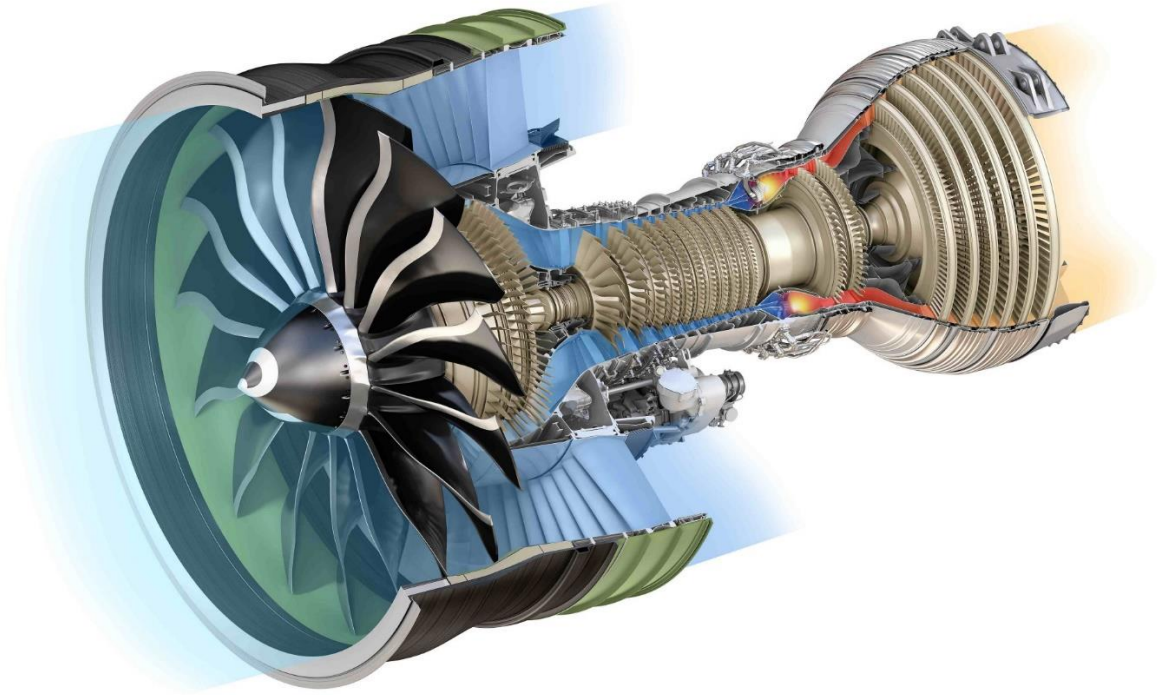
2020

Predictive Maintenance

A Case for Aviation Industry



Supervising Professor:
Prof. Madan Radhakrishnan

Authors:
SINGH Arjun
CHAKRABARTY Anik
FOTIE Kofi Dramane

EISTI

4/5/2020

Contents

Introduction

The data used in this project is the simulated running data of different aircraft's engines, provided by NASA. All engines are of the same type, but each engine starts with different state and had different use circumstances.

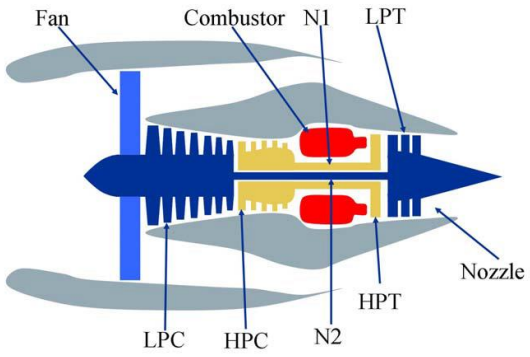
The main goal of this project is to to perform predictive maintenance on commercial turbofan engine. the predictive maintenance approach used here is a data-driven approach, meaning that data collected from the operational jet engine is used to perform predictive maintenance modeling.To be specific, the project aim is to build a predictive model to estimate the Remaining Useful Life (RUL) of a jet engine based on run-to-failure data of a fleet of similar jet engines. the algorithm documented here follows closely

Throughout the Experimenting and simulation the questions that needs to be answered are:

- How many unexploited running cycles are left for a given engine till its complete failure ?
- What causes an engine to fail (In case we came across the independent variables' labels ???)
- Binning ???

Data description

NASA has created the Prognostics and Health Management PHM08 Challenge Data Set and is being made publicly available. The data set is used to predict the failures of jet engines over time. The Prognostics CoE at NASA Ames provided the data set. The data set includes time-series measurements of various pressures, temperatures, and rotating equipment speeds that for the jet engine. These measurements are typically measured in a commercial modern turbofan engine. All engines are of the same type, but each engine starts with different degrees of initial wear and variations in the manufacturing process, which is unknown to the user. Three optional settings can be used to change the performance of each machine. Each engine has 21 sensors collecting different measurements related to the engine state at runtime. Six different flight conditions were simulated that comprised of a range of values for three operational conditions: altitude (0–42K ft.), Mach number (0–0.84), and throttle resolver angle (TRA) (20–100).



We were given 4 training datasets, 4 testing datasets as well as 4 vector Files containing the real RUL values for the model evaluation.

- The training dataset contains observations of data of aircraft engines of 100 engines, till the cycle before each engine's failure.

