a single line of code**, providing the **ability to label data**, create **Workbench notebooks using frameworks such as TensorFlow, SciKit Learn, Pytorch, R**, and others. Veex AI Plaorm also includes the ability to **train custom models, build component pipelines, and peorm both online and batch predictions**. This course reviews the ve phases of conveing a candidate use case to be driven by machine learning, and considers why it is impoant to not skip the phases. We end with a **recognition of the biases** that machine learning can amplify and how to recognize them.
Module 1: What It Means to be AI-First
Machine learning is a way to use **standard algorithms** to **derive predictive insights** from data and make **repeated decisions**.
AI is a discipline that has to do with the **theory and methods** to build machines that think and

act like humans. ML is a **toolset** that you can use machine learning to solve ceain kinds of AI Problem

Given the choice between **more data and more complex** models, spend your energy collecting more data. That means, collecting **not just more quantity, but also more variety**

Module 2: How Google Does Machine Learning

Avoid these ten ML pitalls:

1. You thought training your own ML algorithm would be faster than writing the soware
2. You don't collect enough data
3. You haven't looked at the data but assume it's ready to use
4. You forgot to put and keep humans in the loop
5. Your product launch focused on the ML algorithm
6. You optimized your ML algorithm for the wrong thing
7. You don't know if your ML is improving things in the real world
8. You didn't use pre-trained ML algorithm
9. You only trained your ML algorithm once
10. You designed your own perception or NLP algorithm

Module 3: Machine Learning Development with Veex AI

At a high level, machine learning development addresses **framing the problem, preparing the data, experimenting, and evaluating** the model.

You **build and compare many dierent models to determine which works best**. For example, random forests, suppo vector machines, and logistic regression are just three models you could use.

Moving from **experimentation to production** requires **packaging, deploying, and monitoring** your model.

Veex AI provides unied denitions/implementations of four concepts:

1. A dataset can be **structured or unstructured**. It has **managed metadata** including **annotations**, and can be stored anywhere on **Google Cloud**.
2. A **training pipeline** is a series of **containerized steps** that can be used to train an ML model using a dataset. The **containerization** helps with **generalization, reproducibility, and auditability**.
3. A model is an ML model with metadata that was built with a Training Pipeline or directly loaded (only if it is in a compatible format).
4. An endpoint can be invoked by users for online predictions and explanations. It can have one or more models, and one or more versions of those models, with disambiguation carried out based on the request.

Tools to **interact with Veex AI** include **client libraries, VM images, REST API, and containers**.

Module 4: Machine Learning Development with Veex Notebooks

Veex AI Workbench provides two Jupyter notebook-based options for your data science workow: **managed notebooks and user-managed notebooks**.

Managed notebooks instances are **Google-managed environments** with integrations and features that help you set up and work in an **end-to-end notebook-based production environment**. Managed notebooks let you access your data **without leaving the JupyterLab inteace**.

User-managed notebooks are **Deep Learning VM Images instances** that are **heavily customizable** and are ideal if you need a lot of control over your environment. User-managed notebooks can be a good choice for users who require extensive customization or who need a lot of control over their environment.

Module 5: Best Practices for Implementing Machine Learning on Veex AI

Google has recommended best practices for:

- Machine learning development
- Data preprocessing
- Machine learning environment setup
- Model deployment and serving
- Model monitoring
- Veex AI Pipeline
- Aifact organization

Machine learning development best practices

Preparing and storing data:

- Regardless of your data's origin, extract data from the source systems and conve to the format and storage (separate from the operational source) optimized for ML training.
- **Structured** data:
  - Store tabular data in **BigQuery**
    - Use Veex AI **Feature Store** with structured data
- **Unstructured** data:
  - Store image, video, audio, and unstructured data in **Cloud Storage**
    - Use Veex **Data Labeling** to provide labels

Training a model:

- For **small** datasets, train a model within

the **Notebooks** instance.
● For **large** datasets, distributed training, or scheduled training, use the **Veex training service.**
● Veex AI Training provides a set of pre-built algorithms that allows users to bring their custom code to train models.

Best practices for **Explainable** AI:
● Oers **feature aributions** to provide insights into why models generate predictions.
● Details the impoance of each feature that a model uses as input to make a prediction.
● Suppos custom-trained models based on tabular and image data.

