|  |
| --- |
| Amazon |
| AWS ML Specialty |
| AWS Certification 2022 |

|  |
| --- |
| Firdousi Abbas, Begum  7-30-2022 |

Table of Contents

[**DATA ENGINEERING** 2](#_Toc110264831)

[Quick Summary 2](#_Toc110264832)

[Data Store – S3, Data Lake 2](#_Toc110264833)

[AWS Data Stores for Machine Learning 3](#_Toc110264834)

[Streaming Data – Kinesis 3](#_Toc110264835)

[Serverless ETL – Glue 4](#_Toc110264836)

[ETL Pipeline Orchestrator for EC2 Jobs – AWS Data Pipelines 5](#_Toc110264837)

[Serverless Docker Batch Jobs – AWS Batch 5](#_Toc110264838)

[Database Migration to AWS – AWS Data Migration Service (DMS) 5](#_Toc110264839)

[Data Engineering Visual Workflow – AWS Step Functions 5](#_Toc110264840)

[**High Level AWS ML Services** 6](#_Toc110264841)

[Quick Summary 6](#_Toc110264842)

[**Exploratory Data Analytics** 7](#_Toc110264843)

[Python in Data Science and Machine Learning 7](#_Toc110264844)

[Types of Data in ML – Numerical, Categorical, Ordinal 7](#_Toc110264845)

[Data Distributions – Normal, Poisson, Binomial, Bernoulli 7](#_Toc110264846)

[Time Series Analysis - Series of data points over time period 7](#_Toc110264847)

[Serverless ad-hoc Interactive query service – Amazon Athena 8](#_Toc110264848)

[Serverless, Scalable BI Tool – Amazon QuickSight 8](#_Toc110264849)

[Managed Hadoop Cluster for ML in AWS - Elastic MapReduce (EMR) 8](#_Toc110264850)

[Apache Spark on EMR 9](#_Toc110264851)

[**Feature Engineering – Missing Data, Unbalanced Data, Label Data** 10](#_Toc110264852)

[**Amazon SageMaker – Model Tuning, Spark, Studio, Debugger, AutoPilot, Model Monitor, Clarify, Canvas, Training Compiler** 11](#_Toc110264853)

[**SageMaker ML Implementation and ML Ops** 13](#_Toc110264854)

[**Amazon SageMaker Algorithms** 15](#_Toc110264855)

[**General ML and DL** 15](#_Toc110264856)

[Formulas 15](#_Toc110264857)

[Overfitting and Dimensions 16](#_Toc110264858)

[Tuning Neural Networks 16](#_Toc110264859)

[Gradients (Deep Neural networks) 16](#_Toc110264860)

[Deep Learning Algos 17](#_Toc110264861)

[Extra Algorithms 17](#_Toc110264862)

[Activation Functions 17](#_Toc110264863)

# **DATA ENGINEERING**

## Quick Summary

Here's a quick summary of all the services we've mentioned

* **Amazon S3: Object Storage** for your data
* **VPC Endpoint Gateway: Privately access your S3 bucket** without going through the public internet
* **Kinesis Data Streams: real-time** data streams, need **capacity planning**, real-time applications
* **Kinesis Data Firehose: near real-time data ingestion** to **S3**, **Redshift**, **ElasticSearch**, **Splunk**
* **Kinesis Data Analytics: SQL transformations** on **streaming data**
* **Kinesis Video Streams: real-time video** feeds
* **Glue Data Catalog & Crawlers: Metadata repositories for schemas and datasets** in your account
* **Glue ETL: ETL Jobs as Spark programs**, run on a **serverless Spark Cluster**
* **DynamoDB: NoSQ**L store
* **Redshift: Data Warehousing for OLAP**, **SQL** language
* **Redshift Spectrum: Redshift on data in S3** (**without the need to load** it first in Redshift)
* **RDS / Aurora: Relational Data Store for OLTP**, **SQL** language
* **ElasticSearch: index for your data, search** capability, **clickstream** analytics
* **ElastiCache: data cache** technology
* **Data Pipelines: Orchestration of ETL jobs** between **RDS**, **DynamoDB**, **S3**. **Runs on EC2** instances
* **Batch: batch jobs run as Docker containers** - not just for data, manages EC2 instances for you
* **DMS: Database Migration Service, 1-to-1 CDC replication**, no ETL
* **Step Functions: Orchestration of workflows, audit**, **retry** mechanisms

## Data Store – S3, Data Lake

* **Amazon S3** – Store objects in buckets, globally unique name, **Max object size:** 5 TB, **Object Tags:** security, Version objects
* **Amazon S3 Durability**: 11 9’s across AZ for all storage class except S3-IA is within 1 AZ
* **S3 Availability**: Depends on storage class, check table
* **S3 for Machine Learning**: “Data Lake” for ML services like SageMaker, EC2, Athena, Redhisft Spectrum, Glue, etc
  + **File formats for ML**: CSV, JSON, Parquet, ORC, AVRO, Protobuf
  + **Data Partitioning**: Speed up range queries. Ex: By date, customerID, productName, etc
* **S3 Lifecycle:** Rules to automatically archive or delete/expire objects in S3
  + Transition between storage classes (**min 30 days** current or versioned)
  + **Expire Actions** to delete objects and versions (charged for min days of storage)
  + **Rules Based on:** “prefix” (ex: S3://mybucket/mp3/\*) or “Object Tags”
* **S3 Encryption:** Encrypt objects in S3
  + **SSE-S3:** Keys **handled and managed by AWS**
  + **SSE-KMS: Manage** keys **using** **KMS**. **Access to KMS** using **IAM**. **Audit** **usage** using CloudTrail
  + **SSE-C: Manage your own** encryption keys **outside AWS** (Ex. HSM module)
  + **Client-Side Encryption**
  + **Encryption for ML: SSE-S3 and SSE-KMS** most likely used
* **S3 Security:** User based, Resource based, VPC Endpoint Gateway for Private connection
  + **User-based:** IAM
  + **Resource-based:** Bucket policies (bucket wide), Object ACL (fine-grained), Bucket ACL (not used much)
  + **VPC Endpoint Gateway:** Access S3 from private services (SageMaker) in VPC
  + **Logging and Audit:** Access Logs – Another S3 bucket, API Calls – AWS CloudTrail
  + **Tag-based IAM and Bucket policies:** Tag objects for classification. Ex: PHI Data

