MENDESAIN MODEL *MACHINE LEARNING* UNTUK PERAWATAN PREDIKTIF SARANA PERKERETAAPIAN

TESIS

Karya tulis sebagai salah satu syarat
untuk memperoleh gelar Magister dari
Institut Teknologi Bandung

Oleh
HAFID GALIH PRATAMA PUTRA
NIM: 23220095
(Program Magister Teknik Elektro)

INSTITUT TEKNOLOGI BANDUNG
Oktober 2020

ABSTRAK

MENDESAIN MODEL *MACHINE LEARNING* UNTUK PERAWATAN PREDIKTIF SARANA PERKERETAAPIAN

Oleh
Hafid Galih Pratama Putra
NIM: 23220095
(Program Magister Teknik Elektro)

Dalam era informasi ini, penggunaan komputer dan berbagai perangkat digital sudah tidak asing lagi dalam kehidupan sehari-hari. Perangkat tersebut dapat membantu dalam pekerjaan manusia dengan mengolah informasi dan data secara cepat. Dengan demikian, penerapan perangkat tersebut dalam industri mampu meningkatkan produktivitas. Di Indonesia beberapa sektor sedang berusaha untuk beradaptasi dan menerapkan digitalisasi dalam proses bisnisnya.

Salah satu sektor tersebut adalah industri transportasi, dimana sekarang sudah banyak sistem transportasi seperti ojek online yang memanfaatkan teknologi terkini dalam proses bisnisnya. Namun transportasi online tersebut hanya dapat digunakan untuk jarak dekat, sehingga perlu diterapkan pula untuk jarak jauh terutama antar kota. Disinilah peluang PT Kereta Api Indonesia untuk dapat melakukan Revolusi Industri, dan diantara berbagai proses bisnis yang dapat dilakukan digitalisasi, perawatan aset atau *maintenance* merupakan proses krusial dalam operasi kereta api.

Dalam tesis ini akan dibahas bagaimana penggunaan perawatan prediktif dapat bermanfaat dalam mengurangi sumber daya yang diperlukan untuk memastikan lokomotif dan kereta pembangkit dalam kondisi baik, dengan memanfaatkan kecerdasan buatan berupa pembelajaran mesin yang dapat membantu otomatisasi pemeriksaan rutin.

Kata Kunci: Perawatan prediktif, kereta api, pembelajaran mesin.

ABSTRACT

DESIGNING MACHINE LEARNING MODEL FOR PREDICTIVE MAINTENANCE OF RAILWAY VEHICLE

By
Hafid Galih Pratama Putra
NIM: 23220095
(Master's Program in Electrical Engineering)

In this information age, the use of computers and various digital devices is familiar in everyday life. These devices can assist in human work by processing information and data quickly. Thus, the application of these devices in the industry can increase productivity. In Indonesia, several sectors are trying to adapt and implement digitalization in their business processes.

One of these sectors is the transportation industry, where now many transportation systems such as online motorcycle taxis utilize the latest technology in their business processes. However, online transportation can only be used for short distances, so it needs to be applied for long distances, especially between cities. Here is the opportunity for PT Kereta Api Indonesia to carry out the Industrial Revolution. Among the various business processes that can be digitized, asset maintenance or maintenance is essential in railroad operations.

This thesis will discuss how predictive maintenance can be useful in reducing the resources needed to ensure that locomotives and generator carriage are in good condition by utilizing the artificial intelligence method of machine learning, which can help automate routine checks.

Keywords: Predictive maintenance, trains, machine learning.

MENDESAIN MODEL *MACHINE LEARNING* UNTUK PERAWATAN PREDIKTIF SARANA PERKERETAAPIAN

Oleh
HAFID GALIH PRATAMA PUTRA
NIM: 23220095
(Program Magister Teknik Elektro)

Institut Teknologi Bandung

Disetujui
Tim Pembimbing

Tanggal 30 April 2021

Pembimbing

_____

Prof. Dr.Ir. Suhono Harso Supangkat, M.Eng.

PEDOMAN PENGGUNAAN TESIS

Tesis Magister yang tidak dipublikasikan terdaftar dan tersedia di Perpustakaan Institut Teknologi Bandung, dan terbuka untuk umum dengan ketentuan bahwa hak cipta ada pada penulis dengan mengikuti aturan HaKI yang berlaku di Institut Teknologi Bandung. Referensi kepustakaan diperkenankan dicatat, tetapi pengutipan atau peringkasan hanya dapat dilakukan seizin penulis dan harus disertai dengan kaidah ilmiah untuk menyebutkan sumbernya.

Sitasi hasil penelitian Tesis ini dapat di tulis dalam bahasa Indonesia sebagai berikut:

Putra, Hafid G.P. (2020): *Mendesain Model Machine Learning Untuk Perawatan Prediktif Sarana Perkeretaapian*, Tesis Program Magister, Institut Teknologi Bandung.

dan dalam bahasa Inggris sebagai berikut:

Putra, Hafid G.P. (2020): *Designing Machine Learning Model for Predictive Maintenance of Railway Vehicle*, Master's Thesis, Institut Teknologi Bandung.

Memperbanyak atau menerbitkan sebagian atau seluruh tesis haruslah seizin Dekan Sekolah Pascasarjana, Institut Teknologi Bandung.

Dipersembahkan kepada kedua orang tua yang ku hormati, sayangi, dan bukti bakti ku atas segala bantuan yang telah kalian berikan.

KATA PENGANTAR

Puji dan syukur penulis haturkan kepada Allah SWT atas rahmat dan karunia yang telah diberikan sehingga penulis dapat menyelesaikan tesis yang berjudul "*Mendesain Model Machine Learning Untuk Perawatan Prediktif Sarana Perkeretaapian"* sebagai prasyarat untuk memperoleh gelar Magister Teknik bagi mahasiswa program S2 di Program Studi Teknik Elektro Institut Teknologi Bandung.

Dalam proses pengerjaan tesis ini, penulis mendapatkan banyak bantuan dari berbagai pihak. Oleh karena itu, penulis ingin mengucapkan terima kasih kepada:

1. Ibu dan Ayah, atas semua dukungan, perhatian, serta kasih sayang untuk penulis dalam menyelesaikan studi di program Maagister ITB.
2. Ibu Prof. Reini Wirahadikusumah, Ph.D. Selaku Rektor Institut Teknologi Bandung saat penulisan tesis.
3. Bapak Prof. Dr.Ir. Suhono Harso Supangkat, M.Eng. Selaku Dosen Pembimbing tesis penulis yang telah memberikan kesempatan untuk mendalami topik ini, serta kritik dan saran maupun arahan yang bermanfaat dalam pelaksanaan penelitian ini.
4. Bapak Dr. Y. Bandung, ST., MT. Selaku Dosen Pengajar Metode Penelitian bagi penulis yang telah banyak memberikan pengetahuan dan saran maupun arahan yang bermanfaat dalam penulisan tesis ini.
5. PT. Kereta Api Indonesia, selaku pihak kerjasama dalam penelitian tesis ini yang menyediakan kesempatan untuk dapat berkontribusi dalam perkembangan teknologi industri negara Indonesia.
6. Teman Teknik Elektro Magister Institut Teknologi Bandung Angkatan 2020 Ganjil, yang senantiasa bersama penulis berjuang dan memberikan semangat yang berarti bagi satu sama lain.
7. Semua pihak yang telah membantu dalam penyelesaian tesis ini yang tidak dapat disebutkan satu per satu.

Penulis menyadari bahwa karya tulis ini masih jauh dari kata sempurna. Oleh karena itu, kritik dan saran yang membangun sangat penulis harapkan.

Akhir kata penulis mengucapkan terima kasih kepada semua pihak yang telah membaca karya tulis ini. Semoga bermanfaat.

Bandung, Juli 2020

Penulis,

Hafid Galih Pratama Putra

NIM. 23220095

DAFTAR ISI

DAFTAR LAMPIRAN

DAFTAR GAMBAR DAN ILUSTRASI

DAFTAR TABEL

DAFTAR SINGKATAN DAN LAMBANG

SINGKATAN	Nama	Halaman awal digunakan
PT	Perseroan Terbatas	
KAI	Kereta Api Indonesia	1
ML	*Machine Learning*	3
PdM	*Predictive Maintenance*	3
RUL	*Remaining Useful Lifetime*	4
CBM	*Condition Based Maintenance*	6
DRM	*Design Research Methodology*	7
DL	*Deep Learning*	7
RC	*Research Clarification*	8
DS-I	Deskriptif Studi I	8
PS	Preskriptif Studi	9
DS-II	Deskriptif Studi II	10
ANN	*Artificial Neural Network*	15
LSTM-RNN	*Long Short Term Memory-Recurrent Neural Network*	16
SVM	*Support Vector Machine*	18
KNN	*K-Nearest Neighbor*	18
DBSCAN	*Density-Based Spatial Clustering of Applications with Noise*	29
RPM	Rotasi Per Menit	30
ECU	*Electronic Control Unit*	30
MTBM	*Mean Time Between Maintenance*	42
SMOTE	*Synthetic Minority Oversampling TEchnique*	44

BAB I PENDAHULUAN

Agar tidak terjadi kesalahpahaman mengenai istilah yang akan digunakan pada Tesis ini, akan dijelaskan pengertiannya terlebih dulu. Berdasarkan beberapa Peraturan Menteri Perhubungan Indonesia yang mendefinisikan berbagai istilah pada Perkeretaapian, sebagai berikut:

- Perkeretaapian adalah satu kesatuan sistem yang terdiri atas prasarana, sarana, dan sumber daya manusia, serta norma, kriteria, persyaratan, dan prosedur untuk penyelenggaraan transportasi kereta api [1].
- Prasarana perkeretaapian adalah jalur kereta api, stasiun kereta api, dan fasilitas operasi kereta api agar kereta api dapat dioperasikan [2].
- Sarana Perkeretaapian adalah kendaraan yang dapat bergerak di jalan reI [1].
- Kereta Api adalah sarana perkeretaapian dengan tenaga gerak, baik berjalan sendiri maupun dirangkaikan dengan sarana perkeretaapian lainnya, yang akan ataupun sedang bergerak di jalan rel yang terkait dengan perjalanan kereta api [1].
- Lokomotif adalah sarana perkeretaapian yang memiliki penggerak sendiri [1].
- Kereta yang Ditarik Lokomotif adalah kereta yang tidak mempunyai penggerak sendiri, terdiri atas: Kereta penumpang, Kereta makan, Kereta pembangkit, Kereta bagasi [3].
- Kereta adalah sarana perkeretaapian yang ditarik lokomotif atau mempunyai penggerak sendiri yang digunakan untuk mengangkut orang [4].
- Gerbong adalah sarana perkeretaapian yang ditarik lokomotif yang digunakan untuk mengangkut barang, terdiri atas: gerbong datar, gerbong terbuka, gerbong tertutup, dan gerbong tangka [5].
- Perawatan Sarana Perkeretaapian adalah kegiatan yang dilakukan untuk mempertahankan keandalan Sarana Perkeretaapian agar tetap laik operasi [6].