- The test dataset contains observations of operation data of 100 aircraft engines.
- There were 3 given operational settings related to the studied engines which can be used to change the performance of each machine.
- According to a the joined file, a total of 58 sensors were used to evaluate the state of the concerned engines, though only 21 sensor results were perserved and maintained in the Datasets.
- In the training datasets, the true RUL is not initially included. But, it can be calculated from the existing data.

For the training Datasets and the test datasets, the given columns are as follow:

- id: Engine Id. There are 100 engines. Range 1 - 100
- cycle: sequence per engine, starts from 1 to the cycle number where failure had happened
- setting1 to setting3: engine operational settings
- s1 to s21: sensors measurements in each cycle
-

'21' sensor measurements:

Symbol	*Description*	*Unit of measure*	*Label*
T2	Total temperature at fan inlet	°R	sen1
T24	Total temperature at LPC outlet	°R	sen2
T30	Total temperature at HPC outlet	°R	sen3
T50	Total temperature at LPT outlet	°R	sen4
P2	Pressure at fan inlet	psia	sen5
P15	Total pressure in bypass-duct	psia	sen6
P30	Total pressure at HPC outlet	psia	sen7
Nf	Physical fan speed	rpm	sen8
Nc	Physical core speed	rpm	sen9
epr	Engine pressure ratio (P50/P2)	--	sen10
Ps30	Static pressure at HPC outlet	psia	sen11
phi	Ratio of fuel flow to Ps30	pps/psi	sen12
NRf	Corrected fan speed	rpm	se13
NRc	Corrected core speed	rpm	sen14
BPR	Bypass Ratio	--	sen15
farB	Burner fuel-air ratio	--	sen16
htBleed	Bleed Enthalp	--	sen17
Nf_dmd	Demanded fan speed	rpm	sen18
PCNfR_dmd	Demanded corrected fan speed	rpm	sen19
W31	HPT coolant bleed	lbm/s	sen20
W32	LPT coolant bleed	lbm/s	sen21

Problem statement

the problem at hand is to come up with a machine learning model to predict RUL based on time-series data of sensor measurements typically available from aircraft gas turbine engines.

Solution Strategy:

The main strategy will be to use the dataset to train a regression model to predict RUL. Since the data is in a form of a time trajectory of many sensor data, then there will be a need to fused these sensors into a condition indicator or a health index that help in identifying the occurrence of a failure the model in testing mode will compare how similar/ correlated the testing fused signal to the training fused signal. based on this similarity comparison, a prediction is made.

Since the training data composed of run-to-failure trajectories, whereas the testing data contains trajectory up to undefined health state, then the training process will include training the model on a portion of the trajectory before the failure has occurred to simulate the real use of the model in online prediction mode.

We can translate this problem into prediction problem(Bivariate or Multivariate) to draw further inferences

- **Regression**: Time-to-Failure (TTF), for each cycle/engine, is the number cycles between that cycle and last cycle of the engine in the training data.
- **Binary Classification:** if the remaining cycles (TTF) is less than specific number of cycles (e.g. 30) then the engine will fail in this period, otherwise the engine is fine.
- **Multiclass Classification:** segmenting TTF into cycle bands (e.g. 0-15, 16-30, 30+), in which band will the engine fail? How could we improve maintenance planning?

Modeling Strategies

There are multiple modelling strategies for predictive maintenance and we will describe four of them in relation to the question they aim to answer and which kind of data they require:

- Regression models to predict remaining useful lifetime (RUL)
- Classification models to predict failure within a given time window
- Flagging anomalous behavior
- Survival models for the prediction of failure probability over time

STRATEGY 1: **Regression models to predict remaining useful lifetime (RUL)**

- OUTPUT: How many days/cycles are left before the system fails?
- DATA CHARACTERISTICS: Static and historical data are available, and every event is labelled. Several events of each type of failure are present in the dataset.
- BASIC ASSUMPTIONS/REQUIREMENTS:

Based on static characteristics of the system and on how it behaves now, the remaining useful time can be predicted which implies that both static and historical data are required and that the degradation process is smooth.

Just one type of "path to failure" is being modelled: if many types of failure are possible and the system's behavior preceding each one of them differs, one dedicated model should be made for each of them. Labelled data is available and measurements were taken at different moments during the system's lifetime.

STRATEGY 2: **Classification models to predict failure within a given time window**

Creating a model, which can predict lifetimes very accurate can be very challenging. In practice however, one usually does not need to predict the lifetime very accurate far in the future. Often the maintenance team only needs to know if the machine will fail 'soon'. This results in the next strategy:

- QUESTION: Will the machinery/part fail in the next N days/cycles?
- DATA CHARACTERISTICS: Same as for strategy 1
- BASIC ASSUMPTIONS/REQUIREMENTS: The assumptions of a classification model are very similar to those of regression models. They mostly differ on:

  Since we are defining a failure in a time window instead of an exact time, the requirement of smoothness of the degradation process is relaxed.

  Classification models can deal with multiple types of failure, as long as they are framed as a multi-class problem, e.g.:

  - ✓ class = 0 corresponding to no failure in the next n days,
  - ✓ class = 1 for failure type 1 in the next n days,
  - ✓ class = 2 for failure type 2 in the next n days
  - ✓ and so forth.

  Labelled data is available and there are "enough" cases of each type of failure to train and evaluate the model.

