DATA –ML-LIFE-CYCLE

COMPREHENSIVE GUIDE

STEP 1: PROBLEM DEFINITION & BUSINESS UNDERSTANDING

- Define the business objective (e.g., predict customer churn, detect log anomalies)
- Translate business goal into ML task:
  - Classification (binary/multi-class)
  - Regression
  - Clustering
  - Generative AI (e.g., text generation)
- Identify success metrics (accuracy, F1-score, AUC, latency, cost savings)

STEP 2: DATA COLLECTION

- Identify data sources:
  - Databases (SQL, NoSQL)
  - APIs (REST, GraphQL)
  - Log files (system, application)
  - CSV/Excel files
  - Cloud storage (S3, Azure Blob, GCS)
- Ensure data ownership, privacy, and compliance (GDPR, HIPAA)
- Version data using tools like DVC (Data Version Control) or AWS Lake Formation

STEP 3: DATA PREPROCESSING & CLEANING

- Handle missing values:
  - Drop (df.dropna())
  - Impute (mean, median, ML-based)
- Remove duplicates and outliers (IQR, Z-score)
- Standardize/normalize numerical features (StandardScaler, MinMaxScaler)
- Encode categorical variables:
  - Label Encoding
  - One-Hot Encoding (pd.get_dummies(), OneHotEncoder)
- Text cleaning (for NLP):
  - Lowercasing, removing punctuation, stop words
  - Tokenization, lemmatization

STEP 4: EXPLORATORY DATA ANALYSIS (EDA)

- Compute summary stats (df.describe())
- Visualize distributions (histograms, boxplots)
- Analyze correlations (heatmap, pairplot)
- Identify class imbalance (use SMOTE or class weights if needed)
- Tools: Pandas, Matplotlib, Seaborn, Plotly, Sweetviz

STEP 5: FEATURE ENGINEERING

- Create new features:
  - Binning (age → age_group)
  - Interaction terms (price × quantity)
  - Date/time features (day_of_week, hour)
- Dimensionality reduction (PCA, t-SNE) for high-dimensional data
- Text vectorization (TF-IDF, Word2Vec, embeddings)
- Use Featuretools for automated feature engineering

STEP 6: TRAIN/VALIDATION/TEST SPLIT

- Split data:
  - Train (70–80%): Model training
  - Validation (10–15%): Hyperparameter tuning
  - Test (10–15%): Final evaluation
- Use train_test_split (scikit-learn) or TimeSeriesSplit for time-series
- Ensure stratified sampling for imbalanced classes

STEP 7: MODEL SELECTION & TRAINING

- Choose algorithm based on task:
  - Classification: Logistic Regression, Random Forest, XGBoost, Neural Networks
  - Regression: Linear Regression, SVR, Neural Networks
  - Deep Learning: CNN (images), RNN/LSTM (sequences), Transformers (text)

STEP 8: MODEL EVALUATION

- Compute metrics:
  - Classification: Accuracy, Precision, Recall, F1, AUC-ROC
  - Regression: MSE, RMSE, MAE, R^2
- Use cross-validation (cross_val_score)
- Analyze confusion matrix, ROC curve
- Tools: Scikit-learn, Yellowbrick

STEP 9: HYPERPARAMETER TUNING

- Use GridSearchCV or RandomizedSearchCV (scikit-learn)
- Advanced: Optuna, Hyperopt, Ray Tune
- Track experiments with MLflow, Weights & Biases, or TensorBoard

STEP 10: MODEL SERIALIZATION (SAVE MODEL)

- import pickle

- # Save
- with open('model.pkl', 'wb') as f:
-     pickle.dump(model, f)

- # Load
- with open('model.pkl', 'rb') as f:
-     loaded_model = pickle.load(f)

STEP 10: MODEL SERIALIZATION (SAVE MODEL)

- # For TensorFlow/Keras models
- model.save('my_model') # SavedModel format (folder)

- # For Hugging Face Transformers (uses SafeTensors by default)
- from transformers import AutoModel
- model = AutoModel.from_pretrained("bert-base-uncased")
- model.save_pretrained("my_bert_model") # Includes safetensors