I.1 Latar Belakang

Dalam rangka mendukung perkambangan menuju revolusi industri 4.0 di Indonesia, maka mulai diperlukan untuk menerapkan digitalisasi pada berbagai sektor. Diantaranya adalah moda sistem transportasi Perkeretapian, yang merupakan salah satu sistem pendukung industri untuk mengirimkan kargo ataupun personel jarak jauh dengan cepat. PT Kereta Api Indonesia (Persero) sebagai salah satu penyedia layanan transportasi Kereta Api di Indonesia, perlu melakukan transformasi digitalisasi bisnis proses nya. Diantaranya adalah pada bagian operasi, yaitu kegiatan perawatan sarana.

Perawatan aset juga disebut dengan istilah *Maintenance,* merupakan bagian penting dalam kegiatan operasi kereta api untuk memastikan bahwa tidak ada masalah yang terjadi saat sedang bekerja. Ada banyak komponen yang harus dilakukan perawatan secara rutin, dan akan memerlukan sumber daya yang besar apabila diperiksa manual. Diantaranya adalah lokomotif dan kereta pembangkit, yang dapat berjumlah ratusan untuk keseluruhan armada PT. KAI.

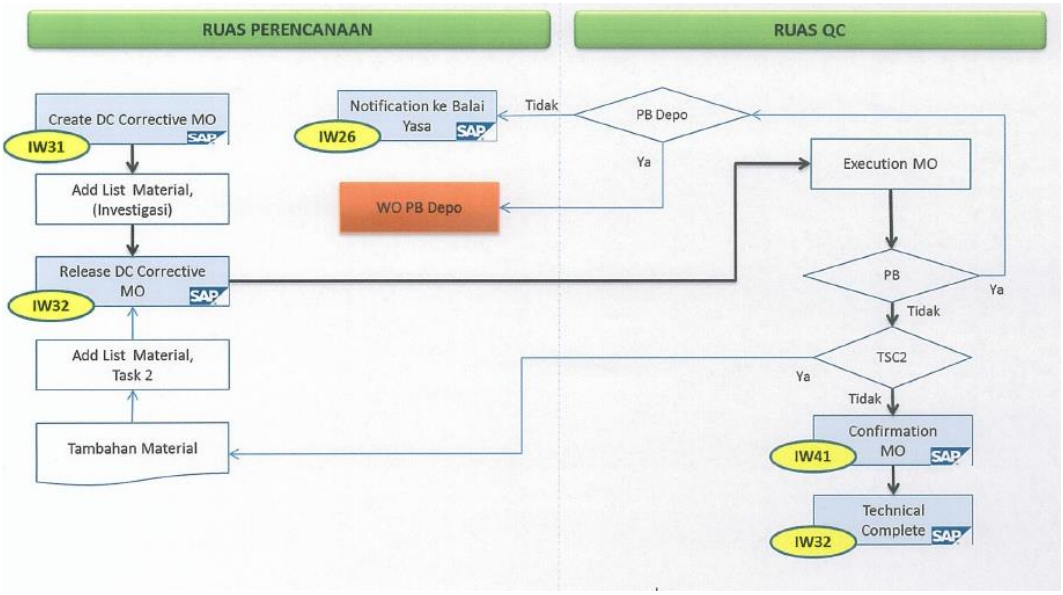
Untuk melakukan perawatan terhadap seluruh lokomotif dan kereta ini sulit tercapai. Pada tahun 2008, sekitar 7% rencana perawatan lokomotif tidak dapat direalisasikan. Selain itu juga terjadi beberapa insiden kerusakan pada tahun 2010. Pada saat itu ternyata strategi perawatan belum ditentukan oleh perusahaan, dan indikator untuk evaluasi hanya berupa *Availability* dan *Mean Kilometer Between Failure* saja. Skor yang didapat untuk kinerja perawatan masih banyak yang dapat ditingkatkan, terutama pada Kualitas dengan skor 37%, dan *Cost Effectiveness* dengan skor 53% [7]

Tabel I.1 Data ketersediaan lokomotif tahun 2019

JENIS LOK	SG Jan-Feb	SG Maret	SG April-Sep	SG Okt-Des	TSGO	SGO	TSO	SO DINAS KA	PERSENTASE	
									SO	PERAWATAN
BB 203	5	5	5	5	1	4	0	4	77%	23%
BB 204	1	0	0	0	0	0	0	0	0%	0%
BB 302	3	3	3	3	0	3	0	3	86%	14%
BB 303	20	20	18	18	4	15	1	14	78%	25%
CC 201	130	130	130	130	12	118	6	112	86%	14%
CC 202	47	47	47	47	5	42	2	40	86%	14%

CC 203	37	37	37	37	3	34	2	32	87%	13%
CC 204	37	37	37	37	3	34	2	32	88%	12%
CC 205	55	55	55	55	4	51	3	48	88%	12%
CC 206	150	150	150	150	5	145	3	143	95%	5%
Jumlah	**485**	**484**	**482**	**482**	**36**	**447**	**18**	**429**	89%	11%

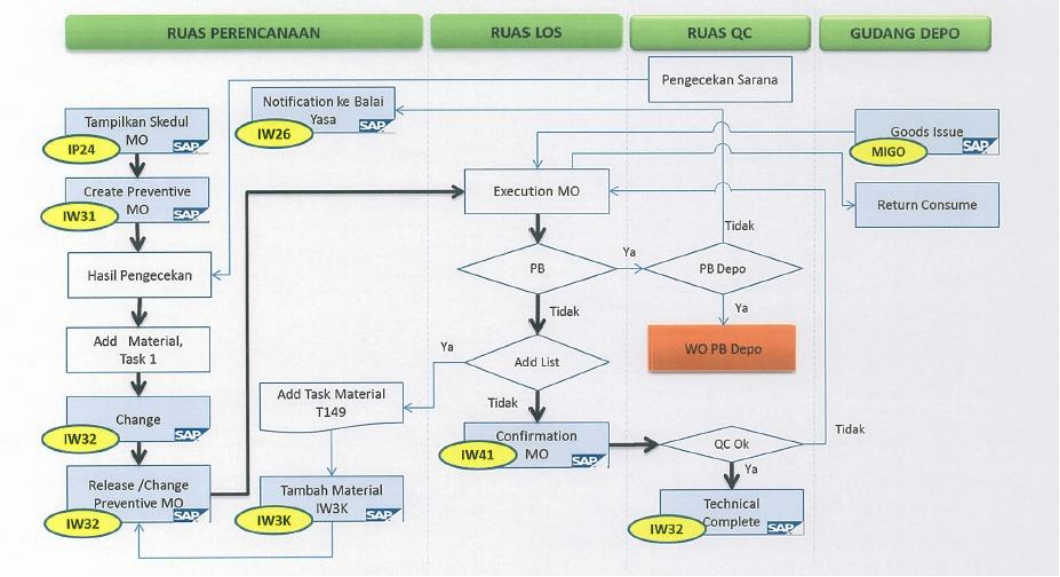
Pada tabel tersebut SG (siap guna) merupakan total sarana lokomotif yang tersedia, TSGO (Tidak siap guna operasi) mengacu pada sarana yang sedang dilakukan perbaikan di balai yasa, SGO (Siap guna operasi) merupakan hasil dari SG-TSGO, kemudian TSO (Tidak siap operasi) adalah Sarana yang di rawat di depo, dan SO (siap operasi) merupakan hasil SGO-TSO. Berdasarkan data tersebut, sekitar 11% dari lokomotif perlu dilakukan perawatan sehingga tidak dapat digunakan. Kebijakan batas kendali dari KAI membutuhkan setidaknya 85% dari armada untuk siap digunakan. Dengan demikian memang target tersebut sudah tercapai, tetapi ini merupakan rekap data selama satu tahun. Dalam operasinya pada rentang waktu mingguan atau bulanan angka tersebut dapat jatuh dibawah batas kendali tersebut, terutama pada jenis lokomotif tertentu yang sudah tua.



Gambar I.1 Bisnis Proses *Daily Check* pada Sarana KAI

Melihat dari bisnis proses perawatan pada KAI, terbagi menjadi dua yaitu Harian dan Bulanan. Aplikasi *enterprise resource management* seperti SAP sudah dipakai oleh PT KAI dalam merencanakan bisnis prosesnya, namun masih ada kegiatan yang datanya belum dalam bentuk digital sehingga tidak tercatat pada sistem.

Aktivitas ‘Execution MO’ dilakukan oleh mekanik secara manual mengacu pada kertas *checksheet* untuk menentukan kondisi keperluan perbaikan (PB). Jika dibutuhkan perbaikan kecil maka dapat dilakukan di depo, sementara jika perlu perbaikan yang cukup besar maka akan diberikan Notifikasi ke Balai Yasa untuk dikerjakan. Dari proses ini, apabila data komponen yang diperbaiki dan waktu perbaikan tersimpan datanya oleh sistem, maka dapat dilakukan prediksi *Remaining Useful Lifetime* (RUL) dengan ML jika jumlah datanya cukup banyak.



Gambar I.2 Bisnis Proses Perawatan Bulanan Sarana KAI

Dari bisnis proses perawatan bulanan pada KAI, beberapa bagian cukup serupa dengan sebelumnya. Hasil dari aktifitas ‘Pengecekan Sarana’ yang dilakukan harian akan menjadi masukan untuk perintah kerja perawatan. Aktivitas ‘Execution MO’ dilakukan oleh mekanik secara manual mengacu pada kertas *checksheet* untuk menentukan kondisi keperluan perbaikan (PB). Jika dibutuhkan perbaikan kecil maka dapat dilakukan di depo, sementara jika perlu perbaikan yang cukup besar maka akan diberikan Notifikasi ke Balai Yasa untuk dikerjakan. Jika tidak ada perbaikan lebih lanjut maka perintah kerja selesai, namun akan dilakukan *quality control* (QC) untuk memastikan tidak ada masalah pada sarana sebelum proses selesai. Dari proses ini, menggunakan informasi historis QC sebelumnya dan data berbagai sensor dapat dimanfaatkan untuk prediksi kondisi sarana menggunakan ML jika jumlah datanya cukup.

Disinilah diperlukannya otomasi untuk dapat mengidentifikasi kondisi lokomotif dan sarana lainnya dari kereta api untuk memastikan kelayakan operasinya. Jika bergantung pada inspeksi rutin periodik secara manual, belum tentu mesin memiliki masalah saat diperiksa, selain itu juga memakai sumber daya manusia dan waktu karena mengecek mesin yang kondisinya masih baik. Sementara itu dengan konsep perawatan prediktif, maka inspeksi hanya diperlukan saat mesin terdeteksi berdasarkan perkiraan akan memiliki masalah yang dapat mengganggu operasi, dan apabila terverifikasi maka kemudian dapat dilakukan perawatan. Dengan demikian akan mengurangi waktu saat mesin tidak dapat digunakan (meningkatkan *Availability*), dan dapat memaksimalkan keuntungan dengan mengoperasikan mesin serta mengurangi sumber daya yang digunakan (meningkatkan *Cost Effectiveness*).

