# https://github.com/sathishvj/awesome-gcp-certifications/blob/master/professional-machine-learning-engineer.md

<https://towardsdatascience.com/a-comprehensive-study-guide-for-the-google-professional-machine-learning-engineer-certification-1e411db4d2cf>

<https://cloud.google.com/architecture/ml-on-gcp-best-practices>

<https://cloud.google.com/architecture/best-practices-for-ml-performance-cost>

<https://cloud.google.com/architecture/architecture-for-mlops-using-tfx-kubeflow-pipelines-and-cloud-build>

# Section 0: Google Products

1. DataFusion and DataPrep: both are codeless
   1. Datafusion is more designed for data ingestion from one source to another one, with few transformations.
   2. Dataprep is more designed for data preparation (as its name means), data cleaning, new column creation, splitting column. Dataprep also provide insight of the data for helping you in your recipes.
2. Cloud Build: Build, test, and deploy on our serverless CI/CD platform.
   1. Build software quickly
   2. Deploy across multiple environments such as VMs, Kubernetes or Firebase
   3. Keep your data at rest within a geographical region or specific location with data residency
3. Pre-packaged
   1. Bigquery ml
   2. Dialogflow
   3. Recommendations ai
   4. Cloud natural language api
   5. Translation api
   6. Speech to text
   7. Video ai
   8. Automl

# Section 1: ML Problem Framing

# Section 2: Architecting ML Solutions

## Designing reliable, scalable, and highly available ML solutions.

**Choosing appropriate ML services for the use case (e.g., Cloud Build, Kubeflow)**

Best practices for ML on GCP: <https://cloud.google.com/architecture/ml-on-gcp-best-practices>

1. ML environment setup: Vertex AI
   * Use vertex ai workbench user-managed notebooks for experimentation and development
   * Create a user-managed notebook instance for each team member. If a team member is involved in multiple projects, especially projects that have different dependencies, we recommend using multiple user-managed notebook instances
   * PII in a user-managed notebooks instance: production folder for ready to use data; trust folder for confidential data
   * Storing data:
     + When dealing with tabular data, google suggest to store all results in bigquery
       - For tf/kera, use tf.data.dataset reader
       - For tfx, use bigquery client
       - For dataflow, use bigquery I/O connector
       - For anyother framework, use python client library
     + Use vertex data labelling for unstructured data
     + Don’t store data in block storage area like file system / vms --> hard to manage and hard to optimize for performance
2. ML development
   1. Big query
   2. Cloud storage
   3. Vertex AI workbench notebook
   4. Vertex data labelling
   5. Vertex explainable AI
      * Vertex Explainable AI helps you understand your model's outputs for classification and regression tasks. Vertex AI tells you how much each feature in the data contributed to the predicted result.
      * Feature attribution methods:
        + Sampled shapley: for tableau data
        + Integrated gradients
          - **Differentiable models**, such as neural networks.
          - Recommended especially for models with large feature spaces.
          - Recommended for low-contrast images, such as X-rays
        + XRAI (eXplanation with Ranked Area Integrals)
          - XRAI assesses overlapping regions of the **image** to create a saliency map
   6. Vertex AI feature store
   7. Vertex AI tensorboard
   8. Vertex training
   9. Others: Google offers AutoML and BigQuery ML as pre-built training routine alternatives to Vertex AI custom-trained model solutions. The following table provides recommendations about when to use these options or Vertex AI.
      * Big query ML. Use this when
        + Tabular data, and
        + Comfortable with SQL, and
        + Models are available here: <https://cloud.google.com/bigquery-ml/docs/reference/standard-sql/bigqueryml-syntax-create#model_type>
      * AutoML (in the context of Vertex AI). Use this when
        + Models are available here:

Image data: classification / detection

Tabular data: regression / classification / forecasting

NLP data: classification / entity extraction / sentiment analysis

Video data: classification / object tracking / action recognition

* + - * Model can be served from google cloud
      * For Natural Language, Video, or Tables models, your model can tolerate inference latencies > 100ms.
    - Vertex AI custom trained models
      * The above solution can’t match your need, or
      * You are using TensorFlow Extended. TensorFlow Extended Trainer and Pusher steps both support running the step on Vertex A