In general, what regression and classification models are doing is modelling the relationship between features and the degradation path of the system. That means that if the model is applied to a system that will exhibit a different type of failure not present in the training data, the model will fail to predict it.

STRATEGY 3: **Flagging anomalous behavior**

Both previous strategies require many examples of both normal behavior (of which we often have a lot of) and examples of failures. However, how many planes will you let crash to collect data? If you have mission critical systems, in which acute repairs are difficult, there are often only limited, or no examples of failures at all. In this case, a different strategy is necessary:

- QUESTION: Is the behavior shown normal?
- DATA CHARACTERISTICS: Static and historical data are available, but either labels are unknown or too few failure events were observed or there are too many types of failure
- BASIC ASSUMPTIONS/REQUIREMENTS: It is possible to define what normal behavior is and the difference between current and "normal" behavior is related to degradation leading to failure.

The generality of an anomaly detection model is both its biggest advantage and pitfall: the model should be able to flag every type of failure, despite of not having any previous knowledge about them. Anomalous behavior, however, does not necessarily lead to failure. In addition, if it does, the model does not give information about the time span it should occur.

The evaluation of an anomaly detection model is also challenging due to the lack of labelled data. If at least some labelled data of failure events is available, it can and should be used for evaluating the algorithm. When no labelled data is available, the model is usually made available and domain experts provide feedback on the quality of its anomaly flagging ability.

STRATEGY 4: **Survival models for the prediction of failure probability over time**

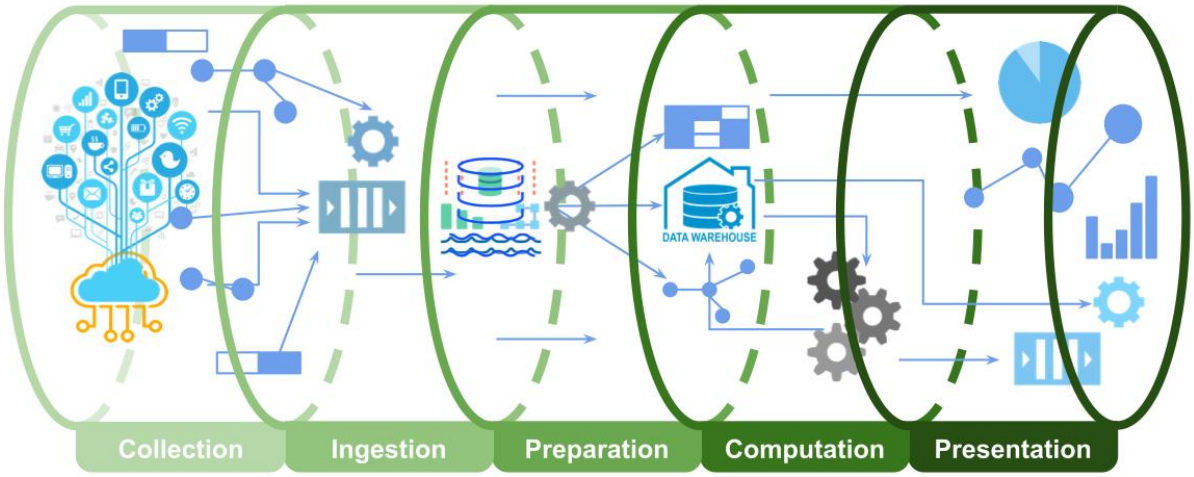
The previous three approaches focus on prediction, giving you enough information to apply maintenance before failure. If you however are interested in the degradation process itself and the resulting failure probability, this last strategy suits you best.

- QUESTION: Given a set of characteristics, how does the risk of failure change in time?
- DATA CHARACTERISTICS: Static data available, information on the reported failure time of each machine or recorded date of when a given machine became unobservable for failure.

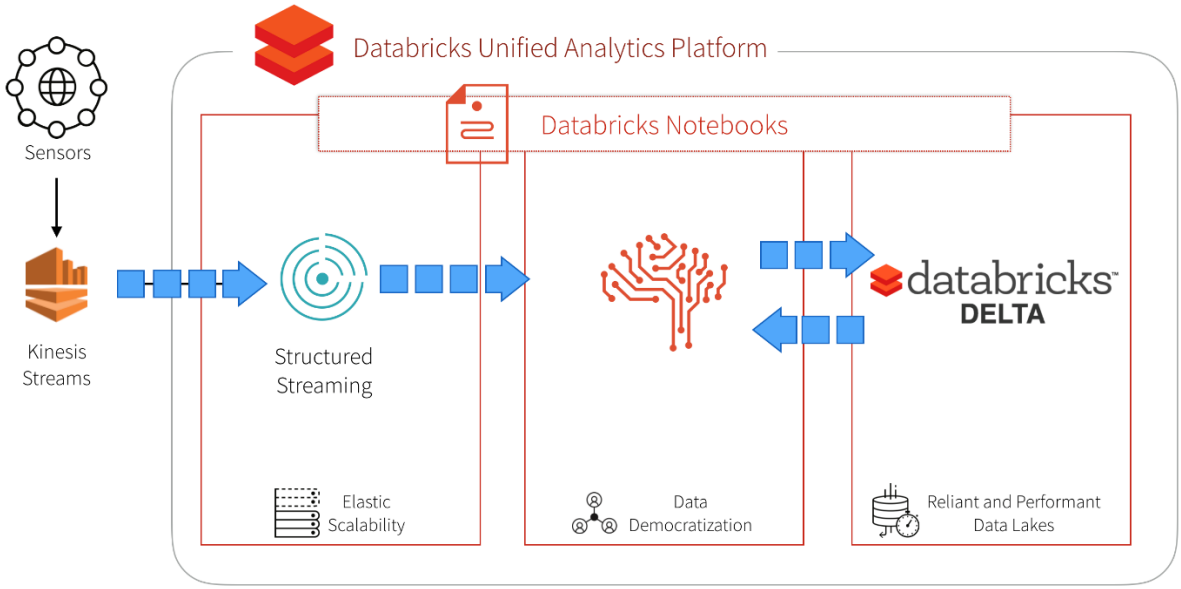
A survival model estimates the probability of failure for a given type of machine given static features and is useful to analyze the impact of certain features on lifetime. It provides, therefore, estimates for a group of machines of similar characteristics. Therefore, for a specific machine under investigation it does not take its specific current status into account.