Best practices for using **Workbench** Notebooks:
● Use Notebooks to evaluate and understand your models. In addition to built-in common libraries like scikit-learn, Notebooks oers **What-if Tool (WIT) and Language Interpretability Tool (LIT).**

Best practices for using Veex AI **TensorBoard**:
● Veex AI TensorBoard service lets you **track experiment metrics** such as **loss and accuracy over time**, visualize a model graph, project embeddings to a lower dimensional space, and much more.

Data preprocessing best practices
● Use **BigQuery** to process **tabular** data and use **Dataow** to process **unstructured** data.

ML environment setup best practices
● Use Notebooks for **experimentation and development**.
● Create a Notebooks instance for **each team member**.
● Help **secure PII** in Notebooks.
● **Store prepared data and your model in the same project**.
● Optimize peormance and cost.
● Use Veex **SDK** for **Python**.

Module 6: Responsible AI development

Unconscious biases exist in data. They can exist in both **collecting and labeling data**. These biases will then be reected in your ML, and aect the entire pipeline.

A checklist for bias-related issues:
● Biometrics ● Religion ● Income ● Health
● Race ● Sexual orientation ● Country ● Language
● Skin color ● Socioeconomic status ● Location ● Dialect

Use the **What-If too**l to help you diagnose **fairness** issues in your data, in your labels, and in the eects of predictions.

A **confusion matrix** helps in understanding **inclusion** and how to introduce inclusion across dierent kinds of groups across your data.

Equality of oppounity is an approach that strives to **give individuals an equal chance of the desired outcome**. Incorporating this approach into your machine learning system gives you a way to scrutinize your model in order to discover possible areas of concerns. Once you identify oppounities for improvements, you can now make the necessary adjustments to strike a **beer tradeo between accuracy and non-discrimination**—which, in turn, could make your machine learning model **more inclusive**.

The **Facets** tool can help you make machine learning **more inclusive.**
● Facets **Overview** automatically gives you a **quick understanding of the distribution of values across the features** of their datasets.
● Facets **Dive** provides an **easy-to-customize**, **intuitive** inteace for exploring the **relationship between the data points across the dierent features of a dataset**.

Launching into Machine Learning

Improve data quality

data extraction

you retrieve data from various sources.

Those sources can be **streaming in real time or batch**.

For example, you may extract data from a Customer Relationship Management system, or **CRM**, to analyze customer behavior.

This data may be **structured** where it is in a given format such a **CSV,a text, JSON, or XML** format.

Or, you may have unstructured source data where you may have **images** of your customers or **text comments** from your chat sessions with your customers.

Or, you may have to extract **streaming** data from your company's transportation vehicles that are equipped with sensors that **transmit data in real time**.

Other examples of unstructured data may include **books and journals,documents, metadata, health records, audio and video**.

data analysis

you analyze the data you've extracted.

For example, you can use **Exploratory Data Analysis, or EDA**, which involves using **graphics** and **basic sample statistics** to **get a feeling for what information might be obtainable** from your dataset.

Sns.jointplot(): Bivariate plot+univariate plot in the margins

You look at various aspects of the data such as **outliers or anomalies, trends, and data distributions**, all while attempting to identify those features that can aid in increasing the predictive power of your machine-learning model

Lab:

- Df.info(): # of nulls, column name, data type
  - Category (string) feature data type is 'object'
- Print(df, 5): prints first and last 5 rows
- Df.describe(): gives summary statistics. For numeric features
- Df.groupby('col name').first(): First entry after grouping
- Df.rename(columns = {old1:new1, old2:new2, ...}): rename the column names

Data preparation

includes data **transformation**, which is the process of **changing or converting** the **format, structure, or values** of data you've extracted into another format or structure.

There are many ways to prepare or **transform** data from machine-learning model.

- For example, you may need to perform **data cleansing** where you need to remove **superfluous and repeated** records from log data.
- Or you may need to alter data types where a data feature was **mistyped** and you need to convert it.
- Or you may need to **convert categorical data to numerica**l data. Most ML models require categorical data to be in numerical format, but some models work with **either numeric or categorical** features while others can **handle mixed** type features.
  - Df_dummies = Pd.get_dummies(df, drop_first=true) : categorical to OHE
  - Df = pd.concat([df, Df_dummies], axis = 1) Concat old and new features
  - Df = df.drop([], axis = 1) Remove old categorical features

To determining the **data quality levels**, organizations typically perform **data asset inventories** in which the **relative accuracy, uniqueness, and validity** of data is measured.

attributes related to data quality.