1. Data processing
   1. Preprocessing
      * Use TFX when leverage TF
      * Use bigquery to process tabular data
      * Use Dataflow to process unstructured data
        + Apach beam to perform batch processing
        + Process both stream and batch
      * Use Dataproc if using spark
      * Use dataprep for UI-drive (codeless) data preparation
   2. Use managed datasets to link data to your models
      * 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.
2. Operationalized training
   1. Run your code in a managed service
      * Vertex training service
        + Requirements, either
          - A python training application to use with a pre-built container
          - A custom container image --> docker
      * Vertex AI pipelines
      * Optionally, you can run your code directly in a Deep Learning Virtual Machine container or on Compute Engine; however, we don't recommend this approach because the Vertex AI managed services provide automatic scaling and burst capability that is more cost effective.
   2. Regularly compute new feature values: feature store
3. Model deployment and serving
4. ML workflow orchestration
   * If you're using TensorFlow, use TensorFlow Extended to define your pipeline and the operations for each step, then execute it on Vertex AI's serverless pipelines system.
   * For all other frameworks, use the Kubeflow Pipelines with Vertex AI Pipelines. Use Vertex AI to launch and interact with the platform.
     + Use Kubeflow Pipelines for flexible pipeline construction
5. Artifact organization

Store your artifacts in these locations:

|  |  |
| --- | --- |
| Storage location | Artifacts |
| Source control repo | * Vertex AI Workbench user-managed notebooks * Pipeline source code * Preprocessing Functions * Model source code |
| Experiments and ML metadata | * Experiments * Parameters * Metrics * Datasets (reference) * Pipeline metadata |
| Vertex AI | * Trained models |
| Artifact Registry | * Pipeline containers * Custom training environments * Custom prediction environments |
| Vertex Prediction | * Deployed models |

1. Model monitoring

* Skew detection: This approach looks for the degree of distortion between your model training and production data
* Drift detection: In this type of monitoring, you're looking for drift in your production data. Drift occurs when the statistical properties of the inputs and the target, which the model is trying to predict, change over time in unforeseen ways. This causes problems because the predictions could become less accurate as time passes.
* Tune the thresholds used for alerting so you know when skew or drift occurs in your data.

**Component types (e.g., data collection, data management)**

<https://cloud.google.com/architecture/framework>

1. System design
2. Operational excellence
3. Security, privacy and compliance
4. Reliability
5. Cost optimization
6. Performance optimization

**Exploration/analysis**

**Feature engineering**

**Logging/management**

1. Cloud logging
   1. Fully managed, real-time log management with storage, search, analysis and alerting at exabyte scale.
2. Vertex ai experiments and tensorboard

Vertex AI TensorBoard provides various detailed visualizations, that includes:

* Tracking and visualizing metrics such as loss and accuracy over time
* Visualizing model computational graphs (ops and layers)
* Viewing histograms of weights, biases, or other tensors as they change over time
* Projecting embeddings to a lower dimensional space
* Displaying image, text, and audio samples

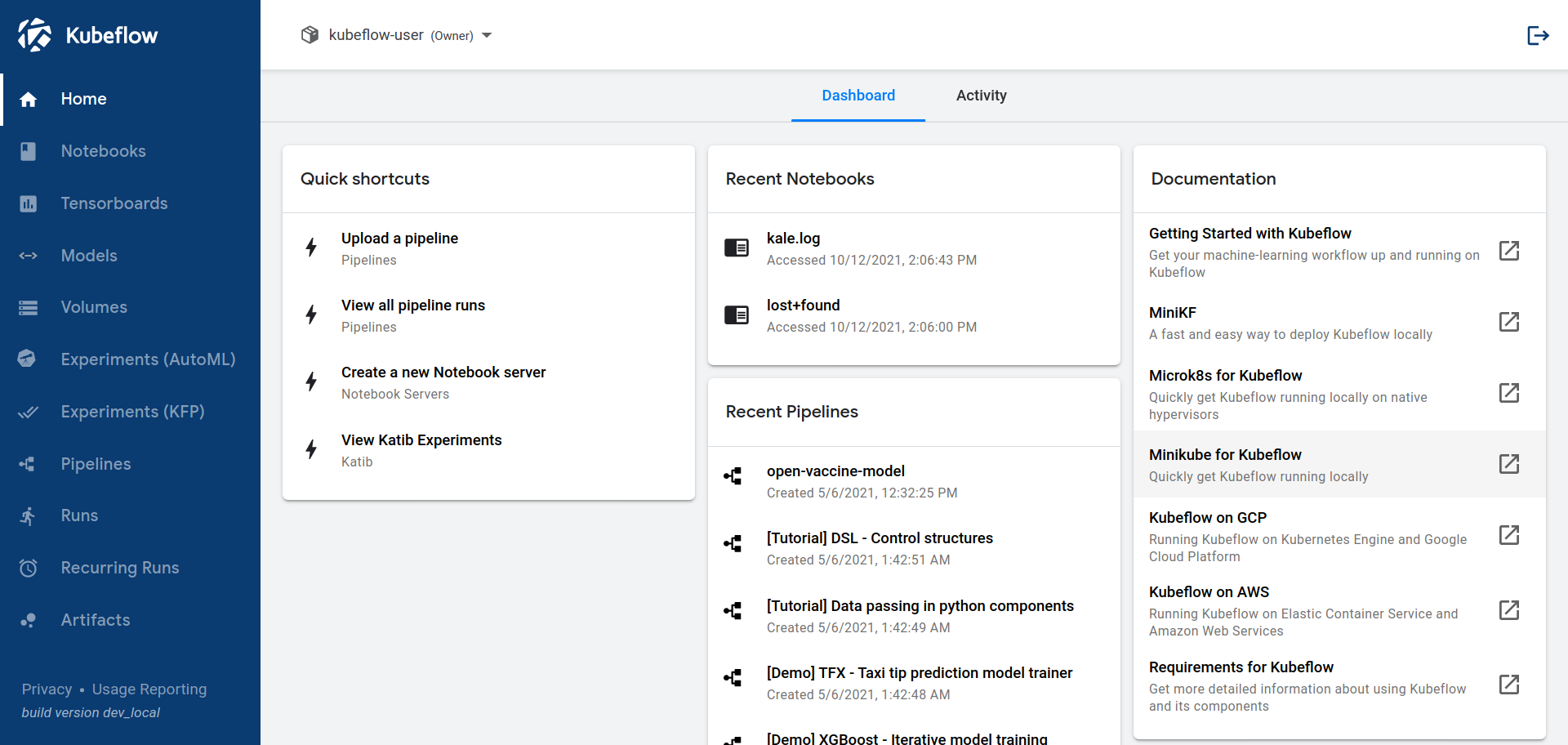
In addition to the powerful visualizations from TensorBoard, Vertex AI TensorBoard provides:

* + A persistent, shareable link to your experiment's dashboard
  + A searchable list of all experiments in a project
  + Tight integrations with Vertex AI services for model training
  + Enterprise-grade security, privacy, and compliance
  + With Vertex AI TensorBoard, you can track, visualize, and compare ML experiments and share them with your team.

1. Kubeflow UIs

The Kubeflow UIs include the following:

* Home: Home, the central hub to access recent resources, active experiments, and useful documentation.
* Notebook Servers: To manage Notebook servers.
* TensorBoards: To manage TensorBoard servers.
* Models: To manage deployed KFServing models.
* Volumes: To manage the cluster’s Volumes.
* Experiments (AutoML): To manage Katib experiments.
* Experiments (KFP): To manage Kubeflow Pipelines (KFP) experiments.
* Pipelines: To manage KFP pipelines.
* Runs: To manage KFP runs.
* Recurring Runs: To manage KFP recurring runs.
* Artifacts: To track ML Metadata (MLMD) artifacts.
* Executions: To track various component executions in MLMD.
* Manage Contributors: To configure user access sharing across namespaces in the Kubeflow deployment.



**Cloud logging: Fully managed, real-time log management with storage, search, analysis and alerting at exabyte scale.**

**Automation**

1. Vertex ai pipelines
2. Cloud build: Build, test, and deploy on our serverless CI/CD platform; build software quickly and deploy it on various environments
3. Cloud scheduler:
   1. Cloud Scheduler is a fully managed enterprise-grade cron job scheduler. It allows you to schedule virtually any job, including batch, big data jobs, cloud infrastructure operations, and more.
   2. Run your batch and big data jobs on a recurring schedule to make them more reliable and reduce manual toil.
   3. Automate your cloud infrastructure operations (I.e., when to turn on/off your VMs)

**Orchestration**

1. Cloud Composer: A fully managed workflow orchestration service built on **Apache Airflow**. Fully manage nature allows you to focus on authoring, scheduling, and monitoring your works
2. Vertex ai pipeline
3. Kubeflow pipelines

**Monitoring**

**Serving**

* Vertexi ai prediction
* Alternative, although not recommended, serving on compute Engine possibly with Deep Learning VM Images for your choice of API frameworks so you can have access to a GPU, serving on Cloud functions, app engine, Cloud Run, GKE

## Designing architecture that complies with security concerns across sectors/industries.

**Building secure ml systems**

Never trust user input. Secure a model api in the same way you would secure an api for a web application. Even for machine learning it is a good idea to process the data you receive and verify it corresponds to the input you are expecting to receive. Be aware of the concept of Adversarial Inputs.