Manfaat ini dibuktikan oleh French National Railways (SNCF) yang telah mulai menerapkan sistem diagnosis jarak jauh yang terpasang secara *on-board* pada sarana perkeretaapian. Sistem ini dapat digunakan untuk mendeteksi kesalahan, kode status, dan berbagai parameter sensor terkait. Data yang didapat kemudian diolah dan dianalisa untuk menjadi indikator yang dapat menentukan degradasi sistem, hasil berupa status kondisi sistem kemudian dapat digunakan untuk mempertimbangkan keperluan Perawatan. Strategi perawatan baru menggabungkan perawatan terjadwal (*Preventive*) dan *Condition Based Maintenance* (CBM), dengan tujuan meningkatkan keselamatan, ketersediaan, keandalan, dan mengurangi frekuensi perawatan terjadwal. Namun, beberapa tugas perawatan terjadwal, seperti pelumasan atau inspeksi visual, masih digunakan karena memerlukan campur tangan manusia. Hasil yang didapatkan dari pengamatan selama 5 tahun terakhir menunjukan: jumlah kerusakan berkurang setengahnya, sementara efisiensi alat *troubleshooting* telah diperkuat oleh alat analisis data. Peningkatan 30% pada pintu masuk/keluar di depot perawatan, yang menunjukkan keuntungan dua kereta per lokasi. Pemotongan 20% dalam jumlah kereta api yang dihentikan layanannya untuk perawatan, dan pengurangan 20% dalam biaya perawatan [8].

I.2 Masalah Penelitian

Berdasarkan latar belakang yang telah dikemukakan, dibuat pernyataan masalah yang akan dikerjakan dalam penelitian tesis ini yaitu: Mendesain program pembelajaran mesin untuk melakukan prediksi kebutuhan perawatan sarana berdasarkan rekaman data sensor yang dapat digunakan untuk menggantikan inspeksi rutin yang dilakukan manual sehingga dapat meningkatkan ketersediaan sarana.

Batasan dalam penelitian tesis ini adalah:

1. Dataset untuk melatih *Machine Learning* didapat dengan bekerjasama oleh pihak PT. KAI berdasarkan data sensor untuk kereta pembangkit yang beroperasi di Pulau Jawa tahun 2020 dan 2021.
2. Pengujian terbatas hanya pada komponen kelistrikan dari sarana kereta.
3. Prediksi keperluan perawatan sarana ditunjukkan dalam bentuk keterangan hasil klasifikasi antara Outlier, Normal atau Maintenance dan hasil regresi waktu RUL berdasarkan input nilai sensor yang digunakan sebagai fitur.

I.3 Tujuan, Kontribusi, dan *Benefit* Penelitian

Tujuan Penelitian Tesis ini secara umum adalah:

1. Menganalisa proses perawatan yang diterapkan PT.KAI saat ini.
2. Membuat program pembelajaran mesin untuk sistem perawatan prediktif.
3. Mendesain berbagai alternatif opsi model pembelajaran mesin untuk perawatan prediktif sarana kereta api.
4. Mengevaluasi kinerja model pembelajaran mesin.

Kontribusi dari penelitian Tesis ini dibandingkan lainnya adalah objek penelitian berfokus pada kereta pembangkit, membandingkan berbagai algoritma dan opsi menggunakan data real dari sensor, dan menghasilkan prediksi kondisi yang ditunjukkan dalam bentuk keterangan hasil klasifikasi antara Outlier, Normal atau Maintenance berdasarkan input nilai sensor yang digunakan sebagai fitur ML. Sementara itu dalam bidang keilmuan Rekayasa dan Manajemen Teknik Keamanan Informasi, kontribusi yang diberikan adalah manfaat penggunaan model ML untuk PdM yang dapat menjadi alternatif solusi manajemen resiko keamanan informasi untuk proses bisnis Perawatan sarana pada PT.KAI. Dengan hasil penelitian ini, diharapkan proses perawatan bisa menjadi lebih baik dan mengurangi insiden

kerusakan akibat informasi yang tidak akurat karena beberapa komponen dinilai secara kualitatif tanpa alat ukur yang jelas saat perawatan dilakukan menggunakan cara manual seperti saat ini. Selain itu melihat dari sudut pandang ketersediaan (*Availability*) informasi perawatan, dapat ditingkatkan karena tidak lagi terbatas pada kertas, tetapi sudah dalam bentuk digital dan bisa diakses melalui internet. Berdasarkan aspek integritas (*Integrity*) informasi, dengan menggunakan verifikasi *hashing* dari file data masukan untuk prediksi perawatan dapat dipastikan bahwa tidak ada perubahan terhadap isi file.

Berdasarkan tujuan yang telah dikemukakan sebelumnya, dan dengan memahami kondisi saat ini, dapat dianalisa keuntungan (*benefit*) penelitian ini bagi PT.KAI adalah sebagai berikut.

Tabel I.2 Analisa Benefit Penelitian bagi PT.KAI

No	Kondisi Saat ini	Hasil Penelitian
1	Sebagian istilah belum jelas mengenai berbagai proses Maintenance	Mendefinisikan sebagian istilah pada proses Maintenance
2	Data sensor hanya di monitor dan disimpan saja.	Data sensor diolah untuk mendapatkan prediksi kondisi serta perkiraan RUL
3	Pengecekan kondisi manual oleh manusia berdasarkan waktu tertentu	Pengecekan kondisi otomatis oleh komputer berdasarkan kondisi mesin yang dioperasikan manusia
4	Belum ada penerapan ML maupun PdM untuk perawatan sarana.	Menerapkan ML dan PdM untuk perawatan sarana Kereta Pembangkit

I.4 Teknik Penelitian

Bagian ini menjelaskan teknik yang dilakukan dalam menjalankan langkah-langkah penelitian dan proses pembuatan program ML.

1. *Studi Literatur*

   Teknik ini berfungsi untuk dapat merumuskan tujuan dan permasalahan terhadap penelitian tesis ini. Dengan melakukan kajian yang meliputi dasar teori berkaitan topik tesis. Seperti konsep perawatan secara umum, pembelajaran mesin, perangkat lunak tersedia yang dapat digunakan, serta

algoritma yang sesuai untuk diterapkan. Data dan informasi yang diperoleh bersumber dari buku, jurnal ilmiah, dan internet.

2. *Studi Empiris*

   Teknik ini bertujuan untuk meningkatkan pemahaman terhadap penelitian tesis ini. Dengan mengetahui bagaimana proses perawatan dilakukan saat ini oleh PT. KAI berdasarkan proses bisnis dan laporan perawatan yang tersedia. Kemudian mencari penelitian terkait perawatan prediktif pada kereta api, dan menganalisa hasil yang didapatkan serta metode yang digunakan.

3. *Eksperimen*

   Teknik ini digunakan untuk menentukan hasil terbaik dari model pembelajaran mesin yang dibuat terhadap kasus perawatan prediktif untuk sarana kereta api. Dengan membandingkan beberapa algoritma, aturan Hyperparameter serta fitur yang akan memberikan hasil terbaik jika diterapkan pada kasus penelitian ini. Menggunakan Bahasa pemrograman Python versi 3 yang diterapkan pada aplikasi pengembangan program Google Colab serta *library* yang tersedia untuk umum. Parameter utama yang akan menjadi kriteria penilaian adalah Akurasi dan *Runtime*.

BAB II TINJAUAN PUSTAKA

Pada bagian ini akan dijelaskan detail mengenai berbagai hasil studi literatur yang dilakukan untuk penelitian tesis berkaitan seperti penjelasan konsep perawatan, dan kecerdasan buatan. Membandingkan dengan riset yang serupa, atau peluang untuk mengembangkan dari riset yang sudah ada sebelumnya.

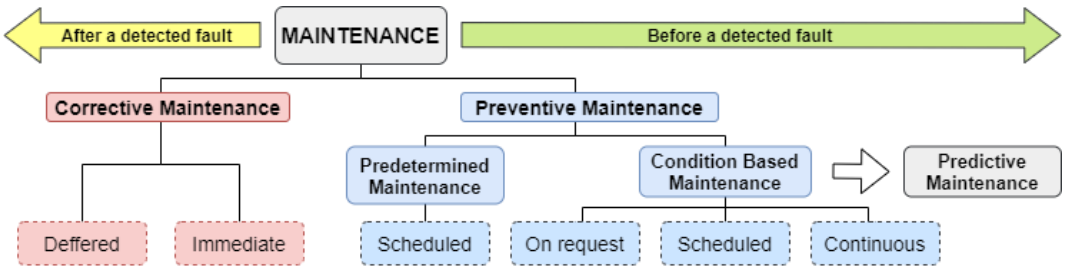
II.1 Konsep *Maintenance*

Yang dimaksud dengan istilah 'perawatan' adalah: kombinasi dari semua tindakan selama siklus hidup suatu barang yang dimaksudkan untuk mempertahankan atau memulihkannya ke keadaan di mana barang tersebut dapat melakukan fungsi yang diperlukan [9]. Untuk dapat menentukan suatu kondisi yang dapat dianggap baik akan bergantung pada berbagai macam faktor seperti kondisi operasi, tipe industri, dan tujuan bisnis. Meski demikian secara umum dapat ditentukan berdasarkan parameter berikut [10]:

- *Performance*, yaitu kemampuan suatu mesin untuk dapat menjalankan fungsinya.
- *Downtime*, merupakan periode saat mesin tidak bekerja yang harus diatur dalam level yang dapat ditoleransi.
- *Service Life*, adalah lama waktu yang diperlukan hingga mesin harus digantikan; penting untuk dipertimbangkan sehingga dapat memberikan keuntungan dari investasi terhadap mesin tersebut.
- *Efficiency*, yaitu rasio yang mengukur produktivitas kinerja mesin dengan sumber daya yang digunakan; harus selalu berada dalam level yang dapat diterima.
- *Safety*, merupakan ukuran yang dapat menjamin keamanan personil dari risiko bahaya yang mungkin terjadi.
- *Environmental impact*, yaitu ukuran yang menjamin dampak penggunaan mesin aman terhadap lingkungan dan perangkat sekitar.
- *Cost*, adalah parameter yang mengukur sumber daya yang digunakan pada mesin; diantaranya adalah biaya untuk perawatan harus berada dalam rentang yang dapat diterima.

Tujuan dari perawatan adalah untuk dapat memastikan bahwa mesin berada dalam kondisi yang baik dengan mempertimbangkan faktor tersebut. Dengan demikian maka dikembangkan berbagai strategi perawatan berbeda sebagai berikut.