STEP 11: MODEL TESTING & VALIDATION

- Test on unseen data (X_test, y_test)
- Validate edge cases and failure modes
- Perform bias/fairness checks (using SHAP, LIME, AI Fairness 360)
- Document model limitations and assumptions

STEP 12: MODEL DEPLOYMENT TO CLOUD USE FLASK/FASTAPI + DOCKER:

- # app.py
- from flask import Flask, request, jsonify
- import pickle

- app = Flask(__name__)
- with open('model.pkl', 'rb') as f:
-     model = pickle.load(f)

- @app.route('/predict', methods=['POST'])
- def predict():
-     data = request.json
-     pred = model.predict([data['features']])
-     return jsonify({'prediction': int(pred[0])})

STEP 12: MODEL DEPLOYMENT TO CLOUD AWS - SAGEMAKER

- from sagemaker.tensorflow import TensorFlowModel
- model = TensorFlowModel(model_data='s3://bucket/model.tar.gz', role=role)
- predictor = model.deploy(initial_instance_count=1, instance_type='ml.m5.large')

STEP 13: MONITORING & MAINTENANCE

- Track:
  - Prediction latency
  - Model drift (data/concept drift)
  - Accuracy decay over time
- Tools: SageMaker Model Monitor, Evidently AI, Prometheus + Grafana
- Set up alerts (CloudWatch, PagerDuty)
- Retrain model periodically with fresh data

STEP 14: GOVERNANCE & SECURITY

- Encrypt data at rest/in transit
- Use IAM roles (not access keys)
- Audit model predictions (for compliance)
- Store models in secure, versioned repositories (S3, Model Registry)

STEP 15: CI/CD FOR ML (MLOPS)

- Automate pipeline using:
  - AWS SageMaker Pipelines
  - Azure ML Pipelines
  - Kubeflow, MLflow, GitHub Actions
- Steps: Data → Train → Test → Deploy → Monitor

END-TO-END MACHINE LEARNING PIPELINE WITH TENSORFLOW

1. DATA COLLECTION & INGESTION

```
# Libraries
import pandas as pd                           # Data loading (CSV, Excel, DB)
import sqlalchemy                             # SQL database connection
import boto3                                  # AWS S3 access
import requests                               # API data fetching
From google.cloud import storage              # GCP Cloud Storage
```

Ingest from databases, APIs, cloud storage (S3, GCS), or files
Use pandas.read_csv(), pd.read_sql(), boto3.client('s3')

2. DATA PREPROCESSING & CLEANING

```
# Libraries
from sklearn.preprocessing  import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.impute import SimpleImputer
import re, string
```

- Handle missing values (SimpleImputer)
- Encode categoricals (OneHotEncoder, LabelEncoder)
- Scale features (StandardScaler)
- Clean text: lowercase, remove punctuation, regex

3. EXPLORATORY DATA ANALYSIS (EDA)

```
# Libraries
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

- Summary stats: df.describe()
- Visualize distributions, correlations, class balance
- Detect outliers and anomalies

4. FEATURE ENGINEERING

```
# Libraries
from sklearn.feature_extraction.text import TfidfVectorizer    # For NLP
from sklearn.decomposition import PCA                          # Dimensionality reduction
import numpy as np
```

- Create interaction/binned features
- Text vectorization (TfidfVectorizer)
- Reduce dimensions (PCA)

5. TRAIN/VALIDATION/TEST SPLIT

```
# Library
from sklearn.model_selection    import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)
```

6. MODEL BUILDING WITH TENSORFLOW

```
# Libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Conv2D, Embedding, GlobalAveragePooling1D
from tensorflow.keras.optimizers import Adam
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')
])
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

7. MODEL TRAINING & EVALUATION

```
# Train
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    epochs=10, batch_size=32)


# Evaluate
from sklearn.metrics import classification_report, confusion_matrix
y_pred = np.argmax(model.predict(X_test), axis=1)
print(classification_report(y_test, y_pred))
```