Suggested/Recommended Model

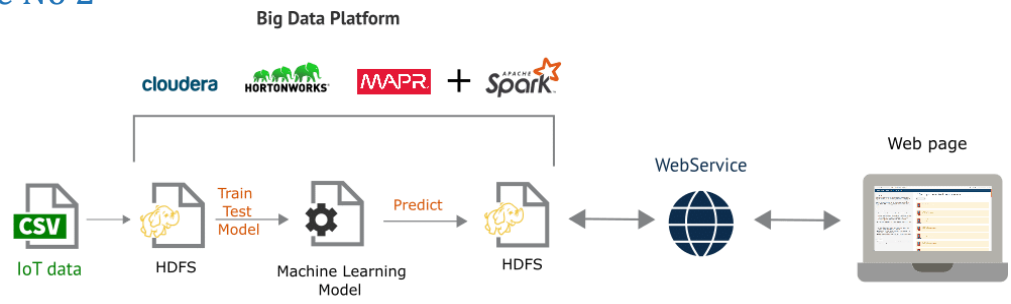
Big Data architecture



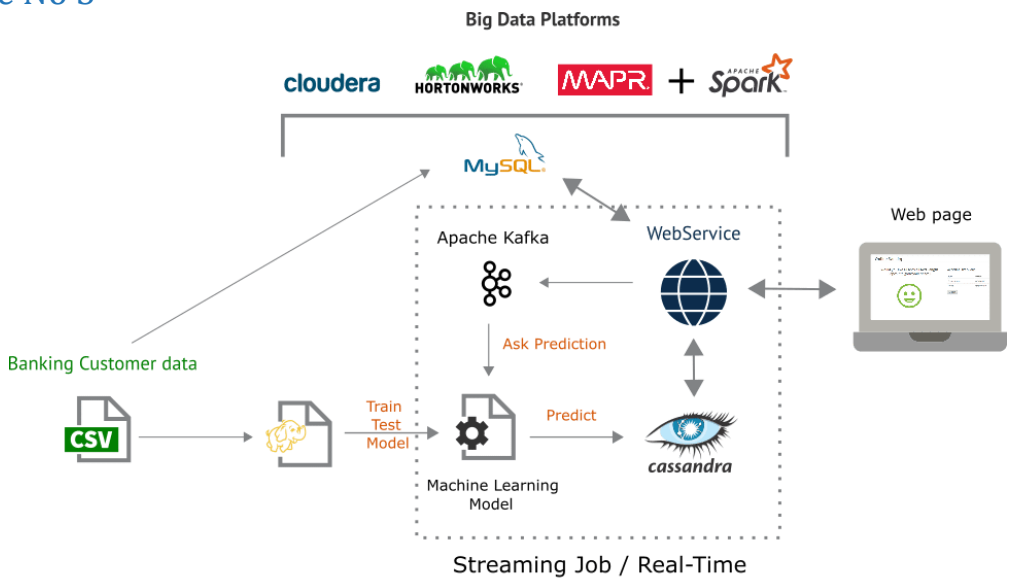
Pipeline No 1



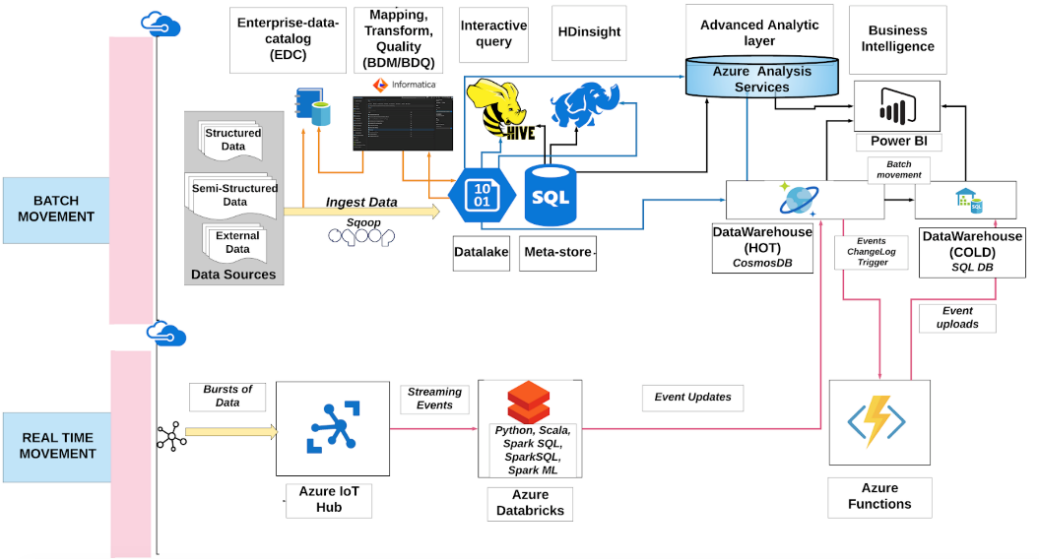