II.1.1 Konsep Predictive Maintenance



Gambar II.1 Jenis strategi perawatan berdasarkan standar EN13306 [11]

Dalam melakukan perawatan, ada beberapa strategi atau sistem yang dapat diterapkan. Diantaranya adalah Pencegahan (*Preventive*), dan Pembenaran (*Corrective*). Strategi pencegahan digunakan untuk merawat suatu barang agar dapat bekerja dengan baik sebelum terjadi kerusakan, sementara strategi pembenaran digunakan setelah terjadi kerusakan. Dari kategori pencegahan terbagi lagi menjadi dua, perawatan rutin (*Predetermined Maintenance*) merupakan strategi yang menjadwalkan perlunya melakukan perawatan mesin secara periodic. Perawatan berbasis kondisi (*Condition based Maintenance*) adalah strategi baru yang dapat dilakukan sesuai keperluan ataupun terjadwal berdasarkan pengamatan kondisi mesin. Pemelihaan prediksi (*Predictive Maintenance*) adalah tahapan lanjut dari CBM yang dapat menggunakan bantuan sensor dan pembelajaran mesin. Jenis perawatan ini dibantu dengan menggunakan sensor untuk memantau berbagai parameter yang berpengaruh dalam mesin atau sistem, berdasarkan data yang didapat dan tren historis sebelumnya maka dapat dianalisis apakah diperlukan perawatan atau tidak. Data secara terus menerus dianalisis dan dimonitor untuk mengevaluasi kondisi sistem dan memberikan prediksi jika perlu dilakukan perawatan sebelum dapat terjadi kerusakan [9].

Melihat kembali pada gambar yang menjelaskan bisnis proses perawatan harian dan bulanan sarana pada KAI, dapat terlihat ada beberapa istilah yang tidak sesuai

artinya jika dibandingkan dengan penjelasan referensi [9]. Beberapa istilah yang menjadi permasalahan tersebut adalah:

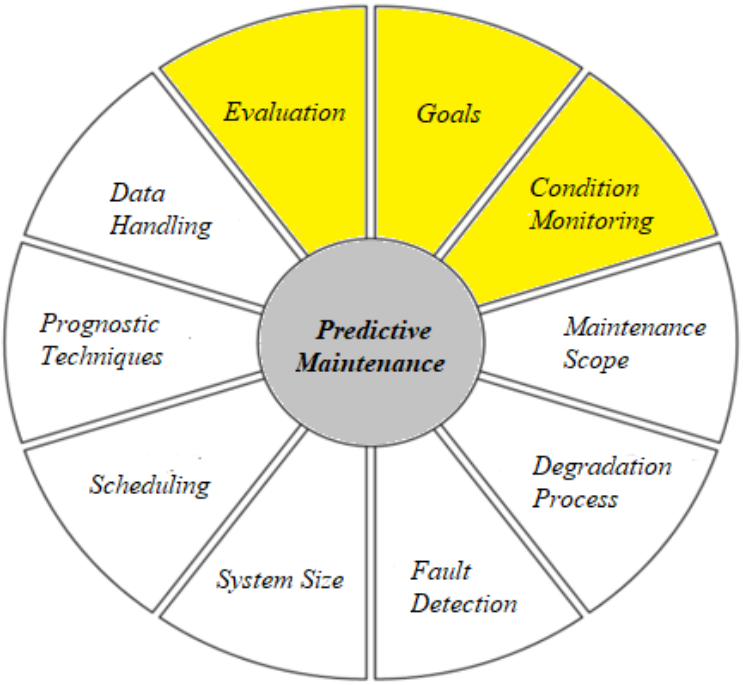
- Daily Check, kegiatan pengecekan kondisi sarana yang dilakukan dengan periode Harian
- Perawatan Bulanan, kegiatan pengecekan kondisi sarana yang dilakukan dengan periode Bulanan
- Perbaikan (PB), kegiatan untuk membenarkan masalah sehingga penggunaan mesin menjadi normal kembali
- Corrective MO, perintah untuk melakukan kegiatan Daily Check
- Preventive MO, perintah untuk melakukan kegiatan Perawatan Bulanan

Ketidaksesuaian pengertian terkait istilah tersebut perlu diperbaiki dan diperjelas dengan membuat definisi formal atau standar yang dapat disahkan oleh pemerintah. Pada penelitian ini akan disesuaikan terhadap referensi [9], yang menghasilkan definisi dan perubahan istilah menjadi sebagai berikut:

- Perawatan Harian, kegiatan perawatan kondisi sarana yang dilakukan dengan periode Harian
- Perawatan Bulanan, kegiatan perawatan kondisi sarana yang dilakukan dengan periode Bulanan.
- Perawatan/Pemeliharaan, kombinasi dari semua kegiatan teknis, administratif dan tindakan manajerial selama siklus hidup item dimaksudkan untuk mempertahankan, atau mengembalikannya pada keadaan yang dapat melakukan fungsi yang diperlukan.
- Perbaikan (PB), tindakan fisik yang diambil untuk memulihkan fungsi yang diperlukan dari barang yang rusak.
- Inspeksi, pemeriksaan kesesuaian dengan mengukur, mengamati, atau menguji karakteristik yang relevan dari suatu barang.
- Pencegahan, kegiatan yang dilakukan untuk menilai atau mengurangi degradasi dan probabilitas kegagalan item

Dari survey [12] mengenai penelitian PdM, dapat diterapkan dengan berbasis ML tradisional ataupun menggunakan DL (*Deep Learning*). Kompleksitas model pembelajaran mesin, khususnya model DL terus meningkat setiap tahun. Salah satu

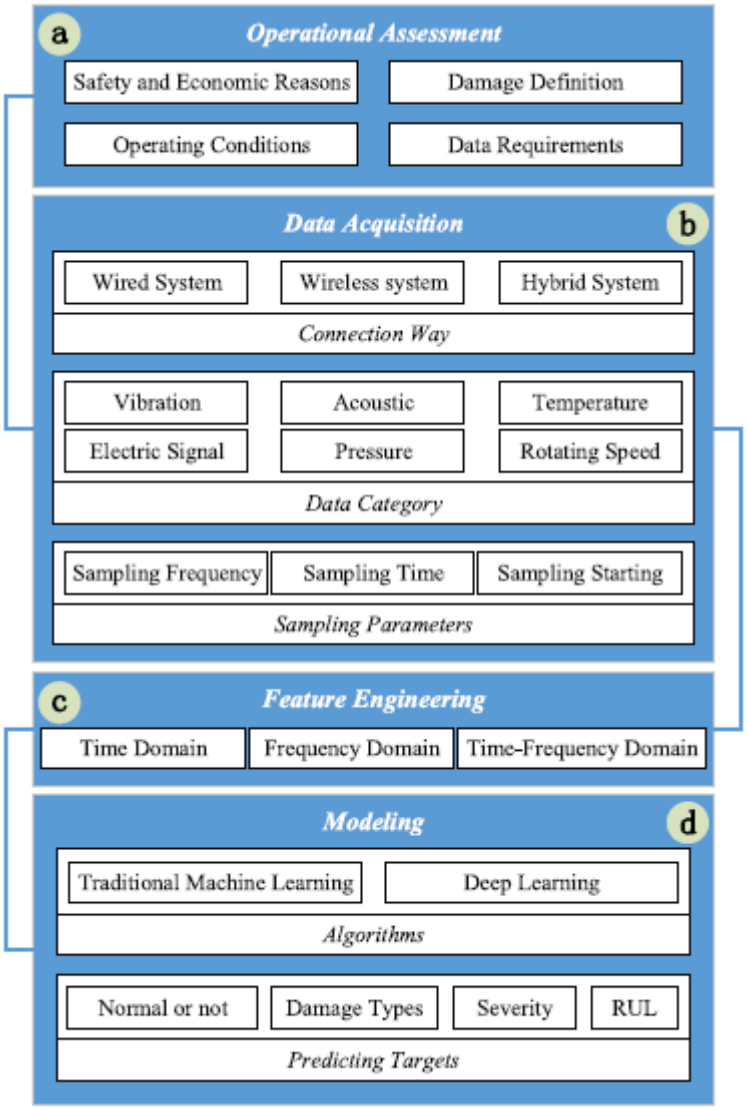
hal yang menyebabkan ini adalah meningkatnya jumlah lapisan yang digunakan di dalam *Neural Network*. Dengan meningkatnya kompleksitas serta jumlah data yang akan diolah, maka akan meningkatkan waktu eksekusi program. Tidak peduli apakah program melalui pelatihan menggunakan *cloud* ataupun secara lokal, maka akan memerlukan biaya yang besar. Jika menyewa penggunaan GPU terbaru pada *cloud* untuk melatih DL dapat mengeluarkan biaya beberapa dolar per jam, maupun menggunakan sumber daya komputasi lokal akan membutuhkan biaya tambahan untuk listrik dan pendingin udara [13]. Saat ini PT.KAI baru mulai menerapkan sistem *data collection* pada beberapa kereta pembangkit, dengan data yang tersedia sekitar 1 tahun terakhir. Selain itu perawatan rutin masih dilakukan menggunakan referensi dari kertas *checksheet*, sehingga informasinya belum tersimpan dalam bentuk digital. Keterbatasan jumlah data ini menyebabkan penggunaan DL tidak terlalu dibutuhkan untuk saat ini. Selain itu infrastruktur dan arsitektur TI (Teknologi Informasi) yang belum mapan oleh PT.KAI akan meningkatkan kebutuhan biaya jika menerapkan DL. Karena proses transformasi digital ini dilakukan secara bertahap sesuai kemampuan organisasi, maka untuk tahap awal perlu menyediakan solusi dengan biaya murah yang dapat menjadi basis untuk perkembangan tahap selanjutnya. Berdasarkan pertimbangan tersebut, maka penelitian ini difokuskan pada pendekatan yang menggunakan ML tradisional. Dari survey [12] juga disampaikan beberapa jenis algoritma yang umum digunakan pada sistem perawatan prediktif, yaitu: Artificial Neural Network (ANN), Decision Tree (DT), Support Vector Machine (SVM), dan k-Nearest Neighbor (k-NN). Algoritma tersebut dapat dipertimbangkan penggunaannya, dengan membandingkan performanya untuk mengetahui algoritma yang cocok pada kasus penelitian ini.



Gambar II.2 Kategori dari kerangka untuk perawatan prediktif [14]

Hasil penelitian terhadap beberapa literatur mengenai PdM menghasilkan gambar yang menyatakan bahwa strategi perawatan prediktif memiliki 10 kategori besar kerangka (*framework)* yang dapat diterapkan dalam menyusun sistem. [14] Pada penelitian tesis ini, akan berkaitan dengan monitor kondisi (*Condition Monitoring*) kereta pembangkit dan evaluasi (*Evaluation*) kebutuhan perawatan berdasarkan data dari sensor, serta tujuan (*Goals*) untuk meningkatkan ketersediaan (*Availability*) dari sarana yang siap beroperasi.