- Data **accuracy**
  - relates to whether the data value was stored or an object or the correct values.
- Data consistency:
  - To be correct, data values must be the **right** value and must be **represented in a consistent and unambiguous form**.
- Timeliness
  - can be measured as the **time between when information is expected and when it is readily available for use**.
- Data completeness
  - relates to whether all the intended data being produced in the data set is complete.Or, is any of the data missing?

ways to improve data quality (untidy/messy data).

- **resolve missing values**,
  - Missing values can **skew** your data
  - .Isnull().sum()
  - .Isnull().sum().values.sum() : All missing
  - .nunique() : # of unique values per column
  - Df.columns.tolist
  - Df = Df.apply(lambda x:x.fillna(x.value_counts().index[0])) : 'apply' applies function to the rows or columns of a df
- convert date time features to a **date/time format** if it is not in the correct format already,
  - Df[date] = pd.to_datetime(Df[date], format=)
    - Df[date].dt.year
    - Df[date].dt.month
    - Df[date].dt.day
    - If the data type of the data column is 'object', use above to convert it to datetime64
- **parse** the data/time features to get **temporal features** that allow you to create more insight in to your data.
  - We should also consider **parsing the date feature** in to three distinct feature columns: **year, month, and day**.
  - This would allow you to look at the **seasonality** of your data, to spot **trends**, and to also perform **time series related predictions**.
- You can **remove unwanted values** from a feature column.
- convert **categorical** feature columns to **one-hot encodings**.
- Another data quality issue is unwanted screen characters in a column.
  - Now the intent of the **less than sign** is valid, the researcher wants to show models less than 2006. But we **cannot leave this less than sign in our feature column**.
    - There are many ways to deal with this. We could create **year buckets**
- Improving data quality can be done **before and after data exploration**.
- We can **explore and clean data iteratively**, as you will see the lab.The process does not have to be a **sequential process**.
- Data is said to be **messy or untidy** if it is **missing** attribute values,contains **noise or outliers**, has **duplicates**, **wrong** data, upper-lower case column names and is essentially **not ready for ingestion** by a machine-learning algorithm.

Improving Data Quality

Launch Vertex AI Notebooks instance

i. In the Google Cloud Console, on the **Navigation Menu**, click **Vertex AI > Workbench**. Select **User-Managed Notebooks**.
ii. On the Notebook instances page, click **New Notebook > TensorFlow Enterprise > TensorFlow Enterprise 2.6 (with LTS) > Without GPUs**.
iii. In the **New notebook** instance dialog, confirm the name of the deep learning VM, if you don't want to change the region and zone, leave all settings as they are and then click **Create**. The new VM will take 2-3 minutes to start.
iv. Click **Open JupyterLab**.
A JupyterLab window will open in a new tab.
v. You will see "Build recommended" pop up, click **Build**. If you see the build failed, ignore it.

Clone course repo within your Vertex AI Notebooks instance

i. In JupyterLab, to open a new terminal, click the **Terminal** icon.
ii. At the command-line prompt, run the following command:
iii. To confirm that you have cloned the repository, double-click on the training-data-analyst directory and ensure that you can see its contents.
The files for all the Jupyter notebook-based labs throughout this course are available in this directory.

Improve data quality

i. In the notebook interface, navigate to **training-data-analyst > courses > machine_learning > deepdive2 > launching_into_ml > labs**, and open **improve_data_quality.ipynb**.
ii. In the notebook interface, click **Edit > Clear All Outputs**.
iii. Carefully read through the notebook instructions and fill in lines marked with #TODO where you need to complete the code as needed.

What is exploratory data analysis

- EDA is an approach to analyzing data sets to **summarize their main characteristics**, often with **visual** methods.
- A **statistical model can be used or not**,but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.
- Exploratory data analysis is a loosely defined term that involves using **graphics and basic sample statistics** such as mean and median or standard deviation to get a feeling for what information might be obtainable from your data set.
- EDA is a set of techniques that allows analysts to quickly look at data for trends, outliers and patterns.
- The eventual goal of EDA is to obtain theories that can later be tested in the modeling step. Exploratory data analysis is an approach for data analysis that employs a variety of techniques, mostly graphical, **to maximize insight into a data set, uncover underlying structure, extract important variables, detect outliers and anomalies, test underlying assumptions, develop parsimonious models and determine optimal factor settings**.
- The three popular data analysis approaches
  - Classical data analysis:
    - the data collection is followed by the **imposition of a model, normality, linearity**, for example, and the analysis, estimation and testing that follows are focused on the **parameters of that model**.
  - exploratory data analysis
    - the data collection is **not followed by a model imposition**.
    - Rather, it is followed immediately by analysis with a goal of inferring **what model would be appropriate**.
    - Unlike the classical approach, the exploratory data analysis approach does not impose deterministic or probabilistic models on the data.
    - On the contrary, the EDA approach allows the data to suggest admissible models that best fit the data.
  - Bayesian data analysis:
    - the analyst attempts to answer research questions about **unknown parameters** using **probability** statements based on **prior data**.
    - They may bring their own **domain knowledge and/or expertise** to the analysis **as new information is obtained**, so that's the purpose of Bayesian analysis, is **to determine posterior probabilities based on prior probabilities and new information**.
    - **Posterior** probabilities is a the probability an event will happen after **all evidence or background information** has been taken into account.
    - **Prior** probability is the probability an event will happen **before you've taken adding new evidence** into account.
- EDA techniques are generally **graphical**.
- They include **scatterplots, boxplots, histogranoams**, **regplot**, et cetera.
- In the real world, **data analysts freely mix elements of all of the above three approaches and other approaches**, as well.