Pipeline No 2



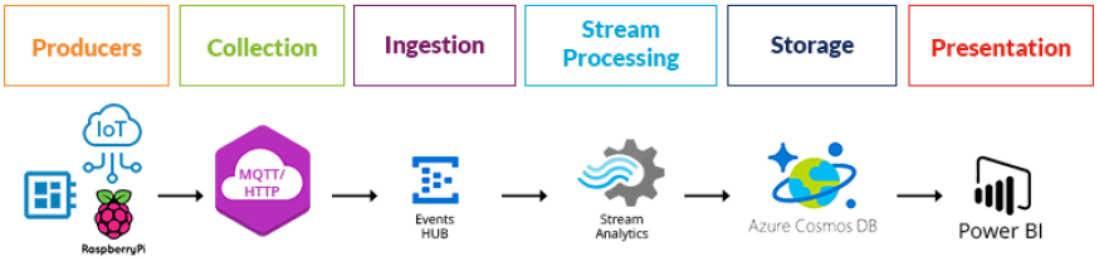
Pipeline No 3



Pipeline No 4



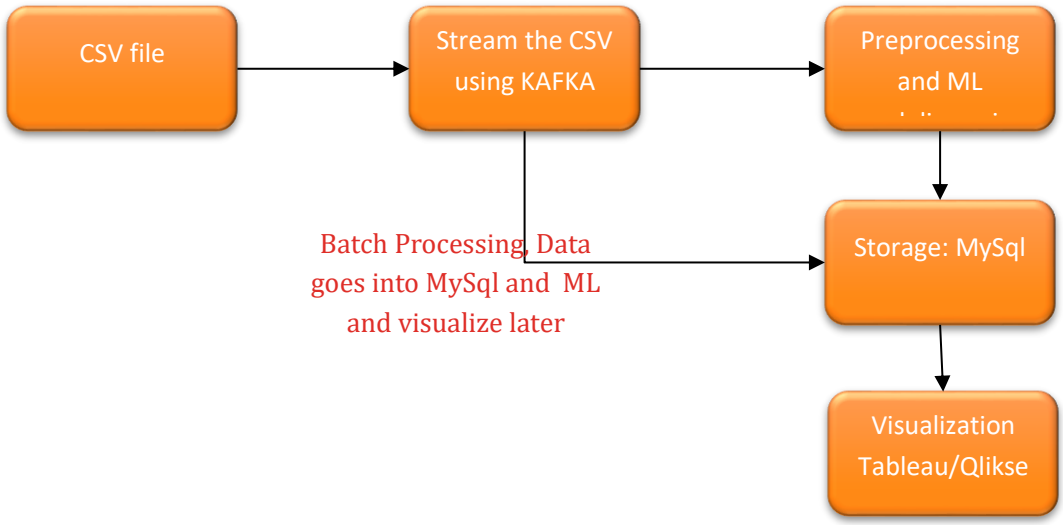
Pipeline No 5 (Amazon AWS)