Dari studi yang dilakukan [15], disimpulkan bahwa penggunaan kecerdasan buatan dan teknik pemodelan yang dibuat cukup membantu untuk mencapai tujuan melakukan *Predictive Maintenance*. Dengan konsep ini, memungkinkan perawatan dilakukan dengan lebih efisien, karena pengawasan terhadap kondisi mesin atau sistem dilakukan secara kontinu tanpa menggangu kinerja. Berdasarkan penelitian lainnya, didapatkan hasil biaya perawatan yang lebih rendah apabila digunakan metode perawatan prediktif dengan bantuan sensor dibandingkan metode lainnya [16].



Gambar II.3 Proses implementasi PdM [17]

Dari hasil survey yang dilakukan [17] dibuat langkah proses implementasi *Predictive Maintenance* yang secara umum dilakukan oleh para peneliti dalam membuat sistem. Penelitian ini mengikuti langkah yang serupa dengan proses ini, terutama saat akan mendesain program pembelajaran mesin yang dapat diterapkan untuk perawatan prediktif lokomotif dan kereta pembangkit.

II.1.2 Pemantauan Kondisi Komponen

Manfaat utama dari algoritma pemeliharaan prediktif adalah hasil model deteksi atau model prediksi. Model ini menganalisis indikator kondisi yang diekstraksi

untuk menentukan kondisi sistem saat ini (deteksi kesalahan dan diagnosis) atau memprediksi kondisi masa depan (prediksi RUL) [18].

Condition Monitoring menggunakan data dari mesin untuk menilai kondisinya saat ini dan untuk mendeteksi dan mendiagnosis kesalahan pada mesin. Data mesin adalah data seperti pengukuran suhu, tekanan, tegangan, kebisingan, atau getaran, yang dikumpulkan menggunakan sensor khusus. Algoritma *Condition Monitoring* memperoleh metrik dari data yang disebut indikator kondisi. Indikator kondisi adalah fitur apa pun dari data sistem yang perilakunya berubah dengan cara yang dapat diprediksi saat sistem menurun. Indikator kondisi dapat berupa kuantitas apa pun yang diturunkan dari data yang mengelompokkan status sistem yang serupa bersama-sama, dan menetapkan status yang berbeda secara terpisah. Dengan demikian, algoritma pemantauan kondisi dapat melakukan deteksi atau diagnosis kesalahan dengan membandingkan data baru dengan penanda kondisi kesalahan yang sudah ada [18].

Condition Monitoring mencakup pembedaan antara keadaan rusak dan keadaan yang sehat (deteksi kerusakan) atau, ketika ada keadaan kerusakan, menentukan sumbernya (diagnosis kerusakan). Untuk merancang algoritma *Condition Monitoring*, dapat menggunakan indikator kondisi yang diekstrak dari data sistem untuk melatih model yang dapat menganalisis indikator yang diambil dari data uji untuk menentukan status sistem saat ini [19].

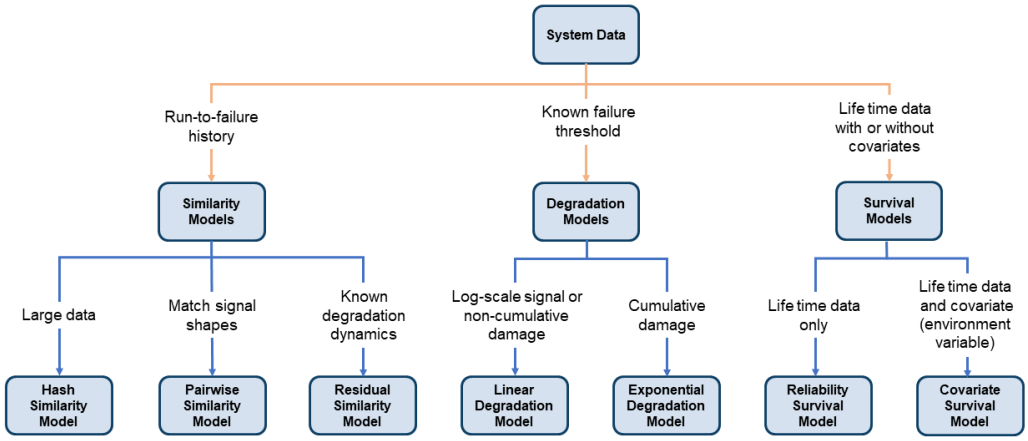
Berdasarkan referensi [19], beberapa contoh model untuk pemantauan kondisi meliputi :

- Nilai batas atau serangkaian batas pada nilai indikator kondisi yang menunjukkan kesalahan saat indikator melebihinya
- Distribusi probabilitas yang menggambarkan kemungkinan bahwa nilai tertentu dari indikator kondisi menunjukkan jenis masalah tertentu
- Pengklasifikasi yang membandingkan nilai indikator kondisi saat ini dengan nilai yang terkait dengan status kerusakan, dan mengembalikan kemungkinan adanya status kerusakan

II.1.3 Prediksi *Remaining Useful Life*

Model estimasi RUL menyediakan metode untuk melatih model menggunakan data historis dan menggunakannya untuk melakukan prediksi *Remaining Useful Lifetime*. Istilah *Lifetime* di sini mengacu pada masa pakai mesin yang didefinisikan dalam unit apapun yang digunakan untuk mengukur masa pakai sistem. Sisa waktu dapat berarti perubahan nilai dengan penggunaan, jarak yang ditempuh, jumlah siklus, atau kuantitas lain yang menggambarkan masa hidup [20]. Model estimasi RUL ini berguna jika memiliki data dan informasi historis seperti:

- Riwayat mesin yang mengalami kegagalan serupa dengan yang ingin dilakukan diagnosis
- Nilai batasan yang diketahui dari beberapa indikator kondisi yang menunjukkan adanya kegagalan
- Data tentang berapa banyak waktu atau berapa banyak penggunaan yang diperlukan untuk mesin serupa untuk mencapai kegagalan.



Gambar II.4 Diagram alir pendekatan pembelajaran mesin [20].

Similarity Models mendasarkan prediksi RUL dari mesin pada perilaku yang diketahui dalam data historis. Model tersebut membandingkan tren dalam data uji atau nilai indikator kondisi dengan informasi yang sama diekstraksi dari sistem serupa lainnya [20]. *Similarity Models* berguna ketika:

- Memiliki data *run-to-failure* dari sistem (komponen) serupa. Data *run-to-failure* adalah data yang dimulai selama operasi yang sehat dan berakhir

ketika mesin dalam keadaan mendekati kegagalan atau saat terjadi perawatan.

- Data *run-to-failure* menunjukkan perilaku degradasi yang serupa. Artinya, data berubah dalam beberapa karakteristik yang sama saat kinerja sistem menurun.

Degradation Models mengekstrapolasi perilaku masa lalu untuk memprediksi kondisi masa depan. Jenis perhitungan RUL ini cocok dengan model linier atau eksponensial untuk profil degradasi indikator kondisi. Kemudian menggunakan profil degradasi komponen uji untuk menghitung secara statistik waktu yang tersisa sampai indikator mencapai beberapa ambang batas yang ditentukan. Model-model ini paling berguna ketika ada nilai indikator kondisi yang diketahui menunjukkan kegagalan [20].

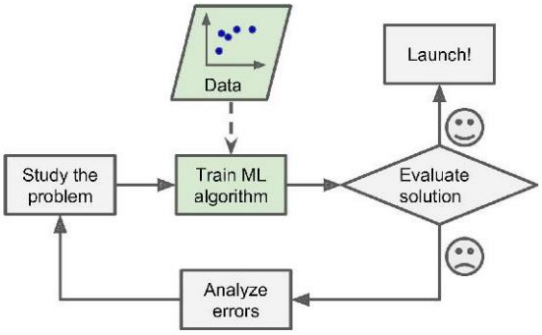
Survival Models adalah metode statistik yang digunakan untuk memodelkan data *time-to-event* [20]. Ini berguna ketika tidak memiliki riwayat *run-to-failure* yang lengkap, tetapi memiliki:

- Hanya data tentang masa pakai komponen serupa. Misalnya, mungkin diketahui berapa mil setiap mesin berjalan sebelum membutuhkan perawatan, atau berapa jam operasi setiap mesin berjalan sebelum gagal.
- Baik rentang hidup maupun beberapa data variabel lain (kovariat) yang berkorelasi dengan RUL. Kovariat, juga disebut variabel lingkungan atau variabel penjelas, terdiri dari informasi seperti penyedia komponen, rezim di mana komponen digunakan, atau batch manufaktur.

II.2 *Machine Learning*

Revolusi Industri 4.0 dikarakterisasi dengan lebih banyaknya internet seluler yang tersebar di mana-mana, juga tersedianya sensor yang semakin kecil dan akurat menjadi lebih murah dan terjangkau, serta munculnya kecerdasan buatan dan *machine learning* [21]. Dalam bahasa sehari-hari, istilah "kecerdasan buatan" sering digunakan untuk mendeskripsikan mesin (atau komputer) yang meniru fungsi "kognitif" yang diasosiasikan dengan pikiran manusia, seperti kemampuan untuk "belajar" dan "pemecahan masalah" [22]. Dengan demikian, pembelajaran mesin merupakan suatu bagian dari kecerdasan buatan. Pembelajaran mesin adalah

pemrograman komputer untuk mengoptimalkan kinerja suatu kriteria menggunakan data contoh atau pengalaman sebelumnya. Dengan memiliki model yang ditentukan berisi beberapa parameter. Maka dapat dilakukan pembelajaran mesin yaitu melaksanakan suatu program komputer untuk mengoptimalkan parameter dari model dengan menggunakan data pelatihan atau pengalaman masa lalu. Namun pembelajaran mesin bukan hanya itu saja, tetapi juga merupakan bagian kecerdasan buatan. Untuk menjadi cerdas, saat sistem berada di lingkungan berbeda, maka harus memiliki kemampuan untuk belajar sehingga dapat beradaptasi. Jika sistem dapat belajar dan beradaptasi dengan perubahan tersebut, maka perancang sistem tidak perlu meramalkan dan menulis program untuk semua kemungkinan situasi [23].



Gambar II.4 Diagram alir pendekatan pembelajaran mesin [24].

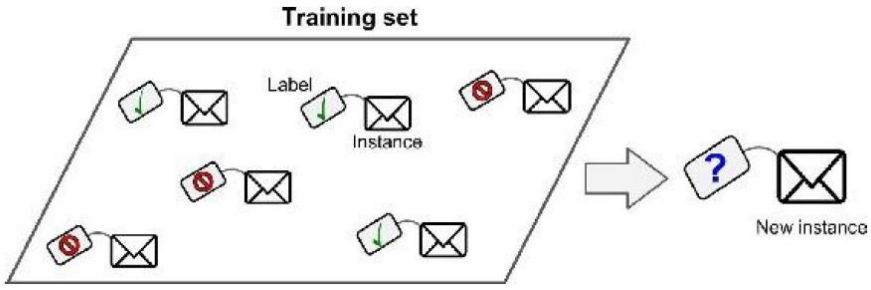
Berdasarkan diagram alir tersebut, pembelajaran mesin adalah suatu siklus yang terus berulang hingga dapat menghasilkan nilai yang memuaskan untuk dapat diluncurkan. Data yang akan digunakan perlu diatur formatnya menyesuaikan algoritma yang digunakan. Hal ini membuat algoritma ML menjadi hal utama yang perlu ditentukan dalam membuat program. Algoritma tersebut dapat diklasifikasi berdasarkan beberapa kriteria yaitu keterlibatan pengawasan manusia, kemampuan sistem dalam pembelajaran aliran data bertahap, dan proses prediksi hasil. Penggunaan kriteria tersebut tidak eksklusif dan dapat digunakan secara bersamaan [24]. Pada bagian berikut ini akan dijelaskan mengenai beberapa jenis algoritma yang diterapkan dalam penelitian Tesis ini.