## AWS Data Stores for Machine Learning

* **DynamoDB:**Serverless NoSQL store
  + Store Machine Learning Model served by application
* **Redshift:**Data Warehousing for OLAP (Online Analytical Processing), SQL language
  + **Data Stored in Columns**
  + **Frequently used Data:** Load data from S3 to **Redshift**
  + **Historical / Not-frequent Data:** Use **“Redshift Spectrum” to query data directly in S3** (no loading)
* **RDS / Aurora (Server-based):**Relational Data Store for OLTP (Online Transaction Processing), SQL language
  + **Data Stored in Rows**
  + Must provision servers in advance
* **OpenSearch / ElasticSearch (Server-based):** index for your data, search capability, clickstream analytics
* **Amazon FSx for Lustre file system:** faster startup and training times, which is natively integrated with Amazon S3.
* **ElastiCache:**data cache technology, mostly used for applications to for to reduce latency

## Streaming Data – Kinesis

* **Real-time or Near-real time Streaming**: from logs, metrics, IoT, clickstreams, etc
* **“Real-Time” big data processing**: Spark, NiFi, etc
* **Availability**: Data automatically replicated across 3 AZ
* **Kinesis Data Stream:** Real-Time Streaming, Broadcast to multiple consumers
  + **Shards / Partitions:** Streams divided into shards based on property and ordered data
  + **Scaling:** Provisioned or On-Demand (based on last 30 days throughput)
  + **Data Ingestion:** Real-Time (approx. 200ms latency for classic, 70ms for enhanced fan-out)
  + **Data retention:** Default **24 hours**, **up to 365 days, with replay capability**
  + **Record size:** Max 1 MB size
  + **Modes:**
    - **Provisioned:**  **Pay per shard**, Scaled manually**, Throughput In: 1 MB/s/shared or 1000 records, Throughput Out: 2 MB/s/shard or 5 API calls/sec/shard**: Shared or dedicated throughput for enhanced fan-out consumers
    - **On-Demand: Pay per stream/hour & Data in/out per GB**, **Scaled automaticall**y based on last 30 days, **Default capacity:** 4 MB/s/shard or 4000 records/sec
* **Kinesis Firehose:** Near Real-Time Streaming, Ingest, Transform & Store, Serverless, Auto – Scale
  + **Scaling**: Serverless, Automatic, fully managed
  + **Data Ingestion**: Near-real time with 60s min latency
  + **Data Retention**: No Data Retention, directly send to destination
  + **Record size:** Max 1 MB size
  + **Destination Storage:** Amazon S3, Redshift (copy via S3), ElastiSearch, Splunk
  + **Data Conversion:** JSON to Parquet/ORC (only for S3), **Transform via Lambda** – CSV to JSON
  + **Data Compression to Amazon S3:** GZIP, ZIP and SNAPPY
  + **Pricing**: Pay per 5KB increment of data passing through+ Format Conversion price in 5KB increments
* **Kinesis Data Analytics (KDA):** Real-Time Serverless Streaming ETL using SQL / Flink or ML
  + **Scaling:** Serverless, Automatic, fully managed
  + **Data Access**: IAM Permissions to source and Destination, Schema Discovery
  + **Data Computation:** SQL or Flink
  + **Data Pre-processing:** Lambda
  + **Pricing**: Pay for resources consumed (NOT CHEAP!!)
  + **Machine Learning on KDA:** RANDON\_CUT\_FOREST, HOTSPOTS
    - **RANDON\_CUT\_FOREST:** Anomaly Detection on Numeric columns, Use recent history
    - **HOTSPOTS:** Info on relative Dense regions in Data
* **Kinesis Video Streams:** Real-Time video Streaming with 1-stream/producer device
  + **Producers:** Security camera, AWS DeepLens, Smartphone camera, RTSP camera
  + **Consumers:** AWS SageMaker, Amazon Rekognition Video, BYO from EC2 with MXNet, Tensorflow
  + **Data retention:** 1 hour to 10 years with playback capability

## Serverless ETL – Glue

* **Glue Data Catalog:** Metadata/Schema repository for Data sources
  + **Automated Schema Inference**
  + Schema **Versioning**
  + **Schema and Data Discovery:** Integrated with Athena and Redshift Spectrum
* **Glue Crawlers:** Build Data Catalog by inferring schema and partitions
  + **Data Types:** JSON, Parquet, CSV, Relational store
  + **Data Sources:** S3, DynamoDB, Amazon Redshift, RDS, JDBC, MongoDB
  + **Schedule run:** On-demand or schedule
  + **Data Access:** IAM role/credentials
* **Glue ETL:** Transform , Clean, Enrich Data using Serverless ETL SPARK platform
  + **Scaling**: Serverless, Automatic, fully managed
  + **Code**: Auto-generate code in Python or Scala, modify
  + **Data Destinations**: S3, JDBC (RDS, Redshift), Glue Data Catalog
  + **Jobs**: Manual, **Glue Scheduler** - Scheduled, **Glue Trigger** - Triggered **based on events**
  + **Bundled Data Transformations**: DropFields, DropNulls, Filter (specify function), Join (Encrich), Map (add, delete fields, external lookup), redact sensitive data
  + **ML Data Transformations**: FindMatched ML – identify duplicate or matching records
  + **Data Format conversion**: CSV, JSON, Avro, Parquet, ORC, XML
  + **Apache Spark Transformation**: K-Means