8. HYPERPARAMETER TUNING (OPTIONAL)

```
# Libraries

import keras_tuner as k                    # For Keras models
# or
import optuna                              # General-purpose tuner
```

MODEL SERIALIZATION – TWO OPTIONS OPTION A: SAVE AS .PICKLE (ONLY FOR SCIKIT-LEARN OR NON-TF MODELS)

```
import pickle
# ONLY IF using sklearn model
with open('model.pkl', 'wb') as f:
    pickle.dump(sklearn_model, f)

# Save full TensorFlow model (includes weights, architecture, optimizer)
model.save('my_tf_model')  # Creates SavedModel directory

# For Hugging Face-style SafeTensors (requires transformers)
from transformers import TFAutoModel
# (Used when working with BERT, etc.)
model.save_pretrained('my_model_dir')  # Automatically uses safetensors if available
```

10. MODEL VALIDATION & TESTING

- Test on hold-out dataset
- Validate edge cases
- Use SHAP or LIME for explainability:

```
import shap
explainer = shap.DeepExplainer(model, X_train[:100])
```

11. CLOUD DEPLOYMENT OPTIONS (A. AWS SAGEMAKER (RECOMMENDED)

```
# Libraries
import sagemaker
from sagemaker.tensorflow import TensorFlowModel

# Deploy SavedModel to SageMaker
model = TensorFlowModel(
    model_data='s3://my-bucket/model.tar.gz',  # Upload saved_model.tar.gz
    role='SageMakerRole',
    framework_version='2.12'
)
predictor = model.deploy(instance_type='ml.m5.large', initial_instance_count=1)
```

11. CLOUD DEPLOYMENT OPTIONS
B. FLASK API + DOCKER (SELF-MANAGED)

```
# app.py
import tensorflow as tf
from flask import Flask, request, jsonify

app = Flask(__name__)
model = tf.keras.models.load_model('my_tf_model')

@app.route('/predict', methods=['POST'])
def predict():
    data = request.json['features']
    pred = model.predict([data])
    return jsonify({'prediction': pred.tolist()})
```

12. MONITORING & MLOPS

```
# Libraries
import boto3                              # AWS CloudWatch logs
from sagemaker.model_monitor import ModelMonitor  # SageMaker monitoring
import evidently                          # Data drift detection
```

DEPLOYING MACHINE LEARNING MODELS TO AWS SAGEMAKER

STEP 1: PREPARE TRAINED MODEL

- Ensure your model is trained using TensorFlow, PyTorch, scikit-learn, or XGBoost
- Save model in SageMaker-compatible format:
  - TensorFlow: model.save('my_model') → creates saved_model.pb + variables folder
  - PyTorch: Save as .pth with torch.save(model.state_dict(), 'model.pth')
  - Scikit-learn: Save as .joblib (preferred over .pickle for security):

```
import joblib
joblib.dump(model, 'model.joblib')
```

Do NOT use .pickle in production — use .joblib or framework-native formats (SavedModel, safetensors)

STEP 2: CREATE MODEL TARBALL(COMPRESS MODEL INTO .TAR.GZ)

```
tar -czf model.tar.gz model/  # For TensorFlow SavedModel
tar -czf model.tar.gz model.joblib  # For scikit-learn
```

STEP 3: UPLOAD MODEL TO AMAZON S3

```
import boto3
s3 = boto3.client('s3')
s3.upload_file('model.tar.gz', 'your-sagemaker-bucket', 'models/model.tar.gz')
```

STEP 4: CREATE AN IAM ROLE FOR SAGEMAKER

- Go to IAM Console → Roles → Create Role
- Trusted Entity: SageMaker
- Attach policies:
  - AmazonSageMakerFullAccess
  - AmazonS3FullAccess (or scoped to your bucket)
- Note the ARN: arn:aws:iam::123456789012:role/SageMakerExecutionRole

STEP 5: CREATE SAGEMAKER MODEL

```
import sagemaker
from sagemaker import get_execution_role

role = get_execution_role()  # Or use your ARN

from sagemaker.tensorflow.model     import TensorFlowModel

model = TensorFlowModel(
    model_data='s3://your-sagemaker-bucket/models/model.tar.gz',
    role=role,
    framework_version='2.12',                    # Match your TF version
    py_version='py310'
)
```