How is EDA used in machine learning

- For exploratory data analysis, the focus is on the **data, its structure, outliers and models suggested by the data**
- **EDA type:**
  - Univariate analysis
    - It **doesn't** deal with **causes or relationships**, unlike **regression**, and its major purpose is to **describe**.
    - It takes the data, it **summarizes** that data, and it **finds patterns** in the data.
    - types
      - categorical
        - numerical EDA :Pandas' crosstab function
        - visual EDA :Seaborn's countplot function.
      - Continuous
        - numerical EDA :Pandas' **describe** function,
        - visual EDA : **boxplots**, **distribution** plots and kernel density estimation plots, or **KDE** plots in Python, using Matplotlib or using Seaborn.
  - Bivariate analysis:
    - to find out if there is a relationship between two sets of values
    - Types:
      - Category to category:
        - One of the most powerful features of Seaborn is the ability to easily build **conditional plots**.
        - This lets us see **what the data looks like when segmented by one or more variables**.
        - The easiest way to do this is through the **factor plot** method, which is used to draw a categorical plot up to a facet grid.
      - Category to continous:
        - Seaborn's **jointplot** function draws a plot of two variables with bivariate and univariate graphs.
      - Continuous to category:
        - Seaborn's factorplot map method can map a factorplot onto a KDE, distribution or boxplot chart.
          - Sns.factorplot().map(sns.kde)

Data analysis and visualization

- the purpose of an EDA is to **find insights** which will serve for **data cleaning, preparation, or transformation**, which will ultimately be used in a machine learning algorithm.
- Histogram:
  - A histogram displays the **shape and spread of continuous sampled data**.
  - Series.hist(bins=50)
- Scatter plot
  - Plt.scatter()
- heatmap function to show correlations (**multivariate graphical analysis**)
  - uses a system of **color coding** to represent different values
  - Sns.heatmap(Df.corr())

Exploratory Data Analysis Using Python and BigQuery

Machine Learning in Practice

supervised learning

Rows: examples

Columns: features

Regression: labell is continuous

Classification : label is discrete

If data is **not labeled: clustering algorithms** to discover interesting properties of the data

Linear regression

decision boundary

Line in higher dimension is hyperplane

Regression: MSE

Classification : cross-entropy

A categorical feature can be embedded into a continuous space.

Both of these problem types, r**egression and classification,** can be thought of as **prediction problems** in contrast, to **unsupervised** problems,which are like **description** problems

very common **source of structured data** for machine learning is your **data warehouse**

Logistic regression

In coin-flip if we use linear regression with the **standard mean square error** or loss function, our predictions could end up being **outside the range of zero and one**.

Simple tricks, like **capping the predictions at zero or one, would introduce bias**, so we need something else.

Converting this from linear regression to logistic regression can solve this dilemma.

The input into the sigmoid, normally the output of linear regression, is called the **logit**

we are performing a **nonlinear transformation on our linear model.**

Unlike mean squared error, the sigmoid never guesses 1.0 or 0.0 probability. This means that in gradient descent's constant drive to get the loss closer and closer to zero, it will drive the weights closer and closer to plus or minus infinity **in the absence of regularization** which can lead to problems

the output of a sigmoid: It is a **calibrated probability estimate**.

Beyond just the range, the sigmoid function is the **cumulative distribution function of the logistic probability distribution**, whose quantile function is the inverse of the logic which models the long odds. Therefore, mathematically, **the opposite of a sigmoid can be considered probabilities**.