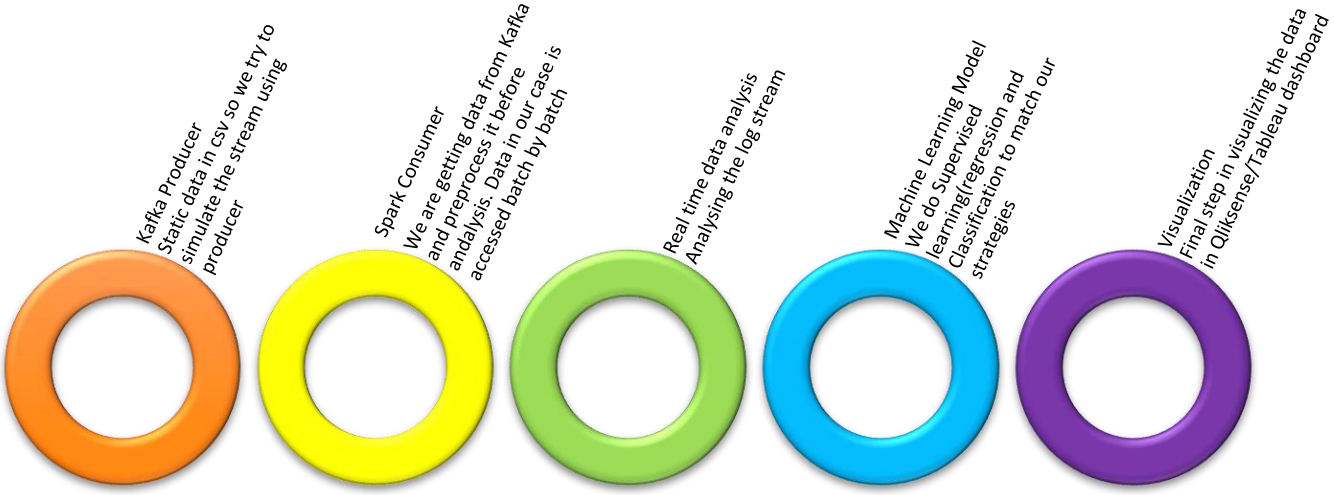


In AWS IoT architecture, Kinesis Stream is used for the data ingestion. For this, we need to define the Kinesis action which will collect data from MQTT and send it to Kinesis Analytics for further processing. After the stream-processing, the processed data is sent to Amazon Redshift and Amazon S3.In this, we can use Amazon QuickSight for the data representation. In Amazon QuickSight, we can build our visualization dashboards perform ad hoc analysis, and quickly get business insights from your data.

Model Pipeline

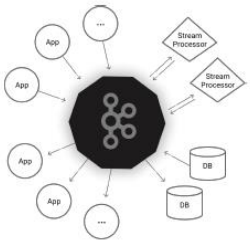
Of all the above-mentioned Pipelines Pipeline No 3 is what we planned to execute for the problem in hand





Tools Overview

Apache Kafka:
Apache Kafka is an open-source stream-processing software platform developed by LinkedIn and donated to the Apache Software Foundation, written in Scala and Java.The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. Kafka can connect to external systems (for data import/ export) via Kafka Connect and provides Kafka Streams, a Java stream processing library.



Zookeeper:
ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

PySpark:
To support Python with Spark, Apache Spark community released a tool, PySpark. Using PySpark, you can work with RDDs in Python programming language also. It is because of a library called Py4j that they are able to achieve this.

Start-to-End process

Data Ingestion

For this since we tried to experiment with streams we read the CSV as a data stream using Kafka

[1]:
```
pip install kafka-python
```

```
Requirement already satisfied: kafka-python in /opt/conda/lib/python3.7/site-
packages (2.0.1)
Note: you may need to restart the kernel to use updated packages.
```

[1]:
```
from kafka import KafkaConsumer, KafkaProducer
import json
import pandas as pd
#producer = KafkaProducer(bootstrap_servers='192.168.99.100:9092')
```

[2]:
```
#cannes_csv = pd.read_csv("Cannes2_plus.csv", delimiter=",", engine='python')
#movie_csv['timestamp'] = movie_csv['timestamp'].astype(str)
#cannes_csv = cannes_csv.drop(['timestamp'], axis=1)
#cannes_json_convert = cannes_csv.to_json("cannes.json")
```

[4]:
```
# To illustrate streaming we read csv file and then stream using kafka
train_csv = pd.read_csv('train.txt', sep=" ", header=None)
train_csv = train_csv.loc[:, train_csv.isnull().sum() < 0.8*train_csv.shape[0]]
maximum = train_csv.groupby(0).agg({1: 'max'})
maximum
train_csv = pd.merge(train_csv, maximum, on=[0])
train_csv[27] = train_csv['1_y'] - train_csv['1_x']
train_csv = train_csv.drop(columns = ['1_y'])
train_csv.columns = ['id', 'cycle', 'setting1', 'setting2', 'setting3',␣
 ↪'s1','s2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12',␣
 ↪'s13', 's14', 's15', 's16', 's17', 's18', 's19', 's20', 's21', 'ttf']
data_json_convert = train_csv.to_json("training.json")
```

[ ]:
```
#import json
from kafka import KafkaConsumer, KafkaProducer
import time
import pandas as pd
from pyspark.sql import SparkSession

KAFKA_TOPIC_NAME = "pred_model"
KAFKA_BOOTSTRAP_SERVER_CONN = "192.168.99.100:9092"

kafka_producer_object =␣
 ↪KafkaProducer(bootstrap_servers=KAFKA_BOOTSTRAP_SERVER_CONN,
                                    value_serializer=lambda x: json.dumps(x).
 ↪encode('utf-8'))

training_json = pd.read_json("training.json")
print(training_json.head(1))

traininglist= training_json.to_dict(orient="records")
print(traininglist[0])

for record in traininglist:
    print("Message to be send : ", record)
    kafka_producer_object.send(KAFKA_TOPIC_NAME,record)
    time.sleep(1)

kafka_producer_object.close()
```

```
   id  cycle  setting1  setting2  setting3      s1      s2      s3      s4  \
0   1      1   -0.0007   -0.0004       100  518.67  641.82  1589.7  1400.6

      s5  …      s13      s14     s15   s16  s17   s18  s19    s20     s21  \
0  14.62  …  2388.02  8138.62  8.4195  0.03  392  2388  100  39.06  23.419

   ttf
0  191

[1 rows x 27 columns]
{'id': 1, 'cycle': 1, 'setting1': -0.0007, 'setting2': -0.0004, 'setting3': 100,
's1': 518.67, 's2': 641.82, 's3': 1589.7, 's4': 1400.6, 's5': 14.62, 's6':
21.61, 's7': 554.36, 's8': 2388.06, 's9': 9046.19, 's10': 1.3, 's11': 47.47,
's12': 521.66, 's13': 2388.02, 's14': 8138.62, 's15': 8.4195, 's16': 0.03,
```