II.2.1 Keterlibatan Pengawasan Manusia

Berdasarkan kriteria keterlibatan pengawasan manusia dalam proses pembelajaran mesin, terbagi menjadi kategori sebagai berikut [24] :

1. Diawasi (*Supervised*)

   Pada pembelajaran dengan metode ini, data pelatihan akan diberikan informasi mengenai solusi yang diinginkan, disebut dengan istilah label. Label inilah yang menjadi keterlibatan manusia dalam sistem pembelajaran mesin, karena ditentukan oleh manusia sebelum diolah oleh algoritma.

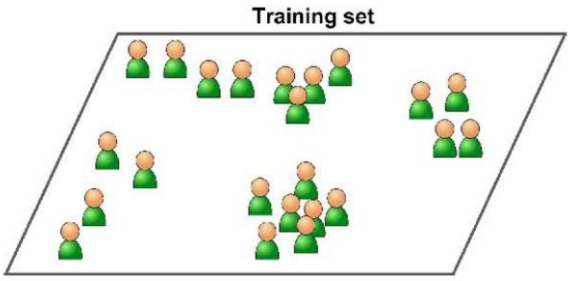


Gambar II.5 Set data pelatihan dengan label

   Sistem yang menerapkan metode ini akan melakukan pembelajaran dengan menganalisis set data pelatihan dan mencari pola yang dapat menentukan hasil yang sesuai. Jika diberikan suatu data uji baru tanpa label, maka sistem akan membandingkan nya dengan pola dari dataset untuk menentukan hasil.

2. Tanpa Pengawasan (*Unsupervised*)

   Pada pembelajaran dengan metode ini, data pelatihan tidak diberikan informasi label sehingga sistem belajar tanpa keterlibatan manusia dalam mengolah set data pelatihan.



Gambar II.6 Set data pelatihan tanpa label [24]

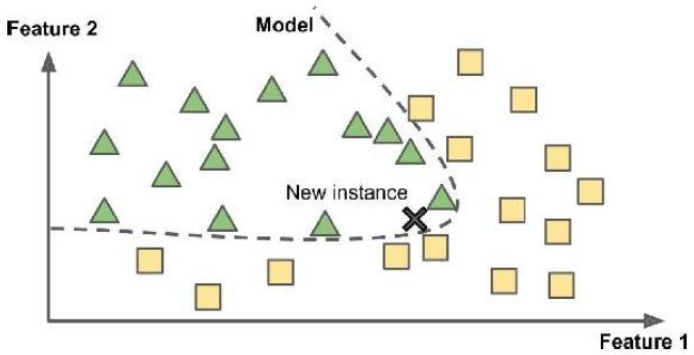
   Sistem yang menerapkan metode ini maka algoritma yang digunakan langsung menganalisis hubungan antar elemen yang ada dari data pelatihan dan menentukan sendiri hasilnya. Jika diberikan suatu data uji baru tanpa

label, maka sistem akan membandingkan karakteristik nya lalu menentukan hasil berdasarkan kedekatan nya dengan kategori yang didapat dari data pelatihan.

II.2.2 Proses Prediksi Hasil

Berdasarkan metode yang digunakan dalam proses prediksi hasil, sistem berbasis pembelajaran mesin dibagi menjadi dua kategori sebagai berikut [24] :

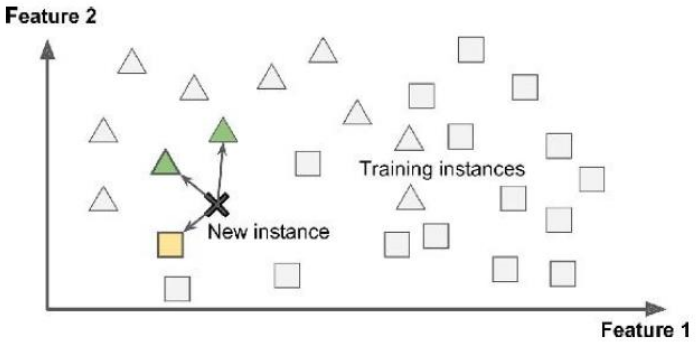
1. *Model-Based*



Gambar II.8 Pengelompokan hasil dengan pembelajaran *Model-based*

Metode ini bekerja dengan melakukan penarikan kesimpulan berdasarkan model hasil pembelajaran dari set data pelatihan. Saat melakukan proses pembelajaran, sistem akan mengolah set data pelatihan terlebih dulu untuk menemukan hubungan antar elemen dan menyimpulkan model yang paling cocok untuk dapat menjadi representasi set data. Ketika sistem diterapkan maka prediksi hasil dari data terbaru akan sesuai berdasarkan model yang telah ditentukan.

2. *Instance-Based*



Gambar II.9 Pengelompokan hasil dengan pembelajaran *Instance-based*

Metode ini bekerja dengan melakukan penarikan kesimpulan berdasarkan karakteristik yang berdekatan antar data baru dengan data yang sudah ada. Proses prediksi hasil akan mengambil kesimpulan bahwa data baru akan dikategorikan berdasarkan data lama yang paling dekat karakteristiknya.

II.3 Pembahasan Penelitian Terkait

Salah satu studi yang paling mendekati dengan topik ini dilakukan oleh Antony Barnes pada tahun 2014 [25]. Membahas mengenai meningkatkan performa lokomotif dengan melakukan *Condition Based Maintenance* (CBM). Namun dalam studi tersebut belum mencapai *predictive maintenance* maupun menggunakan *machine learning*. Metode yang dijelaskan sudah menggunakan berbagai perangkat sensor mengambil data kondisi lokomotif, namun masih dianalisa secara manual. Penelitian tersebut ditujukan untuk dapat memberikan pendeteksian dan peringatan dini atas kesalahan yang berkaitan dengan komponen yang memiliki kategori masalah berikut:

- Kesalahan Bantalan Elemen Bergulir (*Rolling Element Bearing faults*)
- Kerusakan roda gigi *(Gear mesh faults)*
- Masalah daya dan kelistrikan (*Load and electrical problems)*
- Masalah pelumasan (*Lubrication issues)*
- Masalah pembakaran (*Combustion Problems)*

Dari kesimpulan penelitian ini menyatakan bahwa metode dan strategi program perawatan CBM ini telah beroperasi selama 5 tahun dan dianggap sukses oleh pihak operator kereta api. Ini membuktikan bahwa metode ini layak untuk dapat dikembangkan lebih lanjut menjadi sistem perawatan prediktif yang akan dilakukan pada tesis ini.

Studi lainnya yang dilakukan di Singapura pada tahun 2018. [26] Berusaha menggunakan ANN sebagai cara untuk melakukan perawatan prediktif pada sistem kereta, namun belum sampai tahap diujinya model yang ditunjukkan. Beberapa parameter yang dipilih untuk menjadi basis prediksi adalah *Oil Level*, *Screw compressor input current and input voltage*, *Lubricant temperature*, dan *Output compressed air pressure*. Penelitian baru memberikan proposal untuk memasang sensor yang mengumpulkan data tersebut, dan kesimpulan hanya mengatakan untuk

melakukan finalisasi pemasangan sensor. Dari uraian tersebut, keberhasilan dari implementasi PdM masih belum diberikan.

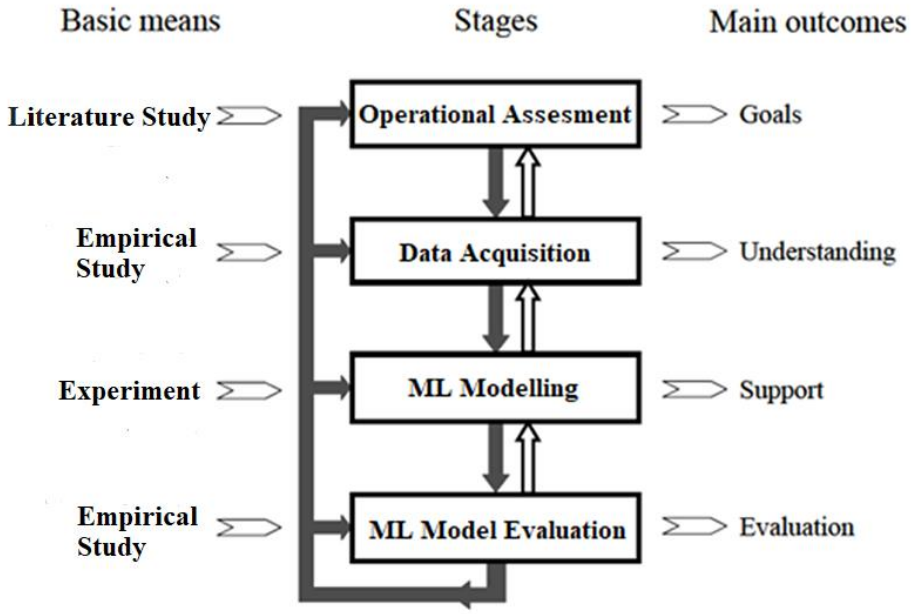
Hasil penelitian terbaru pada tahun 2020 oleh Wang, et al. [27] berusaha menggabungkan sistem perawatan berdasarkan *data-driven* dengan *model-based* memanfaatkan LSTM-RNN untuk perangkat daya pada kereta cepat. Penelitian dilakukan hingga pengujian model yang membuktikan bahwa pendekatan yang diajukan memungkinkan. Meski demikian, metode penelitian yang disampaikan masih menggunakan data yang berdasarkan simulasi Monte-Carlo saja, walaupun menggunakan parameter berdasarkan peralatan *Traction Power Supply System* (TPSS) yang ada pada kereta cepat.

Penelitian lain dilakukan di Indonesia, pada tahun 2020 oleh Nafisah. [28] Mencoba memprediksi kejadian patah rel menggunakan beberapa algoritma untuk membuat model pembelajaran mesin, algoritma yang dicoba adalah Regresi Logistik, SVM, Decision Tree, Random Forest, dan KNN. Hasil terbaik dicapai saat menggunakan Random Forest dengan akurasi 67%. Walaupun memiliki perbedaan objek prediksi dengan tesis ini, namun metodologi nya dapat dipertimbangkan.