## ETL Pipeline Orchestrator for EC2 Jobs – AWS Data Pipelines

* **Data Pipeline Orchestrator**: For ETL Jobs running on EC2 instances
* **Data Destination:** RDS, DynamoDB, Amazon S3, Redshift and EMR, On-prem Data source
* **Code:** Code on EC2 instances, not in the pipeline. More control on infrastructure
* **Workflow:** Retry and Notify on Failures
* Highly Available

## Serverless Docker Batch Jobs – AWS Batch

* **Fully serverless cluster: Run batch Jobs on Docker Images, Non-ETL work**
* **Workflow**: Dynamically provision EC2 & Spot instances based on volume and requirements
* **Price**: Pay for underlying EC2 instances
* **Schedule Batch Jobs**: CloudWatch Events
* **Orchestrate Batch Jobs**: AWS Step Functions

## Database Migration to AWS – AWS Data Migration Service (DMS)

* **Database Migration to AWS:**  Replicaion tasks with EC2, 1-to-1 CDC replication, no ETL
* **Source** database **remains** **available**
* Continuous Data Replication (CDC)
* **Data Sources:** Homogeneous (Oracle to Oracle) or Heterogeneous (SQL to Aurora)
* No Data Transformation. Use Glue once data is in AWS

## Data Engineering Visual Workflow – AWS Step Functions

* **Design workflows (Visual):** Orchestration of workflows, audit, retry mechanisms
* **Ability to Wait** (Upto 1 year)
* **Max Execution Time for State Machine:** 1 year
* **Ex:** Workflow to Tune, Train and Deploy Model

# **High Level AWS ML Services**

## Quick Summary

* **Amazon Comprehend** - Text Analytics, NLP, Extract: Entities, Key phrase, Language, Sentiment, Syntax. Custom Entity recognition for specification
* **Amazon Translate**- Translate Text, Deep Learning, Custom terminology in CSV or TMX
* **Amazon Transcribe**- Speech to Text, Custom Vocabulary, Identify Speaker, Auto-Lang detect, Identify Channel
* **Amazon Polly**- Text to Speech, **Pronunciation** with **Lexicons**, **Emphasis** with **SSML**, **Encode** **sentence**/**word** ends for lip syncing with **SpeechMarks**
* **Amazon Rekognition**- Computer Vision, Object & Scene Detection, Facial Recg, Text in Image, Video Analysis, Custom Labels for Unique items
* **Amazon Forecast**- Forecast Time Series, AutoML
* **Amazon Lex**- Chatbot, Intents for users, Utterances for words, Slots & Slot Value for input choices, Lambda for execution. Automated Chatbot Designer from Transcripts (preview 2022)
* **Amazon Personalize** - Recommendation Engine: recommend, personalized ranking from events.
  1. **Functions**: Real-Time or Batch, New users or new items, contextual recomm, similar items, unstructured text input, user segmentation
  2. **Terminology**: Datasets for input, Recipes pre-built, Solutions to train model with relevance, Campaigns to deploy solution version
  3. **Maintaining Relevance**: Updated Dataset, Real-Time user behaviour, retrain model every 2 hours incremented and full weekly
  4. **Security**: Restrict Data across accounts, Encrypt in Rest and Transit, Access control IAM, S3 bucket policy, Monitor, Log and Audit
  5. **Pricing**: Data Ingestion, Training, Inference (Transaction per sec), Batch recomm / user / item
* **Textract** – Identofy OCR, Form fields
* **DeepRacer** – Reinforcement Learning
* **DeepLens** - AI powered (DL enabled) camera for edge. **Use cases:** Facial Recognition, Object Detection
* **Amazon Lookout** - **Detect anomalies** in Industrial Equipment (Sensor Data, Computer Vision, Metrics)
* **Amazon Monitron** (**Sensor Device** to **monitor equipment**) - End to end system for monitoring industrial equipment and predictive maintenance
* **Amazon TorchServe** - Model serving PyTorch Framework
* **AWS Panorama** - Appliance for Computer vision at the edge for **existing** IP **cameras. Ingest Videos in near-real time**
* **AWS Neuron** - SDK for ML Inference on EC2 Inf1 instance
* **AWS DeepComposer** - AI-powered music piano keyboard
* **Amazon Fraud Detector** - Build Model from historical fraud data and expose API to use in app
* **Amazon CodeGuru** - Automated code reviews: Java, Python
* **ContactLens for Amazon Connect** - Search support calls and chat, Sentiment analysis, find utterances, categorize calls, talk speed, Theme detection
* **Amazon Kendra-** Enterprise search with natural language from files, sharePoint, intranet. Central search repository
* **Amazon Augmented AI (A2I)**- Workflows for Human review of low-confidence ML predictions. Integrated with SageMaker, Textract, Rekognition