In this way, we can think of calibration as the fact the outputs are real-world values like probabilities. This is in contrast to uncalibrated outputs, like an embedding vector, which is internally informative,but the values have no real correlation.

Lots of output activation functions, in fact, an infinite number, could give you a number between zero and one, but **only this sigmoid is proven to be a calibrated estimate of the training data set probability of occurrence**.

Using this fact about the sigmoid-activation function, we can **cast binary-classification problems into probabilistic problems**.

This paired with a **threshold** can provide a lot **more predictive power** than just a simple binary answer

regularization is important in logistic regression because driving **loss to zero** is **difficult and dangerous**.

Due to the equation of the sigmoid, the function **asymptotes to zero when the logit is negative infinit**y and to one when the logit is positive infinity. To get the logits to negative or positive infinity, the manager of the weights is increased and increased, leading to **numerical-stability problems, overflows and underflows**.

Also near the asymptotes, as you can see from the graph, the sigmoid function becomes **flatter and flatter**. This means that the **derivative is getting closer and closer to zero**.

Saturation: Since we used the **derivative and back propagation to update the weights**, it is important for the **gradient not to become zero or else training will stop**.

when **all activations end up in these plateaus**, which leads to a v**anishing-gradient problem** and makes training difficult.

If you use **unregularized logistic regression**, this will lead to **absolute overfitting**, as the model tries to drive **loss to zero** on all examples and never gets there, the **weights** for each indicator feature will be driven to **positive infinity or negative infinity**.

Adding regularization to logistic regression helps keep the model simpler by having smaller parameter weights.

This **penalty term added to the loss function** makes sure that **cross entropy** through gradient descent doesn't keep pushing the weights from closer to closer to plus or minus infinity and causing numerical issues. Also with now **smaller logits,** we can now stay in the **less-flat portions of the sigmoid function**, making our **gradients less closer to zero** and thus allowing weight updates and training to continue.

regularization does not transform the outputs into calibrated probability estimate

The great thing about logistic regression is that it already outputs the **calibrated probability estimate,** since the sigmoid function is **accumulated distribution function of the logistic-probability distribution**. This allows us to **actually predict probabilities** instead of just binary answers

To counteract overfitting, we often do both **regularization and early stopping**.

For regularization, **model complexity increases with large weights**, and so as we tune and start to get larger and larger weights for rarer and rarer scenarios, we end up increasing the loss, so we stop.

L2 regularization will keep the **weight values smaller**, and **L1** regularization will keep the models **sparser** by dropping poor features.

To find the **optimal L1 and L2 parameter choices** during **hyper parameter tuning**, you are searching for the point in the **validation**-loss function where you obtain the lowest value.

any **less regularization increases your variants, starts overfitting and hurts generalization**, and any **more regularization increases your bias, starts underfitting and hurts your generalization**.

Early stopping stops training when overfitting begins.

As you train your model, **you should evaluate your model on your validation data set every so many steps, epochs, minutes**, et cetera.

As training continues, **both the training error and the validation error should be decreasing**, but **at some point the validation error might begin to actually increase**.

It is at this point that the **models begin to memorize the training data set and lose its ability to generalize to the validation data set**, and most importantly, to the **new data** that we will eventually want to use this model **for**.

Using early stopping would stop the model at this point and then **back up and use the weights from the previous step before it hit validation-error-inflection point**.

early stopping is an approximate equivalent of L2 regularization and is often used in its place because it is **computationally cheaper**.
Even though **L2 regularization and early stopping seem a bit redundant**, for **real-world systems, you may not quite choose the optimal hyper parameters**, and thus **early stopping can help fix that choice for you**
A **simple threshold of a binary-classification** problem would be all probabilities less than or equal to 50 percent should be a no,and all probabilities greater than 50 percent should be a yes.
However, for studying real-world problems, we may want **a different split, like 60/40, 20/80, 99/1**, et cetera, depending on how we want our **balance of our type one and type two errors**, or in other words, our balance of false positives and false negatives.
In ROC, As we **lower the threshold**, we are likely to have **more false positives**, but we'll also **increase the number of true positives** we find.
there's an efficient **sorting-based algorithm** to do this
AUC helps you **choose between model**s when you **don't know what decision threshold is going to ultimately used**.
AUC is that it's **scale invariant and classification-threshold invariant**
People sometimes also use AUC (TPR vs FPR) for the **precision-recall curve**, or more recently, **precision-recall-gain curves**, which just use different combinations of the four prediction outcomes as metrics along the axes
However, treating this only as an **aggregate measure** (AUC) can **mask some effects**.
- For example, a small improvement in AUC might come by doing a better job of ranking very unlikely negatives as even still yet more unlikely, which is fine, but potentially not materially beneficial.