Penelitian [29] bertujuan untuk mengimplementasikan PdM untuk kendaraan *rolling stock* menggunakan pendekatan berbasis model. Pendekatan dilakukan dengan membuat model matematika untuk ML yang diperoleh dari skematik kendaraan pengujian PV7 EVO. Meskipun masih hasil penelitian awal, hasilnya dapat mengidentifikasi kondisi di mana perawatan harus dilakukan dengan memperhatikan perpindahan as roda dengan jarak yang ditempuh.

BAB III DESAIN PENELITIAN

Pada bagian ini akan dijelaskan detail mengenai tahapan proses penelitian yang direncanakan untuk tesis yang dilakukan. Penelitian berbasis metodologi penelitian desain (*Design Research Methodology*) atau disebut DRM [30] dengan penyesuaian berdasarkan tahapan implementasi PdM [12], [17].



Gambar III.1 Prosedur sistematis yang akan digunakan

Inti pada metodologi penelitian desain adalah melakukan proses iterasi yang terdiri dari berbagai tahap [30]. Tahap *Operational Assessment*, keluaran nya adalah untuk membuat beberapa tujuan yang ingin dicapai pada tahap iterasi kali ini. Kemudian dilanjutkan ke tahap *Data Acquisition*, dengan hasil mendapatkan pemahaman tentang masalah yang diteliti dan mencari faktor yang mempengaruhi hasil penelitian. Tahap ketiga adalah *ML Modelling*, bertujuan untuk mendapatkan data atau informasi yang mampu mendukung hasil pada proses sebelumnya. Terakhir dilakukan *ML Model Evaluation*, bertujuan untuk mengevaluasi dan menginvestigasi dampak hasil yang didapat sebelumnya terhadap tujuan kita. Proses ini akan terus berulang, berlanjut sampai pemahaman yang memadai atau menemukan solusi untuk masalah penelitian tercapai.

III.1 *Operational Assessment*

Menyerupai tahap *Research Clarification* (RC) pada DRM, akan dilakukan pencarian untuk menemukan beberapa bukti atau setidaknya indikasi yang mendukung asumsi penelitian agar dapat dirumuskan secara realistis dan tujuan penelitian yang bermanfaat. Berdasarkan temuan tersebut, gambaran awal tentang situasi yang ada dikembangkan, serta deskripsi situasi yang diinginkan, kemudian membuat asumsi yang mendasari setiap deskripsi situasi ideal. Kemudian merumuskan beberapa kriteria atau parameter yang bisa dijadikan ukuran terhadap hasil penelitian [30]. Pada tahap ini, teknik yang digunakan adalah studi literatur. Berbagai macam informasi yang dibutuhkan dalam penelitian akan dicari dari sumber yang terpercaya. Diantaranya pencarian berbagai macam spesifikasi dari perangkat lunak yang cocok untuk dapat digunakan pada penelitian ini, mempelajari berbagai jenis algoritma pembelajaran mesin serta aplikasinya pada bidang PdM, dan yang paling utama adalah melakukan kajian terhadap kegiatan perawatan pada PT. KAI melalui dokumen yang disediakan.

Keluaran dari tahap ini secara umum adalah berbagai hal yang telah dijelaskan pada Bab I Pendahuluan, terutama pernyataan tujuan penelitian dan latar belakang perlunya dilakukan penelitian. Secara khusus tahap ini akan menentukan tujuan yang perlu dicapai saat proses iterasi tertentu, adapun tujuan yang telah dapat dirumuskan saat ini adalah sebagai berikut:

Iterasi	Tujuan
0	• Mendefinisikan tujuan penelitian • Menentukan kontribusi penelitian • Melakukan kajian perawatan PT. KAI • Mempelajari data perawatan yang tersedia pada KAI • Menentukan target Akurasi program • Menentukan target batasan *Runtime* program • Membuat program ML yang berhasil melakukan prediksi kondisi sarana kereta

1
- Mengoptimalkan kinerja model
- Menguji model ML dengan data tanpa label

2
- Menentukan opsi terbaik untuk sistem PdM
- Menentukan Algoritma terbaik

III.2 *Data Acquisition*

Mengikuti tahap Deskriptif Studi I (DS-I) pada DRM, setelah mendapatkan gambaran yang jelas mengenai tujuan dan fokus, perlu meninjau literatur untuk faktor lain yang lebih mempengaruhi untuk menguraikan deskripsi awal dari situasi yang ada. Tujuannya adalah untuk membuat deskripsi cukup rinci untuk menentukan faktor mana yang harus ditangani untuk meningkatkan agar desain menjadi efektif dan efisien. Namun, jika tidak menemukan bukti yang cukup dalam literatur untuk menentukan dengan jelas faktor-faktor penting ini, dapat mengamati secara langsung pada lokasi untuk mendapatkan pemahaman yang lebih baik tentang situasi yang ada, sebelum melanjutkan ke tahap berikutnya [30]. Cara yang digunakan adalah studi empiris dengan menyelidiki topik penelitian serupa, yang telah dijelaskan pada bagian sebelumnya, dan juga mengumpulkan sumber data apa pun yang memiliki kemungkinan menjadi sumber untuk melatih ML, seperti: hasil inspeksi rutin dari sarana kereta pembangkit dan lokomotif, rekap pemantauan kondisi di lokomotif, serta perangkat koleksi data yang dipasang pada generator kereta pembangkit.

III.2.1 *Checksheet* perawatan lokomotif

Dari diskusi yang dilakukan dengan pihak PT.KAI bagian sarana, sampai saat ini untuk pemeriksaan kondisi lokomotif masih dilakukan secara manual dengan acuan *checksheet* seperti yang ditampilkan pada gambar dibawah. Terdapat periode yang telah ditentukan untuk dilakukan pemeriksaan rutin, untuk lokomotif ditandakan dengan kode PH (Pemeriksaan Harian), P1 (Pemeriksaan 1 Bulanan), P3 (Pemeriksaan 3 Bulanan), P6 (Pemeriksaan 6 Bulanan), P12 (Pemeriksaan 12 Bulanan). Berdasarkan periode tersebut, penelitian dapat bermanfaat untuk mengurangi keperluan PH dan P1 pada item komponen tertentu. Ada 4 kategori utama dari item yang perlu diperiksa pada PH yaitu Pengereman, Perangkai, Keselamatan, dan Kelistrikan. Sementara itu ada 12 kategori utama item untuk

diperiksa pada P1 yaitu Pengereman, Perangkai, Keselamatan, Kelistrikan, Rangka Dasar, Body, Bogie, Kabin Masinis, Penerus Daya, Penggerak, Pengendali, Penghalau Rintangan. Terlihat dari periode PH dan P1 bahwa ada kesamaan pada 4 kategori, yaitu Pengereman, Perangkai, Keselamatan, dan Kelistrikan. Untuk membatasi cakupan penelitian, dipilih Kelistrikan sebagai fokus kategori untuk diterapkan PdM.

LEMBAR PEMERIKSAAN BERKALA LOKOMOTIF BB203/CC201/CC203/CC204/ CC206	PH	UPT Depo Lokomotif: / Tgl Perawatan : / No. Seri Lokomotif : / No. MO : / KM Tempuh :	No Dokumen : / Revisi ke : / Tgl dikeluarkan : / Halaman :

NO	ITEM YANG DIPERIKSA	STANDAR	HASIL PEMERIKSAAN		KETERANGAN
			OK	NOT OK	
I. PENGEREMAN					
1	Tekanan automatic brake / Rem rangkaian	70 Psi			Memastikan tekanan pengereman untuk Automatic Brake
2	Tekanan independent brake	45 -50 Psi			Memastikan tekanan pengereman
3	Blok rem	Lengkap			Baik, aus merata
II. PERANGKAI					
1	Alat tolak tarik	Baik			Tiada indikasi cacat, retak
2	Slang air brake dan stop cock	Baik, tidak bocor			Slang baik tiada indikasi bocor, kepala baik, stop cock berfungsi
3	Rantai pengaman	Lengkap 2 buah			Lengkap (2 di masing-masing ujung) tiada indikasi cacat

Gambar III.2 Contoh lembar pemeriksaan harian lokomotif

Permasalahan dari penggunaan *checksheet* ini adalah item yang diperiksa kebanyakan berupa kuantitatif, dengan standar yang ditetapkan hanya berupa berfungsi/tidak sehingga sulit jika akan melakukan prediksi untuk *Remaining Useful Lifetime* (RUL) menggunakan regresi berdasarkan data tersebut. Selain itu bentuk yang masih dari kertas menyebabkan perlunya proses mengubah ke bentuk digital jika ingin digunakan, menambah biaya waktu dan tenaga manusia. Meski demikian, informasi mengenai item penting yang harus diperiksa dapat digunakan untuk menentukan komponen terpilih untuk dilakukan prediksi. Proses pemeriksaan pada lokomotif perlu dilakukan perubahan menjadi berbasis digital sebelum data nya dapat digunakan untuk dimanfaatkan.

III.2.2 *Checksheet* pemeriksaan genset

Dari diskusi yang dilakukan dengan pihak PT.KAI bagian sarana, sampai saat ini untuk pemeriksaan kondisi genset juga masih dilakukan secara manual dengan acuan *checksheet* seperti yang ditampilkan pada gambar. Terdapat periode yang telah ditentukan untuk dilakukan pemeriksaan rutin, untuk genset ditandakan dengan kode P100 (Pemeriksaan setiap 100 Jam), P300 (Pemeriksaan setiap 300

Jam), P600 (Pemeriksaan setiap 600 Jam), P1200 (Pemeriksaan setiap 1200 Jam), P2400 (Pemeriksaan setiap 2400 Jam), P4800 (Pemeriksaan setiap 4800 Jam). Berdasarkan periode tersebut, penelitian dapat bermanfaat untuk mengurangi keperluan P100 dan P300 sebagai pemeriksaan yang paling sering dilakukan. Ada 10 kategori utama dari item yang perlu diperiksa pada P100 maupun P300 yaitu Sistem Bahan Bakar, Sistem Pelumasan, Sistem Pendinginan, Filter Udara, V-Belt, Alternator Pengisian, Baterai, Motor Diesel, Instalasi dan Panel Kontrol, Ruang Genset. Dari beberapa kategori tersebut, pada penelitian ini akan difokuskan untuk monitor kondisi di Sistem Pelumasan, Sistem Pendinginan, dan Baterai.