**PII(personal identifiable information) /PHI (protected health information)**

# Section 3: Data Preparation and Processing

**Exploring data (EDA). Considerations include**

**Building data pipelines**

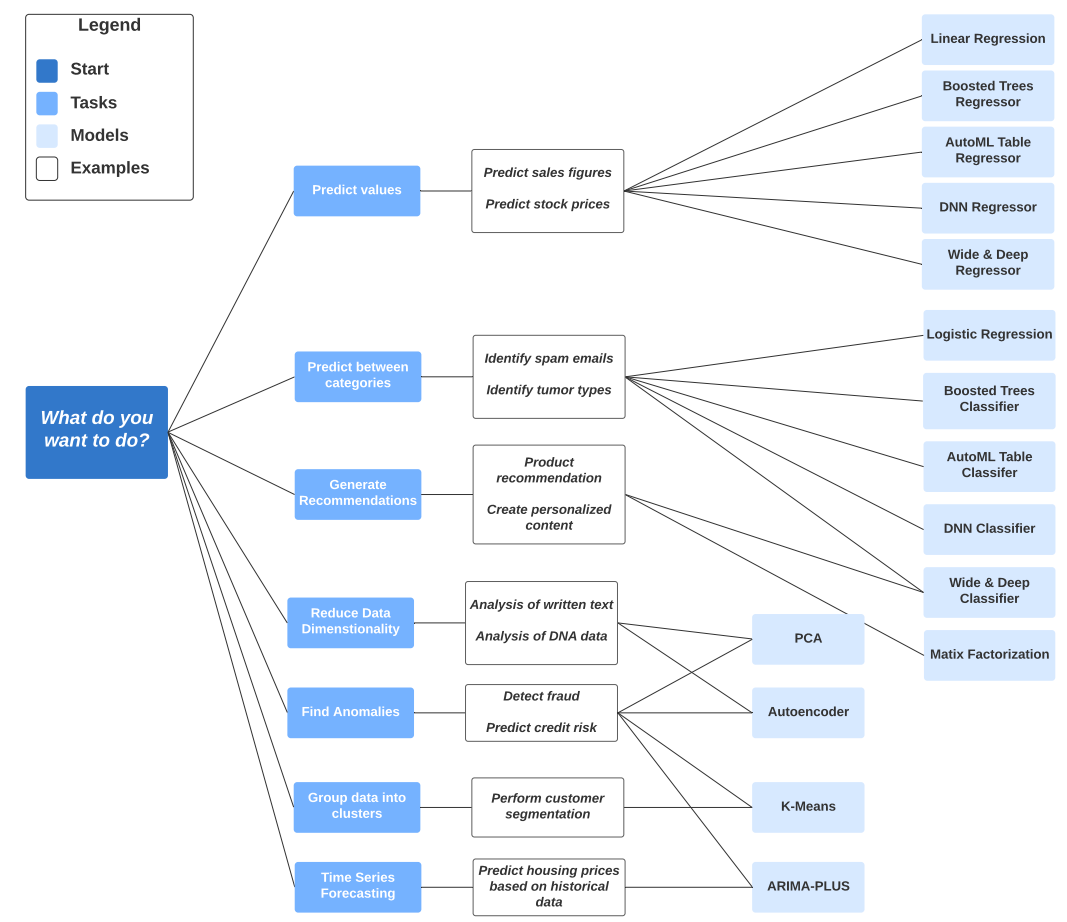
1. Organizing and optimizing training datasets
2. Data validation
3. Handling missing data
4. Handling outliers
5. Data Leakage
6. Creating input features (feature engineering)

**Ensuring consistent data pre-processing between training and serving**

1. Encoding structured data types
2. Feature selection
3. Class imbalance
4. Feature crosses
5. Transformations (TensorFlow Transform)

# Section 4: Developing ML models

**Building models**



**Training models**

**Testing models**

* Unit tests for model training and serving
* Model performance against baselines, simpler models, and across the time dimension
* Model explainability on Vertex AI

**Scaling model training and serving**

* Distributed training
  + Mirror: each GPU
  + TPU
  + MultiworkerMirrored: each node on each GPU
  + ParameterServer: on parameter server
  + CentalStorage
  + Default: Any GPU picked by TF --> no distribution
  + OneDevice: Specific GPU (one GPU only) --> no distribution
* Scaling prediction service (e.g., Vertex AI Prediction, containerized serving)

**QUESTION TO EXAM QUESTIONS**

Logging in Vertex AI:

* Container logging: issue within your container → useful for model debugging
* Access logging: issue with accessing and latency
* To enable/disable logging, need to re-deploy

BigQuery

* Ingest data:
  + Csv must be comma delimited
  + Each ingested file has maximum size of 10 GBs