When we evaluate our logistic-regression models, we need to make sure predictions aren't biased.
- there should be an **overall shift in either the positive or negative direction**.
- A simple way to **check the prediction bias** is to compare the **average value predictions made by the model over a data set to the average value of the labels** in that data set. If they are not relatively close, then you might have a problem.

even **zero bias alone does not mean everything in your system is perfect**, (but it is a great sanity check)
If you have bias, you could have an **incomplete feature set, a buggy pipeline, a biased training sample**, et cetera.
calibration scatter plot: log-log scale
- **Use calibration plots of bucketed bias to find slices of data where your model performs poorly**
- we're comparing the **bucketized log odds predicted** to the **bucketized log odds observed**
- This can happen when parts of the data space is not well represented or because of noise or because of overly strong regularization.
- You can bucket by literally **breaking up the target predictions**, or we can bucket by **quantiles**.
- Why do we need to **bucket prediction** to make calibration plots when predicting probabilities? For any given event, the true label is either zero or one. But our prediction values will always be a probabilistic guess, we're always off, but if we group enough examples together, we'd like to see that **on average the sum of the true zeroes and ones is about the same as the mean probability we're predicting**.

important in performing logistic regression:
- adding penalty terms to the objective function + early stopping (regularization),
- tuned threshold
- predictions are unbiased
  - average of the predictions is very close to the average of observations.
  - look at slices of data (calibration plot)

Training AutoML Models Using Vertex AI
- ML vs DL
  - All machine learning starts with a business requirement, academic requirement or problem that you are trying to solve.
  - wrangle
    - we explored previously when we introduced an untidy dataset and walked you through **making it tidy enough to feed into a machine learning mode**l.
      - how to resolve missing values,
      - convert the date feature column to a **date time format**,
      - rename a feature column,
      - remove a value from a feature column,
      - create one hot coding features
      - understand temporal feature conversions.
    - Pipeline:
      - business understanding
      - Data wrangling
      - Data visualisation
      - Data preprocessing
      - Model training
      - Model validation
      - Deployment (to production)
  - a point-and-click solution
  - The data scientist has domain knowledge, but has limited experience putting a machine learning model into production
  - data analyst knows SQL, but has no ML knowledge
  - software developer knows Java, no ML knowledge
  - difference between machine learning and statistics:
    - In machine learning, you have and want **lots of data**. You'll want to use your dataset for **training, testing and validatin**g, so you'll have to **split it up**.
    - In statistics, you **don't need to split** your data.
    - You take what you are given and you use it.
    - Also in machine learning, you keep the outliers or those data points that stray outside of all the other data points, and then you build models for them.
    - You'll want to use those outliers to train your model with so that you **get a more holistic picture of your data**.
    - Statistics on the other hand is about keeping the data that you have and **getting the best results** out of the data that you have, so it is not uncommon **to toss the outliers out**.
    - Also, in statistics, we examine the r**elationship between variables**.
    - In machine learning, we want to **predict** theY given X .
    - A major difference
      - **data preparation**