For PyTorch: Use PyTorchModel
For Scikit-learn: Use SKLearnModel

STEP 6: DEPLOY TO REAL-TIME ENDPOINT

```
predictor = model.deploy(
    initial_instance_count=1,
    instance_type='ml.m5.large',  # Choose based on latency/cost
    endpoint_name='my-ai-model-endpoint'  # Must be unique
)
```

- Instance Types:
- Low cost: ml.t3.medium
- Balanced: ml.m5.large
- High perf: ml.c5.xlarge, ml.g4dn.xlarge (for deep learning)

STEP 7: INVOKE THE ENDPOINT

```
# For TensorFlow (JSON input)
response = predictor.predict({
    'instances': [[1.0, 2.0, 3.0, ...]]  # Match input shape
})

# For scikit-learn (CSV input)
predictor.serializer = sagemaker.serializers.CSVSerializer()
response = predictor.predict([1.0, 2.0, 3.0])
```

STEP 8: MONITOR & MANAGE

- CloudWatch Metrics: Latency, Invocations, Errors
- Enable Auto-Scaling:

```
predictor.update_endpoint(
    initial_instance_count=2,
    instance_type='ml.m5.xlarge'
)
```

Delete when not needed:

```
predictor.delete_endpoint()
```

STEP 9: (OPTIONAL) USE SAGEMAKER SERVERLESS INFERENCE

- Ideal for sporadic traffic
- No instance management
- Pay per millisecond
- Supported for TensorFlow, PyTorch, XGBoost

BUILDING AWS DATA ENGINEERING PIPELINES

END-TO-END DATA FLOW USING NATIVE AWS SERVICES

STEP 1: DATA INGESTION

Source	AWS Service	Purpose
Streaming (logs, IoT)	Amazon Kinesis Data Streams	Real-time data ingestion
Batch (files, DB dumps)	AWS Transfer Family or S3 Upload	Secure file transfer
Databases	AWS DMS (Database Migration Service)	CDC (Change Data Capture)
APIs	AWS AppFlow	SaaS app integration (Salesforce, etc.)

STEP 2: DATA STORAGE LAYER

Storage Type	Service	Use Case
Object Storage	Amazon S3	Immutable raw data, cost-effective
Data Warehouse	Amazon Redshift	Analytics, BI, SQL queries
Data Lake	Amazon S3 + Lake Formation	Governed, cataloged data lake
Archival	S3 Glacier	Long-term, low-cost storage

STEP 3: DATA CATALOGING & DISCOVERY

- Use AWS Glue Data Catalog as central metadata repository
- Run Glue Crawlers to auto-detect schema:

```
# Glue Crawler (via Console or CLI)
aws glue create-crawler --name my-crawler \
  --role arn:aws:iam::123:role/GlueServiceRole \
  --database-name my_db \
  --targets '{"S3Targets": [{"Path": "s3://bucket/raw/"}]}’
```

Output: Tables in AWS Glue Data Catalog → queryable via Athena

STEP 4: DATA TRANSFORMATION (ETL)

- Use AWS Glue Jobs (serverless Spark):
  - Auto-scaling, pay-per-job
  - Supports Python (PySpark) and Scala
- Sample Glue Job

SAMPLE GLUE JOB

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext

glueContext = GlueContext(SparkContext.getOrCreate())

# Read from catalog
datasource0 = glueContext.create_dynamic_frame.from_catalog(
    database="my_db", table_name="raw_logs"
)

# Clean and transform
cleaned = ApplyMapping.apply(
    frame=datasource0,
    mappings=[("timestamp", "string", "event_time", "timestamp"), ...]
)