# **Exploratory Data Analytics**

## Python in Data Science and Machine Learning

* **Pandas**: Slice and Dice Data
  + **Load data into DataFrame (DF)**: Explore and cleanup
  + **Pandas DF to Nympy array**: For sending to ML algorithms
* **Scikit\_learn:** Python Library for ML Models
  + **Scale** and **Pre-process data** **before** sent for **training**
  + Data format: Numpy array
* **Matplotlib**: Visualize Data in Pandas DF or Numpy array
* **Seaborn**: Visualize Data similar to Matplotlib but much prettier
* **Jupyter Notebook**: Web based Coding IDE for exploratory data analysis run by Python environments

## Types of Data in ML – Numerical, Categorical, Ordinal

* **Numerical**: Quantative, **Types**: **Discrete** (**Integer**), **Continuous** (**infinite**, fractional)
  + **Discrete:** Integer, Definite count. Ex. How many mangoes you have?
  + **Continuous:** Has infinite count possibility. Ex: How much rain fell?
* **Categorical**: Qualitative, No numerical meaning. Ex: Binary, Gender, Product
  + **Data Format:** Assign numbers to categories with no numerical meaning to represent in numbers for compact data
* **Ordinal**: Categorical data with numeric meaning. Ex: Ratings, Winning place
  + **Data Format:** Assign numbers to categories with numerical meaning

## Data Distributions – Normal, Poisson, Binomial, Bernoulli

* Likelihood of categorizing the data falling into a specific range
* **Normal Distribution:** Continuous Data, Probability of data point falling within given range of given value
  + Uses “Probability Density function (PDF)”
* **Poisson Distribution**: Discrete Data, Probability of Discrete data occurring in dataset
  + **Uses “**PMF Probability Mass Function (PMF)”
* **Binomial Distribution**: Discrete Data. Number of success in Binary questions
  + **Ex**: Flip a Coin - Heads or Tail, Yes or NO
* **Bernoulli Distribution**: Discrete Data. **Single Trial** Binomial Distribution

## Time Series Analysis - Series of data points over time period

* **Trends (T):** Pattern in Data in a specific time period
* **Seasonality (S):** Fluctuations in Data for a period
* **Noise (N):** Variations in Data
  + Constant variation = Additive (S+T+N)
  + Variation increase with trend = Multiplicative (S\*T\*N - T)

## Serverless ad-hoc Interactive query service – Amazon Athena

* **Scaling**: Fully managed, automatic, serverless
* **Query Type:** SQL
* **Query Engine**: Presto**, Read-Only** from S3
* **Data Type:** Structured, semi-structured and Unstructured data in S3
* **Data Formats:** CSV, JSON, ORC, Parquet, Avro
* **Integration Visualization**: QuickSight, ODBC/JDBC with other visualization tools
* **Integration ML:** Jupyter, Zepplin, RStudio notebooks
* **Security:** IAM, ACL & S3 Bucket policies, Encryption at rest and transit to S3
* **Use cases:** Query staging data before loading to Redshift, Logs

## Serverless, Scalable BI Tool – Amazon QuickSight

* Build visualizations and quickly get insights from data for “Business Users”
* **Scaling**: Fully managed, automatic, serverless
* **Data Sources:** Redhsift, RDS, Athena, EC2-Hosted DB, Files in S3 or on-prem
* **In-memory Engine:** SPICE - In-memory calculation engine for fast performance
* **QuickSight ML Insights:** ML with no prior-ML expertise
  + **Anomaly Detection:** Spot deviations in data Using Random Cut Forest
  + **Forecast:** Predict future metrics with seasonality (RCF)
  + **Auto-Narrative:** Automatic smart insights in Natural Language
* **Security:** 
  + **MFA for Account**
  + **VPC Connectivity:** Add QS IP address range to Database Security Groups
  + **Dataset Access**: Restrict to Row level and Column level access. **Tag-based** for anonymous embedding
  + **Private VPC Access:** Elastic Network Interface **ENI**, AWS Direct Connect

## Managed Hadoop Cluster for ML in AWS - Elastic MapReduce (EMR)

* Managed Hadoop Framework on EC2 Instances
* **Frameworks:** Spark, HBase, Presto, Flink, Hive and more
* **EMR Cluster:**
  + **Master/Leader Node:** Single EC2 instance to manage cluster. M4.larege < 50 nodes, m4.xlarge > 50 nodes
  + **Core Node:** Hosts HDFS data & run tasks, should be Highly Available, scale up and down. M4.large or m4.xlarge. Compute, memory
  + **Task Node:** run tasks only, does not host data. Good for spot instances (cost-effective)
  + **Add/remove Tasks node on the fly, provision new nodes on failure, resize nodes**
* **Notebooks**: EMR notebooks
  + Backed up to S3
  + In-built AWS integrations
  + Hosted inside VPC
  + Accessed via AWS console only
* **Pricing**: Charge by hour + EC2 charges
* **Security:** IAM roles and policies with Tag, Kerberos, SSH, EC2 Instance profiles, Service Linked roles
* **Using EMR:**
  + **Transient Jobs:** Use Spot Instances
  + **Long-Running Jobs:** Use Reserved Instances
  + **Ad-hoc exploration:** Connect to Master to run jobs
  + **Submit Ordered steps**: Via AWS console
* **EMR Storage:**
  + **HDFS**: Distributed Data on Instances for performance,
  + **EMRFS**: access S3 as HDFS, **EMRFS Consistent View:** S3 and DynamoDB to track consistency
  + **Local file system**
  + **EBS for HDFS**
* **Integrations:**
  + **EC2:** Instances for cluster nodes
  + **VPC:** Virtual network to launch instances
  + **S3:** Input & Output data
  + **CloudWatch:** Monitor cluster and configure alarms
  + **IAM:** Permissions
  + **CloudTrail:** Audit requests made to services
  + **Data Pipeline:** Orchestrate, schedule and start clusters