Machine learning **doesn't require explicit commands to find patterns in data**.
In standard statistics, we need to know **variables and parameters** beforehand
Hypothesis
Also, while you need a hypothesis or theory to test in statistics, there's no **hypothesis testing** required in machine learning.
Type of data
In statistics, your data is **linear**, while in machine learning your data can be **multi-dimensional or non-linear** in nature.
Training
In machine learning, you need to train, meaning the model or algorithm needs **to learn the patterns in the data** and it learns those patterns through training.
Goal
Machine learning is better for **making inferences or predictions**, while statistics is generally better for **testing inferences and hypotheses.**
Scientific question:
The question for statistics is **how and why** something happens, for example, the **relationship or correlation** between the variables. In machine learning, the goal is **what** will happen if I give the algorithm data it has not seen before? Will it make a strong prediction?
ML and DL: Within the **subset of machine learning** methods, **deep learning** is usually implemented as a form of **supervised** learning
Data requirement
Deep learning requires **large datasets**, while machine learning allows you to train on **smaller** datasets.
Accuracy
Because deep learning uses large datasets to glean patterns, it provides **higher accuracy** than other methods
Training time
deep learning **takes longer to train a model**
Hardware dependeny
you can train your **ML model** on a **CPU**, while you'll need a **GPU** to train a **deep learning** model given the large data size
hyperparameter tuning
more control over tuning of the hyperparameters with deep learning than with other forms of machine learning.
weight is a parameter. Number of layers is a hyperparameter
machine learning is a subfield of artificial intelligence
The goal of ML is to make computers **learn from the data** that you give them
Instead of writing code that describes the action the computer should take, your code provides an algorithm that **adapts based on examples of intended behavior**
The resulting program consisting of the **algorithm** and associated **learned parameters** is called a **trained mode**l.
What is automated machine learning?
The process of applying machine learning to real-world problems is time consuming.
Automated Machine Learning workflow = pipeline
traditional components of a machine learning pipeline
You need to get the data ready.
Perform feature engineering.
Train and tune your model.
Serve your model.
Understand it.
Present it to any Edge devices.
Monitor your model and
manage it.
Vertex AI automates the following components in the machine learning pipeline:
data readiness, feature engineering, training and hyperparameter tuning, model serving, explainability and interpretability and the ability to deploy to Edge devices.
\+ Vertex AI's Vizier optimization for hyperparameter tuning, managed data sets, feature store
AutoML regression model
XYZ Company has just **defined a business use case,established the success criteria and wants to deliver an ML model to production**.
data scientist has **domain knowledge** of the problem and **machine learning experience**, but they have l**ittle experience bringing a model to production**
point and click solution: with Vertex AI, the team at XYZ Company can load data, generate statistics on the data and build and train their model, without writing a single line of code.
Vertex AI, which is a **dashboard** with **features and services** that allow them to perform various tasks in the machine learning pipeline.
AutoML supports the following data types; image, tabular, text and video.
AutoML tables will **automatically define your problem and model to build**, based on the **data type of your target column**, so if your target column contains **numerical** data, AutoML tables will build a **regression** model. If your target column is **categorical** data,AutoML tables will detect the number of classes and determine if you need to build a **binary or multiclass** model.
Vertex AI is a unified platform
You can use Vertex AI to manage the following stages in the ML workflow:
Create a dataset and upload data,
train an ML model on your data -- train the model, evaluate model accuracy, tune hyperparameters, custom training only.
Upload and store your model in Vertex AI.
Deploy your trained model to an **endpoint** for **serving predictions**.
Send **prediction requests** to your endpoint.
Specify a **prediction traffic split** in your endpoint, and
manage your models and endpoints.
If you want to code, you still can by building a custom solution (Notebooks, Pipelines)
AutoML lets you create and train a model with **minimal technical effort**.
Even if you want the flexibility of a custom training application, you can use AutoML to **quickly prototype** models and explore new datasets **before investing in development**.
Custom training lets you create a training application **optimized for your targeted outcome**. You have **complete control over training application functionalit**y. You can target any **objective, use any algorithm, develop your own loss functions or metric**s, or do any other **customization**.

- AutoML or Custom training
  - No data science expertise is required to use AutoML. Custom training requires data science expertise to develop the training application, and also to do some of the data preparation, like feature engineering.
  - AutoML is codeless, so no programming ability is needed. Custom training requires programming experience to develop the training application.
  - When it comes to training a model, **AutoML saves time** because it requires **less data preparation and no development**. Custom training requires more data preparation and time to develop the training application.
  - One of the major differentiators is your machine learning **objectives**. With AutoML, you must target one of the predefined objectives, such as regression classification and forecasting,which are supervised learning tasks. There are also predefined objectives for image, text and video. You can use custom training for a variety of objectives.
  - In terms of optimizing model performance with hyperparameter tuning,AutoML does some automated hyperparameter tuning, but you can't modify the values used. With custom training, you can tune the model during each training run for experimentation and comparison.
  - If you need more control over aspects of your **training environment**,such as specifying **compute engine type, disk size, machine learning framework, or the number of nodes used for hyperparameter tuning**, then custom training is the best option.
  - Both AutoML and custom training have the **same limits on managed datasets**,and data size limitations vary, depending on the type of datasets. However, for **unmanaged** datasets, where the data comes from **Google cloud storage or BigQuer**y and is **not uploaded as a CSV** file, there is **no limit on data size**,as these are considered unmanaged datasets

Training an AutoML Classification Model - Structured Data
- Previously, models trained with AutoML and custom models were accessible via separate services.
- The new offering combines both into a single API, along with other new products.
- Vertex AI includes many different products to support **end-to-end ML workflows**