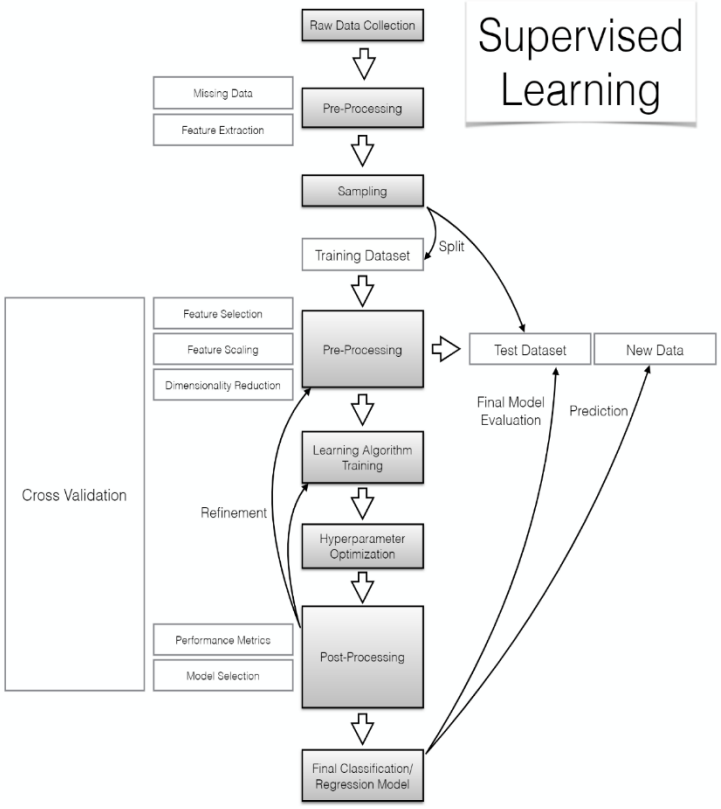
Kafka Consumer

In [1]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *

KAFKA_TOPIC_NAME_CONS = "pred_model"
KAFKA_OUTPUT_TOPIC_NAME_CONS = "outputtopic"
KAFKA_BOOTSTRAP_SERVERS_CONS = '192.168.99.100:9092'

if __name__ == "__main__":
    print("PySpark Structured Streaming with Kafka Demo Application Started ...")

    spark = SparkSession \
        .builder \
        .appName("PySpark Structured Streaming with Kafka Demo") \
        .config("spark.jars", "spark-sql-kafka-0-10_2.12-3.0.0-preview.jar,kafka-clients-2.4.1.jar") \
        .config("spark.executor.extraClassPath", "spark-sql-kafka-0-10_2.12-3.0.0-preview.jar,kafka-clients-2.4.1.jar") \
        .config("spark.executor.extraLibrary", "spark-sql-kafka-0-10_2.12-3.0.0-preview.jar,kafka-clients-2.4.1.jar") \
        .config("spark.driver.extraClassPath", "spark-sql-kafka-0-10_2.12-3.0.0-preview.jar,kafka-clients-2.4.1.jar") \
        .getOrCreate()

    spark.sparkContext.setLogLevel("ERROR")
    print(" kafka Started ...")
    # Construct a streaming DataFrame that reads from testtopic
    transaction_detail_df = spark \
        .readStream \
        .format("kafka") \
        .option("kafka.bootstrap.servers", KAFKA_BOOTSTRAP_SERVERS_CONS) \
        .option("subscribe", KAFKA_TOPIC_NAME_CONS) \
        .option("startingOffsets", "latest") \
        .load()

    print("Printing Schema of transaction_detail_df: ")
    transaction_detail_df.printSchema()
    # Write final result into console for debugging purpose
    trans_detail_write_stream = transaction_detail_df \
        .writeStream \
        .trigger(processingTime='1 seconds') \
        .outputMode("update") \
        .option("truncate", "false")\
        .format("console") \
        .start()
    trans_detail_write_stream.awaitTermination()
    spark.stop()
```

```
PySpark Structured Streaming with Kafka Demo Application Started ...
 kafka Started ...
```

Machine Learning (Predictive Modeling)

Machine learning Model



Feature Engineering Pipeline

Big Data implies model for deployment at industrial level, therefore the need for pipelining.

Feature engineering pipeline: bucketizer + vector assembler + minmax scaling+ Feature vector

```
from pyspark.ml.feature import Bucketizer
from pyspark.ml import Pipeline
bucketizer = [Bucketizer(splits=[-float("inf"),30,float("inf")], inputCol="ttf", outputCol= "binary")]
bucketizer += [Bucketizer(splits=[-float("inf"),15,30,60,float("inf")], inputCol="ttf", outputCol= "ternary")]

assembler = [VectorAssembler(inputCols= [col], outputCol="Vect_"+col)  for col in step2.columns if col not in ['id','cycle','ttf']]
minmax = [MinMaxScaler(inputCol="Vect_"+col, outputCol='scaled_'+col) for col in step2.columns if col not in ['id','cycle','ttf']]
assembler1 = VectorAssembler(inputCols= ['scaled_'+col for col in step2.columns if col not in ['ttf','id','cycle']]+['id','cycle'], outputCol="features")
stage = bucketizer + assembler + minmax+ [assembler1]
stepf = Pipeline(stages = stage).fit(step2).transform(step2).select(*['scaled_'+col for col in step2.columns if col not in ['id','cycle','ttf','binary','ternary']]
```

Best Model (Regression)

The model with low values of RMSE qualifies as the best regression model but we need to tradeoff between RMSE – R2 and Adjusted R2

Models

1) Linear Regression
2) Random forest regression (Winner)
3) Gradient boosting Decision tree regression