## Apache Spark on EMR

* **MapReduce**: Software framework for parallel large data processing
  + **Map**: Prepare and Transform Data
  + **Reduce**: Aggregate and Distil to final results
* **Apache Spark:** faster alternative to MapReduce, open source distributed processing system for big data workloads
  + **Optionally installed on EMR**
  + **Query Execution:** Spark SQL - in-memory cache for optimized fast analytical queries against any size of data
  + **Data Sources:** JDBC, ODBC, JSON, HDFS, Hive, ORC or parquet
  + **Data Storage:** Uses **Resilient Distributed Dataset (RDD)** - logical collection of partitioned data across compute nodes
  + **Data Format:** "**Dataframe in Python**", "**Dataset in Scala**
  + **API:** Java, Scala, python and R
  + **Use cases:** Batch, interactive queries in real time analytics, machine learning, graph processing, stream processing
* **Spark Streaming:** Streaming analytics on Data ingested in mini batches
  + **Data Sources:** Twitter, Kafka, Flume, HFS and zero MQ, Amazon Kinesis
* **GraphX:** Distributed Graph processing framework built on top of Spark
* **MLLib:** Machine Learning Library for Spark running distributed across cluster
  + Classification, regression, Decision Trees, Recommender (ALS), Clustering K-Means, LDA, SVD, PCA, ML workflow utilities, statistics
* **Zeppelin or EMR Notebooks + Spark:** Run Spark code interactively
* **EMR Notebook:** Backed-up in S3 with AWS integration hosted inside VPC

# **Feature Engineering – Missing Data, Unbalanced Data, Label Data**

* Create better features to train model using data knowledge and model in use
* **The Curse of Dimensionality** - Select the features relevant to problem at hand
  + Domain knowledge
  + **Unsupervised Feature Reduction technique:** PCA, K-Means
* **Imputing Missing Data:** MEAN, Drop, Use KNN, Get more Data
  + **Replace with MEAN/Median:** MEAN value from rest of the column, NOT rows, MEDIAN: when outliers present. Not best choice!!
  + **Drop feature:** If not many rows contain missing data, not introduce bias. Not best choice!!
  + **Use ML:** 
    - **KNN for Numerical:** Find K “nearest” (most similar rows) and average
    - **Deep Learning:** Well for categorical data, but complicated
    - **Regression:** Linear or non-linear relationship between features.
    - **MICE (Multiple Imputation by Chained Equations) :** Most advanced!!
  + **Just Get More Data**
* **Unbalanced Data:** Large discrepancy between positive and negative cases. Mainly neural network problem
  + **Oversampling:** Duplicate sample values from minority class randomly
  + **Under sampling**: Remove negative ones instead of creating positive one, Not best choice!!
  + **SMOTE**: Synthetic Minority Over-Sampling Technique. Artificially generate new samples of the minority class from K-nearest neighbours and get the mean of neighbors
  + **Adjusting Thresholds for Classification Problem**: Increase/lower the threshold for false positive or false negative
* **Handling Outliers:** Find and handle deviations/variations from training data
  + **Find deviations/outliers:** Standard Deviation = Square root of variance
  + **Remove outliers responsibly!!:** Do not introduce bias
  + **Random cut forest algorithm to identify outlier**
  + **Plot:** Box & whisker, or just "box plots", organize your data into quartiles, and display outliers in the outer quartiles.
* **Binning:** Categorizing based on range of values. Ex: Age < 30, Age <20,
  + **Quantile binning**: Even sizes in each bin
* **Transforming feature**: Apply function to a feature to make it better suit for training
  + **Logarithmic transform:** Exponential trend like YouTube recommendation
* **Encoding**: Transform data into some new representation required by model
  + **One-hot coding (EXAM!!):** Technique to encode categorical data into binary 1 a
  + **Ex:** “color” variable with “green”, “Blue”, “yellow”, there are 3 categories and therefore 3 binary variables: G:1,0,0 B:0,1,0 Y:0,0,1
* **Shuffling:** Shuffle training data to avoid learning from residual data
* **Scaling / Normalization:** Scale feature values to comparable values to get similar weight
  + Models prefer Feature data Normally distributed around ZERO
  + **Scikit\_learn:** Pre-processor “MinMaxScaler”
* **Label Generation**
  + **Amazon SageMaker Ground Truth**: Manage humans who will label data for Training
    - Creates own model as images are labelled, learn only send more if not sure (Cost-effective 70%)
  + **Amazon Ground Truth Plus:** Outsource labelling to AWS Experts
  + **Rekognition (Classify Image)**
  + **Comprehend (classify text by topics, sentiments)**
  + **Pre-trained or unsupervised technique**

# **Amazon SageMaker – Model Tuning, Spark, Studio, Debugger, AutoPilot, Model Monitor, Clarify, Canvas, Training Compiler**