Evaluate AutoML models
- Evaluating AutoML models begins with understanding how AutoML Tables uses your dataset.
- Your dataset will be split into training, validation,and testing sets.
- training set
  - This is the data your model sees during training.
  - It's used to learn the **parameters** of the model,namely the **weights of the connections between nodes of the neural network**.
- validation set
  - sometimes called the "**dev**" set, is also used during the **training process**.
  - After the model learning framework incorporates training data during each iteration of the training process, it uses the model's performance on the validation set to **tune the model's hyperparameters** which are variables that specify the model's structure.
  - If you try to use the training set to tune the hyperparameters,it's quite likely the model will end up overly focused on your training data and have a **hard time generalizin**g to examples that don't exactly match it.
  - Using a **somewhat novel dataset to fine tune** your model structure,means your model will generalize better.
- test set
  - is **not involved in the training process** at all.
  - Once the model has completed its training entirely, AutoML Tables uses the test set as an entirely new challenge for your model.
  - **The performance of your model on the test set is intended to give you a pretty good idea of how your model will perform on real-world data**.
- There's **no perfect answer on how to evaluate your model.**
- Evaluation metrics should be considered **in context with your problem type** and **what you want to achieve with your model**.
- MAE
  - because it uses absolute values, MAE **doesn't consider the relationship's direction** nor indicate **underperformance** or **overperformance**.
- MAPE (Mean absolute percentage error)
- RMSE
  - is **more sensitive to outliers than MAE**. So if you're concerned about large errors,then RMSE can be a more useful metric to evaluate.
- RMSLE.
  - The root mean squared **logarithmic** error metric
  - natural logarithm of the predicted and actual values **plus one**
  - RMSLE **penalizes under-prediction more heavily than over-prediction**.
- R squared
  - is the **square of the Pearson correlation coefficient** between the observed and predicted values.
  - coefficient of determination
  - 0 <= <=1
- Model feature **attributions**
  - how much each feature **impacts** a mode
- Log loss
  - this is the **cross-entropy between the model predictions and the target** values.
  - This ranges from zero to infinity, where a **lower value indicates a higher-quality model**.
  - Mathematically, log loss is the negative average of the log of the corrected predicted probabilities for each instance.
- F1
  - is a useful metric if you're looking for a balance between precision and recall and there's an **uneven class distribution**.
- Precision and recall
  - how well your model is **capturing information and what it's leaving out**
- In addition to evaluating models via the **Web UI**, evaluations can be made via **REST** and the **command-line** and using **Python**.
- Endpoints are **machine learning models made available for online prediction requests**.
  - Endpoints are useful for **timely** predictions for many users; for example, in response to an application request.
  - You can also request **batch** predictions if you **don't need immediate results**.
  - **Batch** prediction
    - is useful for **making many prediction requests at once**.

Batch prediction is **asynchronous**, meaning that the model will **wait until it processes all of the prediction request**s before returning a CSV file or a **BigQuery Table** with prediction values.

Online prediction,

deploy your model to make it available for prediction requests using a **REST API**.

Online prediction is **synchronous** or **real time**,meaning that it will **quickly return a prediction**, but **only accepts one prediction request per API call**.

Online prediction is useful if your **model is part of an applicatio**n and **parts of your system are dependent on a quick prediction turnaround**.

BigQuery Machine Learning: Develop ML Models Where Your Data Lives

Vertex AI AutoML data requirements

maximum data set size is 100 gigabytes

1,000 < row < 100M.

target column must be categorical (2-500) or numerical.

2 < column < 1000

Google's BigQuery is **more than just a data warehouse**.

It can provide decision-making guidance through **predictive analytics** by using its machine-learning tool, **BigQuery ML**.

you can **create and train a model without ever exporting data out of BigQuery**

Google BigQuery ML is **a set of SQL extensions** to support machine learning.

to do **custom modeling** without BigQuery ML, there is an increased complexity and multiple tools are required.

Custom modeling also **reduces speed**, moving and formatting large amounts of data for **Python based ML frameworks takes longer than model training in BigQuery**.

BigQuery ML **speeds up the time to production**, makes **development work much easier**, and **automates** a number of the steps in the ML **workflow**.

BigQuery ML will **import and preprocess data, split data, build the model and deploy the model.**

The team sees that with BigQuery ML all they really have to do is **have the data in BigQuery, identify a use case, write a short piece of SQL code, and they're ready to deliver their model.**

BigQuery ML is a middle ground between using **pretrained** models and building your **own TensorFlow model with Vertex AI platform.**

Exploratory data analysis or **EDA in a Jupyter Notebook, prototyping and scaling out to a managed service** is a time-consuming process.