# Write to curated zone
glueContext.write_dynamic_frame.from_options(
    frame=cleaned,
    connection_type="s3",
    connection_options={"path": "s3://bucket/curated/"},
    format="parquet"
)
```

STEP 5: ORCHESTRATION & SCHEDULING

- Use AWS Step Functions or Amazon EventBridge + Lambda
- Step Functions for complex workflows:
  - Ingest → Clean → Train → Deploy
  - Visual workflow + error handling
- EventBridge for event-driven pipelines:
  - Trigger Glue Job when new file lands in S3

STEP 6: QUERY & ANALYTICS

Tool	Purpose
Amazon Athena	Serverless SQL on S3 (ad-hoc queries)
Amazon Redshift	High-performance data warehouse
Amazon QuickSight	BI dashboards & visualization

```
SELECT event_type, COUNT(*)
FROM curated_logs
WHERE event_time > current_date - interval '7' day
GROUP BY 1;
```

STEP 7: MACHINE LEARNING INTEGRATION

- Use curated data from S3 to train models in SageMaker
- SageMaker Processing Jobs for feature engineering
- SageMaker Pipelines for MLOps automation:
  - Data → Train → Evaluate → Deploy

STEP 8: MONITORING & GOVERNANCE

- AWS CloudWatch: Monitor Glue jobs, Kinesis throughput
- AWS Lake Formation: Centralized data governance, row/column-level security
- AWS Config: Audit resource compliance
- CloudTrail: Track API calls

STEP 9: DATA LINEAGE & OBSERVABILITY

- Use AWS Glue Data Lineage (preview) or OpenMetadata
- Track: Raw → Curated → ML Features → Model Predictions

BUILDING ENTERPRISE DATA ENGINEERING PIPELINES ON AWS

FROM RAW DATA TO ACTIONABLE INSIGHTS — FULLY AUTOMATED & GOVERNED

PHASE 1: DATA INGESTION — CAPTURING DATA AT SCALE

- Identify Data Sources:
  - Operational databases (Oracle, SQL Server, MySQL)
  - Application logs (system, user, error logs)
  - IoT devices, sensors, clickstreams
  - SaaS platforms (Salesforce, ServiceNow, Workday)
  - Flat files (CSV, JSON, Parquet) from on-prem or partners
- Choose Ingestion Method by Velocity:
  - Real-Time Streaming: Use Amazon Kinesis Data Streams
    → Ideal for log analytics, fraud detection, live dashboards
    → Supports up to 1 MB/sec per shard
  - Batch Ingestion: Use AWS Transfer Family or S3 Batch Operations
    → For nightly ETL, financial reports, archival dumps
  - Database Replication: Use AWS DMS (Database Migration Service)
    → Enables CDC (Change Data Capture) with minimal downtime
  - SaaS Integration: Use Amazon AppFlow
    → No-code integration with Salesforce, Slack, Zendesk
- Land Raw Data in S3 Data Lake:
  - Structure as: s3://company-data/raw/<source>/<year>/<month>/<day>/
  - Enable S3 Versioning + Server-Side Encryption (SSE-S3)
  - Apply S3 Lifecycle Policies to move cold data to S3 Glacier

PHASE 2: DATA STORAGE & ORGANIZATION — THE FOUNDATION

- Use Amazon S3 as Central Data Lake:
  - Store all raw, intermediate, and curated data
  - Format data in columnar formats: Parquet (preferred), ORC, Avro
  - Avoid CSV/JSON for analytical workloads (inefficient I/O)
- Implement Data Lake Zones:
  - Raw Zone: Immutable, unprocessed data (landing area)
  - Cleansed Zone: Validated, deduplicated, schema-enforced
  - Curated Zone: Business-ready datasets (aggregated, joined, labeled)
  - Sandbox Zone: For data scientists to experiment
- Govern with AWS Lake Formation:
  - Central console to manage permissions
  - Enforce row-level and column-level security
  - Auto-revoke access for off-boarded employees
- Optional: Use Amazon Redshift for Analytics:
  - Load curated data into Redshift tables
  - Use Redshift Spectrum to query S3 directly (no ETL needed)
  - Ideal for BI tools like QuickSight, Tableau

PHASE 3: DATA CATALOGING — MAKING DATA DISCOVERABLE

- Run AWS Glue Crawlers:
  - Automatically scan S3 folders and infer schema
  - Store metadata in AWS Glue Data Catalog (Hive-compatible metastore)
  - Schedule crawlers via EventBridge (e.g., trigger on new file upload)
- Define Databases and Tables:
  - Example: Database = marketing, Table = campaign_clicks
  - Tables include: column names, data types, partition keys (e.g., dt=2025-04-01)
- Enable Data Lineage (Preview):
  - Track how raw logs → cleansed events → ML features
  - Critical for compliance (GDPR, SOX)

PHASE 4: DATA TRANSFORMATION (ETL/ELT) — PROCESSING AT SCALE

- Use AWS Glue Jobs (Serverless Apache Spark):
  - No infrastructure to manage — auto-scales from 2 to 100+ DPUs
  - Write jobs in PySpark (Python) or Scala
  - Supports custom Python libraries via Glue Job Libraries
- Sample Glue Job Workflow:
  - Read from Glue Catalog (glueContext.create_dynamic_frame.from_catalog)
  - Clean data: remove nulls, fix formats, deduplicate
  - Enrich: join with reference data (e.g., customer master)
  - Aggregate: daily summaries, rolling windows
  - Write to curated S3 zone in Parquet format
- Alternative: Amazon EMR for Advanced Workloads:
  - Use when you need custom Spark tuning, HBase, or presto
  - Better for long-running, complex analytics
- Use AWS Step Functions for Orchestration:
  - Chain: Crawler → Glue Job → Redshift COPY → SageMaker Training
  - Visual workflow with error handling and retries

PHASE 5: SCHEDULING & EVENT-DRIVEN AUTOMATION

- Trigger Pipelines via Events:
  - Use Amazon EventBridge to detect new files in S3
  - Automatically start Glue Job when data lands
- Schedule Recurring Jobs:
  - Use EventBridge Scheduler or Glue Triggers
  - Example: Run daily at 2 AM UTC
- Monitor with CloudWatch:
  - Alarms on job failures, long runtimes
  - Metrics: DPU hours, data processed

PHASE 6: DATA QUERYING & CONSUMPTION

- Ad-Hoc Analysis: Amazon Athena: Run SQL directly on S3 (no servers)
- Pay per TB scanned (cost-efficient)
- Example query:

```
SELECT event_type, COUNT(*)
FROM curated.logs
WHERE dt = '2025-04-01'
GROUP BY 1;
```

PHASE 7: SECURITY, GOVERNANCE & COMPLIANCE

- Encryption:
  - At rest: S3 SSE-S3 or SSE-KMS
  - In transit: TLS 1.3 enforced
- Access Control:
  - Use IAM roles (not users) for services
  - Apply least privilege: Glue job only accesses its S3 prefix
  - Use Lake Formation for fine-grained table/column permissions
- Auditing:
  - AWS CloudTrail: Log all API calls
  - S3 Access Logs: Track who accessed what data
  - Athena Query History: Audit SQL queries
- Data Retention:
  - Auto-delete raw data after 90 days via S3 Lifecycle
  - Archive curated data to Glacier Deep Archive ($0.00099/GB/month)

PHASE 8: MONITORING, OBSERVABILITY & COST OPTIMIZATION

- Monitor Pipeline Health:
  - CloudWatch Dashboards: Glue job duration, Kinesis lag, S3 uploads
  - SNS Alerts: Notify teams on failures
- Optimize Costs:
  - Use Glue Auto Scaling (pay only for used DPUs)
  - Compress data in Parquet (snappy, gzip)
  - Delete unused tables in Glue Catalog
- Use AWS Trusted Advisor:
  - Identify idle resources, underutilized Redshift clusters

KEY AWS SERVICES USED

Function	AWS Service
Ingestion	Kinesis, DMS, AppFlow, S3
Storage	S3, Redshift, Glacier
Catalog	AWS Glue Data Catalog
ETL	AWS Glue Jobs, EMR
Orchestration	Step Functions, EventBridge
Query	Athena, Redshift
BI	QuickSight
Governance	Lake Formation, IAM, KMS
Monitoring	CloudWatch, SNS