* **Hyperparameter Tuning:** Auto-train and deploy hyperparameters with best results
  + **Define:** Hyperparameters and ranges, Metrics to optimize
  + **Training Job:** Auto spun training instances to train as many combinations allowed
  + **“Learns as it goes”**
  + **Best Practices:**
    - Don't optimize too many hyperparameters at once
    - Limit ranges to as small as possible
    - Use logarithmic scales when appropriate
    - DON't run too many training jobs concurrently
    - Limits how well the process can learn
    - Make sure Training jobs runs on multiple instances report correct objective metric in the end
* **Apache Spark with SM:** Pre-process data with “sagemaker-spark” library
  + **SageMakerEstimator:** K-means, PCA, XGBoost
  + **SageMakerModel:** From estimator to make inferences
* **SageMaker Studio:** Visual Integrated IDE for ML on AWS
  + **Notebooks:** Create and share Jupyter notebook, Switch HW configs to execute
  + **Experiments:** Organize, Capture, Compare and search ML Jobs & runs
  + **Debugger:**
    - **Save Model state:** At periodic intervals
    - **Define rules:** Run job for each and fire CloudWatch event when rule is hit
    - **Debugger Insights Dashboard**
    - **Auto-generate training reports**
    - **Built-in rules:** Monitor bottlenecks, profile model framework, debug params
    - **Frameworks:** Tensorflow, MXNet, PyTorch, XGBoost, Generic estimator
    - **DebuggerProfileRule:** HW system metrics, framework metrics
    - **Built-in notifications (via SNS), Stop Training**
    - **Profile resources usage and training**
* **SageMaker Autopilot:** Automatic algorithm selection, data processing, model tuning, manage infrastructure, Trial and Error
  + **Workflow:** Load data from S3 🡪 Target column to predict 🡪 Automatic model creation 🡪 **Generate artifacts:** Notebook, ranked list of models to pick 🡪 Deploy & Monitor 🡪 Refine via Notebook
  + **Problem Types:** Binary classification, Multiclass classification, Regression
  + **Algorithms:** Linear learner, XGBoost, DeepLearning (MLPs)
  + **Data:** Tabular CSV
  + **Add Human Guidance**
  + **No code or Code with AWS SDK**
* **SM Autopilot + Clarify:** Explain Autopilot Model Prediction
  + Transparency on model predictions
  + Feature attributions, Bias
* **SageMaker Model Monitor**: Alerts on Model quality deviations (via CloudWatch)
  + **Monitoring Types:**
    - Drift in Data Quality
    - Drift in Model Quality (accuracy)
    - Detect anomalies, Detect new features
    - Bias Drift
    - Feature Attribution Drift: Based on NDCG, ranking in training vs live data
  + **Data**: Stored in S3 and secured, No Coding
  + **Jobs**: Scheduled via Monitoring Schedule
  + **Metrics**: Sent to CloudWatch to trigger alarms or take corrective actions
  + **Visualize**: Tensorboard, QuickSight, Tableau, SageMaker Studio
* **SageMaker Model Monitor + Clarify**: Detect potential bias and alert via CloudWatch, Helps explain model behavior
* **SageMaker Canvas:** No code machine learning for Business Analyst. Within SageMaker Domain
  + **Workflow:** Upload CSV data (only) 🡪 Select column to predict 🡪 Build 🡪 Predict
  + **Problem Types:** Classification and Regression, Time series forecasting enabled via IAM
  + **Auto-Prepare Data:** Missing values, Outliers, Duplicates
  + **Share models & datasets:** SageMaker Studio
  + **Local Files Upload to S3:** Configured with CORS, by IT administrator
  + **Import from Redshift:** Can be setup
  + **Integrate with Okta SSO, run within a VPC**
  + **Price:** 1.9$/hr + charge based on number of training cells
* **SageMaker Training Compiler** - Automatically optimizes training job on GPU instances, integrated into Deep Learning Containers
  + **Acclerare upto 50%**
  + Incompatible with SageMaker Distributed training Libraries
* **SageMaker Batch Transform:** To get inferences for an entire dataset, use batch transform.
  + exclude attributes before running predictions
  + join the prediction results with partial or entire input data attributes when using data that is in CSV, text, or JSON format.
* **JumpStart:** One-click open-source models
* **Data Wrangler:** Import, Transform, analyse and export data with SM Studio
* **SM Feature Store:** Find, discover and share features in studio
* **SM Edge Manager:** SW agent for Edge device (SM Neo) to monitor, label and retrain