```
df.show()
```

```
+-----+-----------------+------------------+
|model|            train|              test|
+-----+-----------------+------------------+
|   lr|39.57671741182032|  36.5600281987422|
|   rf|35.74470326516795|27.411942754104892|
|  gbt|29.87186721643461| 38.84659908302241|
+-----+-----------------+------------------+
```

Best Model (classification)

We have use both AUC and PR to pick best(imbalanced label in classes i.e. the reason we used both)

Models

1) Logistic Regression (Winner)
2) Support Vector Machine
3) Naïve Bayes
4) Random Forest
5) Gradient Boosting

```
1 df.show()
```

▸ (3) Spark Jobs

```
+-----+------------------+------------------+------------------+------------------+
|model|         train AUC|          test AUC|          train PR|           test PR|
+-----+------------------+------------------+------------------+------------------+
|   lr|0.9896613351483171|0.9802666666666666|0.9933299465667366|0.9933299465667366|
|  svm|0.9880457149339203|0.9813333333333334|0.9936930274185863|0.9936930274185863|
|   nb|0.8020261282211265|0.7557333333333331|0.9094330350208518|0.9094330350208518|
|   rf| 0.988535080256355|0.9754666666666667|0.9917419135847867|0.9917419135847867|
|  gbt|0.9854601365019914|0.9642666666666667|0.9882893700661207|0.9882893700661207|
+-----+------------------+------------------+------------------+------------------+
```

Best for Regression Modelling is Random forest and in case of Classification(Binary classifier) is Logistic Regression

Visualization: The data had feature where only unary and binary relationship is explored. There were few limitation with the data available i.e. feature/variable description. How settings impact the engine part and which part since everything is numerical and few column having same value, it is not an easy task to do. If we had real life feed with timestamp then we can better visualize all the feature on cyclicity in the data and in return better able to advise and assist in decision making. The data is already in Mysql that can then be used to do visualizations.

Bottom line:

What is the most suitable approach for a predictive maintenance model? As for all other data science problems, there is no free lunch! The advice here is to start by understanding, which types of failure you are trying to model, which type of output you would like the model to give and which kind of data is available. Having put all this put together with the advice given above, we now know how to start and address the concern of ever growing worries of aviation industry.

References

[1] B. Republic, "Machine learning for predictive maintenance: where to start?," 29 August 2017. [Online]. Available: https://medium.com/bigdatarepublic/machine-learning-for-predictive-maintenance-where-to-start-5f3b7586acfb.

[2] "PREDICTIVE MAINTENANCE 4.0 - PRACTICAL IOT INTRO FOR VEHICLES & MACHINERY," 2010. [Online]. Available: https://www.csselectronics.com/screen/page/predictive-maintenance-can-bus-iot.

[3] M. Havlena, "Predictive Analytics Project in Automotive Industry," 26 Feburary 2014. [Online]. Available: http://www.havlena.net/en/business-analytics-intelligence/predictive-analytics-project-in-automotive-industry/.

[4] J. L. A. S. T. N. H. H. A. H. L. K. R. W. H. D. A. H. T. Conti, "Predictive maintenance for industrial products using big data," November 2014. [Online]. Available: https://patents.google.com/patent/US20140336791A1/en.

[5] M. M. C. V. Mark Haarman, "Predictive Maintenance 4.0, Predict the unpredictable," 2017.

[6] T. Joseph, "Applied Data Science: Solving a Predictive Maintenance Business Problem," 2017. [Online]. Available: https://www.kdnuggets.com/2017/10/applied-data-science-solving-predictive-maintenance-business-problem.html.

[7] B. Stuart, "Predictive Maintenance ML (IIoT)," [Online]. Available: https://www.kaggle.com/billstuart/predictive-maintenance-ml-iiot/notebook.

[8] S. P. Roshan Alwis, "Machine Learning Techniques for Predictive Maintenance," [Online]. Available: https://www.infoq.com/articles/machine-learning-techniques-predictive-maintenance/.

[9] "Predictive Modeling: The Only Guide You'll Need," [Online]. Available: https://www.microstrategy.com/us/resources/introductory-guides/predictive-modeling-the-only-guide-you-need.

[10] S. Dr. M.Hanumanthappa1, "Predicting the Future of Car Manufacturing Industry using Data Mining Techniques," *ACEEE Int. J. on Information Technology, ,* 2011.

[11] "The Automotive Industry: Driving the Future of AI," [Online]. Available: https://www.dataiku.com/stories/automotive-industry/.

[12] S. PARTHASARATHY, "Top 5 Predictive Analytics Models and Algorithms," 9 July 2019. [Online]. Available: https://www.logianalytics.com/predictive-analytics/predictive-algorithms-and-models/.

[13] "Predictive Modeling Techniques," [Online]. Available: https://www.sas.com/ko_kr/insights/analytics/predictive-modeling-techniques.html.

[14] "Predictive analytics and machine learning," [Online]. Available: https://www.sas.com/en_gb/insights/articles/analytics/a-guide-to-predictive-analytics-and-machine-learning.html.