# **SageMaker ML Implementation and ML Ops**

* **SageMaker ML Architecture:** Training, Deployment and Inference using Docker images
* **Host SM Models in Docker Containers:** Isolated with all dependencies and resources needed to run, training and inference code
  + **Pre-built Deep Learning, scikit-learn, Spark ML, Tensorflow, MXNet, PyTorch**
  + **Distributed Training:** via Horovod or Paramater Servers
  + **Structure of Container files:**
    - **Training Container Code Files:** /opt/ml/code
    - **Deployment Container Code (Inference code):** /opt/ml/model
  + **Docker Image Structure:** From Elastic Container Registry (ECR),WORKDIR/
    - **nginx.conf:** Config file for nginx frontend web server
    - **predictor.py:** Flask web server for making predictions at runtime
    - **serve/:** program started when container is hosted
    - **train/:** program invoked for training
    - **wsgi-py:** wrapper to invoke flask application to serve results
  + **Training entrypoint (EXAM!!):** ENV **SAGEMAKER\_PROGRAM**
  + **NVIDIA Container Toolkit**: build and run GPU accelerated Docker containers
  + Using your own image
* **Production Variants (EXAM!!):** Test out multiple models on live traffic with variant Weights (A/B Testing)
* **SageMaker Lifecycle configuration:** automate customizations to be applied at different phases of the lifecycle of an instance. **Ex:** configure the scripts to automatically execute every time your notebook instance is started
* **SageMaker on the Edge:** Deploy and Run SageMaker model on the edge devices
  + **SM Neo**
    - Train once, run anywhere (Edge Devices)
    - **Optimize for specific devices:** Tensorflow, MXNet, PyTorch, XGBoost, ONNX
    - Compiler + runtime, latency sensitive use case
  + **SM Neo + IoT Greengrass**
    - **Deploy Neo Models:** HTTPS endpoint hosted on EC2 Instances
    - **To IoT Greengrass:** **deploy** models to an **actual edge device**, **Inference** at edge **using model trained in the cloud**, **uses Lambda inference apps**
* **SageMaker Security:**
  + **General AWS Security:** IAM, MFA, SSL/TLS, CloudTrail, Encryption, PII
  + **Protect Data at Rest using KMS:** **Encrypt** in Notebook and all SM Jobs**, S3**: Encrypt buckets for training and hosting models
  + **Protect Data in Transit**: TLS/SSL, IAM Roles for SM to access resources, optionally encrypt inter-node training communication
  + **SageMaker + Private VPC:** setup S3 VPC endpoints, S3 bucket policies
  + **Disable Notebook internet access:** Interface endpoint (private link) or NAT to allow outbound connections for training and hosting
  + **Disable Container internet access**
  + **SageMaker + IAM:** User Permissions, Predefined Policies
  + **SageMaker Logging and Monitoring:** CloudWatch and CloudTrail
    - **CloudWatch:** log, monitor and alarm on **endpoints latency, instance health, GroudTruth**
    - **Amazon CloudWatch API**: send events that **occur during model training** to CloudWatch, and create a dashboard of those metric or send notification via SNS
    - **CloudTrail:** Audit actions from users, roles and services. Log files in S3
* **SageMaker Resource Manager:**
  + **Instance Types:** Training DL**:** GPU (P2 or P3), Inference: CPU (C4, C5), GPU: pricey
  + **Managed Spot Training:** EC2 Spot instances: 90% cost saving for training. Use checkpoints to S3, Increases training time
  + **Automatic Scaling:**
    - **Scaling Policy:** Target metrics, min/max capacity, cooldown period
    - **Monitor** with **CloudWatch**
    - **Dynamically** **adjust** **num** **of** **instances** for prod variants
    - **Load test config** before using
  + **Availability Zones:** Automatically distribute instance across AZ, **Need multiple instances for each prod endpoint**, VPC: 2 subnets in different AZ
* **SageMaker Inference:**
  + **Elastic Inference:** Accelerate DL Inference, fraction of cost using GPU.
    - Added alongside CPU, Apply to notebooks
    - **Custom & pre-built containers:** Tensorflow MXNet,
    - **Problem Types:** Image classification and Object Detect
  + **Serverless Inference**: Auto-provisioned and scaled, Pay per use
    - **Workflow:** Specify container, memory and concurrency
    - **Monitor via CloudWatch:** ModelSetupTime, Invocations, MemoryUtilization
  + **Inference Recommender:** Best instance type and config for models
    - **Automate load testing for instance recommendation (45 min)**
    - **Custom Load Test – 2 hours**
    - Model tuning
    - **Deploy optimal inference endpoint**
    - **Workflow:** Registry Model in registry 🡪 Benchmark endpoint configs 🡪 Collect & Visualize metrics to decide instance types, Existing open source models from zoos
  + **SM Inference Pipelines:** Linear sequence of 2-15 containers
    - Pre-process, predict, post-process.
    - SparkML, scikit-learn containers
    - Real-time inference and Batch transforms
  + **Customer Inference Container:** 
    - Your inference container responds to port 8080
    - respond to ping requests in under 2 seconds
    - Model artifacts need to be compressed in tar forma

# **Amazon SageMaker Algorithms**

Check “SageMaker ML Algorithms” PDF

Dsfffsf

Sdfdffd

# **General ML and DL**

## Formulas

* **Recall** (reduce FN) = TP/TP + FN
* **Precision** (reduce FP) = TP/TP+ FP
* **Specificity** (True Negative) = TN/TN + FP
* **False Negative Rate** = FN / FN+TP
* **F1 Score** (Both Precision and Recall) = 2xTP / 2xTP + FP + FN, 2 (Precision x Recall) / Precision + Recall
* **RMSE:** Accuracy measurement **(care about wrong and right**) = Add Squared error and take square root
* **Residual Plot (Visualization):**  Shows difference between observed vs predicted responses
* **ROC Curve: Plot TP rate vs FP rate = points above diagonal represent good classification**, Best = Point in upper left corner
* **AUC (Area Under Curve)** - Metric for comparing classifier, ROC AUC 1.0 is perfect, 0.5 useless
* **Variance measures how "spread-out" the data is**: Average of squared differences from mean **(Mean, Difference from mean, Square, Average)**
* **Standard Deviation (Outliers)** = Square root of variance
* **Correlation coefficient (Pearson’s correlation):** Relationship between variables. **Negative**: 1 increase, other decrease. **Positive**: 1 increase, other increase
* **TF-IDF - Term Frequency and Inverse Document Frequency:** Search Documents, how relevant given term is for a given document
  + **TF-IDF Score =** Term Frequency / Document Frequency
  + The Tf-Idf is a great way of **giving weights to words as it penalizes generic words that commonly appear across all sentences**.
    - Even if a word has a low frequency across all documents, the Tf-Idf vectorizer is still able to capture how special that word is by giving it a higher score.
  + **TF-IDF Matrix (Rows, Cols) =** Number of Docs, Number Unique Unigrams & Bigrams
    - **Unigrams:** Unique words in all docs
    - **Bigrams:** Unique Pair of words in all docs

## Overfitting and Dimensions

* **Bagging**: Avoid **overfitting** and easier to **parallelize**
* **Boosting**: Better **Accuracy**
* **Regularization Technique (DL) - Prevent Overfitting (in DL ?) on test data** with high accuracy on training data
  + Too many layers and too many neurons - **Use simpler model**
  + **Dropout neurons**
  + **Early Stopping (Epochs) -** when validation score starts to get bad
* **L1 & L2 Regularization (non DL):** Prevent overfitting in general ML, added as weights.
  + L1 = **sum of weights : Feature selection=0, Sparse output, inefficient, avoid curse of dimensionality**
  + L2 = **Sum of Square of weights: Weighted features, Dense output, efficient**
* **Low-Dimensional Data**: Produces low accuracy scores. Add more features

## Tuning Neural Networks

* **Learning Rate: how far are sample solutions (weights)** = when sampling them in order to **minimize cost function, over many epochs** 
  + **Too high = overshoot optimal solution**
  + **Too small = increase training time**
* **Batch Size:** How many **training samples within each batch** of each epoch
  + **Smaller batch** = work **easily out of "local minima"**
  + **Larger batch** = converge to **wrong solution, stuck on "local minima"**

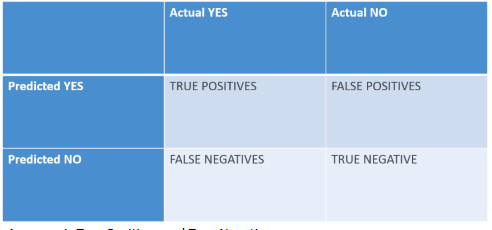
## Gradients (Deep Neural networks)

* **Vanishing Gradient Problem:** Slope of the **Learning curve approaches ZERO in training**, starts to introduce numerical errors
* **Fixing the Vanishing Gradient Problem:** 
  + **Multi-level hierarchy**: Break neural network into sub-network,
  + **LSTM** (Long Short-term memory),
  + **ResNet** (Residual Network),
  + **Activation function** (ReLu)
* **Gradient Checking: Debugging Technique**, Validate code of neural network training
* **Stochastic Gradient Descent (SGD):** makes **updates to the weights of the linear model for every data example** it sees
  + size of these updates is controlled by the learning rate
  + **Too large a learning rate =** **prevent** the weights from **approaching the optimal solution**
  + **Too small =** **increase training time**
* **A Loss function = how effective an algorithm is at modeling the data**.
  + if a model consistently predicts values that are very different from the actual values, it returns a large loss

## Deep Learning Algos

* **CNN:** used for **computer vision applications, or multi-dimensional data** more generally.
* **RNN (Recurrent Neural Network), base model for text:** Sequential computation. **Ex: words in sentence**
  + RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.
  + As it goes deeper in layer, it tends to give less weight to previous computation leading to “Vanishing Gradient” problem solved by LSTM
* **LSTM (Long Short-Term Memory) :** Type of RNN that **solves the problem of items losing their weight over time.** words in a sentence may be significant regardless of their position.
* **Natural Language Processing:**
  + **Part-of-Speech (POS) Tagging: Not pre-processin.** To include only noun and adjective words. categorizing words in a text (corpus).
  + **Remove English Stop words**: Doesn’t add any value. Ex: And, OR
  + **Stemming**: Employ stemming to reduce words to their basic form or stem, which may or may not be a legitimate word in the language.
    - **Ex**: For instance, the stem of these three words, connections, connected, connects, is “connect”. On the other hand, the root of trouble, troubled, and troubles is “troubl,” which is not a recognized word.
  + **Lemmatizing:** Reduce words to Dictionary word that makes sense
  + **Normalize to lowercase character**
* **Support Vector Machines (SVM) with Radial Bias Function Kernel:** Taking features from low dimension to high dimension, 2D to 3D and then use the plane to classify
* **Random Forest Algorithm:** Increase prediction accuracy and prevent overfitting that occurs within single decision tree

## Extra Algorithms

* **Multinomial logistic regression (Not DL):** is used to predict categorical placement in or the probability of category membership on a dependent variable based on multiple independent variables.
* **Binary logistic regression**
* **Confusion Matrix**
* 

## Activation Functions

* **Softmax**: converts a vector of real numbers into a vector of a probability distribution whose sum is equal to 1
* **Rectified Linear Unit (ReLu):** convert inputs into the range of (0, positive input value) and not into a probability distribution
* **Sigmoid Function (Binary classification):** squashes inputs as 0 (as more negative a number gets) or 1 (as more positive a number gets)
* **Hyperbolic Tangent (tanh):** similar to the Sigmoid Function — but it's centered around 0 instead of Sigmoid's 0.5.
  + more preferable because it allows the model to converge faster to a minimum

# **Data Visualization**

**Graphing (scatter plot, time series, histogram, box plot)**

* **Scatter plot:** Scatter plot is a graph of two sets of data along the two axes to visualize the relationship between the two variables
  + **positive (or negative) linear relationship:** value along the Y axis seem to increase as X axis increases (or decreases)
  + **lack of dependent relationship**: points are randomly distributed with no obvious pattern – ZERO CORRELATION
* **Time Series:** visualize trends in counts or numerical values over time(X-axis).
* **Marginal Histogram**
* **Box Plot:**  Divided into Quartiles Minimum – Maximum. Outside quartiles:Outliers

 Interpreting descriptive statistics (correlation, summary statistics, p value)

 Clustering (hierarchical, diagnosing, elbow plot, cluster size)