LEMBAR PEMERIKSAAN GENSET 100 JAM	P 100	NO. MO : DEPO PERAWATAN : TGL PEMERIKSAAN : NO. SERIE KERETA : JAM KERJA ENGINE : MO/GO TERAKHIR :		NO DOKUMEN : 02/RRC/LPG P100/2021 REVISI KE : 2 (dua) TGL. REVISI : 21 Januari 2021 HALAMAN :
NO	ITEM YANG DIPERIKSA	STANDAR	HASIL PENGUKURAN	KETERANGAN
I	SISTEM BAHAN BAKAR			
	1. Fuel separator filter	Baik dan bersih		
	2. Pompa listrik dan pompa manual bahan bakar	Baik dan berfungsi		
	3. Tangki bahan bakar atas (sekunder) dan bawah (primer)	Bersih dan tidak bocor		
II	SISTEM PELUMASAN	Baik dan tidak bocor		
III	SISTEM PENDINGINAN			
	1. Elemen radiator	Baik dan tidak bocor		
	2. Air radiator	Cukup		

Gambar III.3 Contoh lembar pemeriksaan genset setiap 100 jam

Permasalahan dari penggunaan *checksheet* ini sama dengan sebelumnya, yaitu item yang diperiksa kebanyakan berupa kuantitatif dengan standar yang ditetapkan hanya berupa berfungsi/tidak sehingga sulit jika akan melakukan prediksi untuk *Remaining Useful Lifetime* (RUL) menggunakan regresi berdasarkan data tersebut. Selain itu bentuk yang masih dari kertas menyebabkan perlunya proses mengubah ke bentuk digital jika ingin digunakan, menambah biaya waktu dan tenaga manusia. Meski demikian, informasi mengenai item penting yang harus diperiksa dapat digunakan untuk menentukan komponen terpilih untuk dilakukan prediksi.

III.2.3 *Snap Log* lokomotif

Sumber data lainnya yang didapat dari PT.KAI berupa catatan kejadian yang tersimpan pada lokomotif. Tidak dijelaskan detail bagaimana kondisi yang menjadi *trigger* setiap kejadian yang tercatat, sehingga hanya bisa diperkirakan saja berdasarkan deskripsi dan informasi sensor yang tersedia. Hanya beberapa sensor

saja yang terlihat variasi nilai untuk dapat dianalisis, diantaranya adalah temperature air, temperature oli, kecepatan putar mesin, EAFP, dan PAP Pressure. Dari beberapa *trigger* yang tercatat, hanya deskripsi "Engine Lube Oil Outlet Temp Sensor out of Range Low" yang terlihat hubungan antara sensor dan kejadian. Ketersediaan data hanya untuk periode 25-07-2020 pukul 22:10:49 hingga 04-10-2020 pukul 11:23:35 atau sekitar 72 Hari, sepertinya dikarenakan keterbatasan memori untuk menyimpan data sehingga terjadi *overwrite* untuk periode sebelumnya. Untuk dapat melakukan transformasi digital, maka PT.KAI perlu mulai mengarsipkan dan memanfaatkan data seperti ini.

"Indonesian Time"	"Trigger"	"Trigger Description"	"Engine Speed"	"Loco Spd"	"Water Inlet Temp"	"Water Outlet Temp"	"Oil Inlet Temp"	"Oil Out Temp"
	" "	" "	RPM	mph	°F	°F	°F	°F
10-04-2020 19:19:41	34-0006	Fuel Tank Monitor (FTM) Sensing Mechanism Bad	0	0	152.633	163.289	158.714	153.461
10-04-2020 19:19:29	07-0011	Supervisory Maintenance Level Entered	0	0	152.634	163.49	158.72	153.461
10-04-2020 19:19:14	11-3404	PT-1B Receive Error	0	0	152.767	163.49	158.744	153.509
10-04-2020 19:19:12	51-0000	Incident Undefined	0	0	152.797	163.49	158.747	153.506
10-04-2020 19:17:33	08-3003	Console Selector Switch Bad	0	0	153.261	163.916	159.127	154.085
10-04-2020 19:17:33	51-9000	Wait for Control Execution	0	0	153.265	163.923	159.114	154.085
10-04-2020 19:17:18	11-6700	No Communications With BCCA At Startup	-1	0	150	0	0	160
10-04-2020 18:23:39	08-3013	Dual Console Digital Feedbacks Quality Bad	0	0	150	150	160	160
10-04-2020 18:23:37	11-2601	Loss Of Communications With ECU While Running	0	0	150	150	160	160
10-04-2020 18:23:35	01-6085	Loss of Power to the ECU	332.235	0	177.65	178.886	175.863	178.205

Gambar III.4 Contoh log kejadian yang tersimpan pada lokomotif

Dalam konteks penelitian ini, tersedianya data ini menandakan bahwa lokomotif sudah memiliki mekanisme sensor dan kontrol komputer untuk melakukan monitor kondisi bahkan *Fault Detection*, sehingga dapat dimanfaatkan untuk pemeriksaan. Misalnya dapat digunakan untuk mengetahui komponen mana yang perlu diperiksa saat terjadi *trigger* tertentu apabila teknisi memiliki spesifikasi lokomotif dan panduan dari pabrikan pembuat. Namun hingga saat ini, data *snap log* ini tidak diarsipkan dan disimpan secara rutin, sehingga jika ingin digunakan pada penelitian ini sebagai data latih pembelajaran mesin maka dibutuhkan untuk mengambil secara manual terlebih dulu dari lokomotif. Karena keterbatasan waktu penelitian dan kondisi pembatasan aktivitas akibat COVID-19, pengumpulan data ini tidak dilakukan lebih lanjut. Dengan demikian, data ini dianggap tidak cocok digunakan sebagai dataset untuk ML.

III.2.4 *Data collection* kereta pembangkit

Kereta pembangkit merupakan bagian dari rangkaian kereta yang biasa digabungkan dengan kereta penumpang, sebagai penyedia listrik yang didistribusikan ke berbagai lokasi di kereta seperti TV, lampu, serta stop kontak untuk fasilitas penumpang. Umumnya kereta pembangkit dipasang pada kereta antar-kota, dan bukan merupakan bagian untuk penggerak rangkaian kereta.

Tabel III.1 Daftar kereta pembangkit yang digunakan datanya

No	Kereta Pembangkit		DIPO
	Jenis	Nomor	
1	MP3 (Makan dan Pembangkit Ekonomi AC)	01605	PWT (Purwokerto)
2	MP3 (Makan dan Pembangkit Ekonomi AC)	01703	JAKK (Jakarta)
3	P (Pembangkit)	01601	CN (Cirebon)
4	P (Pembangkit)	01801	BD (Bandung)
5	P (Pembangkit)	01808	SDT (Sidotopo)
6	P (Pembangkit)	01810	SLO (Solo)
7	P (Pembangkit)	01811	JAKK (Jakarta)
8	P (Pembangkit)	01812	JAKK (Jakarta)
9	P (Pembangkit)	01818	SLO (Solo)
10	P (Pembangkit)	01820	SLO (Solo)
11	P (Pembangkit)	01821	SLO (Solo)
12	P (Pembangkit)	01822	BD (Bandung)
13	P (Pembangkit)	01823	BW (Banyuwangi)
14	P (Pembangkit)	01825	BW (Banyuwangi)
15	P (Pembangkit)	01903	JAKK (Jakarta)
16	P (Pembangkit)	01908	SBI (Surabaya)
17	P (Pembangkit)	01914	SBI (Surabaya)

Sumber data diperoleh dari beberapa kereta pembangkit yang dipasangkan perangkat untuk mengambil informasi sensor secara periodik dan disimpan pada *cloud platform*. Ada total 45 unit kereta pembangkit yang telah terpasang perangkat ini, dan diberikan data 17 unit kereta yang tertulis pada Tabel III.1 untuk digunakan pada penelitian. Sebagian besar kereta yang sudah terpasang perangkat, dirawat

oleh Dipo yang berada di Pulau Jawa sehingga untuk penelitian ini dibatasi kepada kereta tersebut.

№	Grouping	(Deepsea Control Mode)	(L1)	(L2)	(L3)	(Frequency)	(kVA Total)	(kVA_L1)	(kVA_L2)	(kVA_L3)	(KVAr)	(Oil Pressure)	(Coolant Temperature)	(Charger Alternator)	(Power Factor Average)	(Power Factor_L1)	(Power Factor_L2)	(Power Factor_L3)	(L1_N)	(L2_N)	(L3_N)	(Source Ext Voltage)	(ECU Temperature)	(RPM)
1.1	2020-04-01 04:02:06	2.00	26.00	26.00	26.00	50.00	17.38	5.88	5.74	5.70	4294964.46	636.00	81.00	27.90	0.89	0.86	0.89	0.92	222.70	222.80	222.80	27.38	36.00	1500.00
1.2	2020-04-01 05:00:15	2.00	38.00	34.00	31.00	50.10	25.40	8.42	7.76	6.82	2.40	632.00	82.00	27.90	0.93	0.92	0.94	0.92	222.90	222.50	222.80	27.22	38.00	1500.00
1.3	2020-04-01 06:00:15	2.00	87.00	84.00	82.00	50.00	58.10	19.42	18.88	18.40	24.16	624.00	82.00	27.90	0.89	0.89	0.90	0.87	222.60	222.60	222.30	27.36	40.00	1499.00
1.4	2020-04-01 07:00:15	2.00	103.00	101.00	99.00	50.00	68.77	22.94	22.52	22.08	29.04	620.00	82.00	27.90	0.89	0.89	0.90	0.88	222.50	222.70	222.00	27.33	43.00	1499.00
1.5	2020-04-01 08:00:16	2.00	93.00	89.00	88.00	50.00	62.03	20.62	19.74	19.56	24.32	620.00	82.00	27.90	0.90	0.90	0.91	0.89	221.70	222.20	221.80	27.36	43.00	1500.00

Gambar III.5 Data sensor untuk kereta pembangkit

Dari data yang didapatkan tersebut, sebagai contoh pada kereta jenis P-01801 memiliki berbagai informasi yang dapat digunakan. Diantaranya adalah pemantauan sensor dengan entri data pertama dimulai pada 2020-04-01 pukul 04:02:06 hingga berakhir pada 2020-10-31 pukul 23:00:14 atau sekitar 213 hari dengan total entri sebanyak 1954 baris. Data ini akan bertambah banyak seiring dengan meningkatnya waktu operasi perangkat pada sarana yang terpasang. Berikut adalah penjelasan Parameter yang dikumpulkan oleh perangkat ini yaitu:

1) ***Grouping* :** Tanggal dan waktu saat data dikumpulkan (datetime string)
2) ***Deepsea Control Mode* :** Mode operasi perangkat pengumpul data (integer)
3) **L1, L2, dan L3 :** Nilai arus pada generator di titik pengukuran (float)
4) ***Frequency* :** Nilai frekuensi sumber AC (float)
5) **kVA Total :** Jumlah Daya dari titik pengukuran di Generator (float)
6) **kVA_L1, kVA_L2,** dan **kVA_L3 :** Besar Daya di titik pengukuran Generator (float)
7) **kVAr :** Besar Daya reaktif di Generator (float)
8) ***Oil Pressure* :** Nilai tekanan oli pelumas dalam kPa (integer)
9) ***Coolant Temperature* :** Nilai Pendingin mesin dalam Celsius (integer)
10) ***Charger Alternator* :** Output Daya charger baterai dalam kVA (float)

11) ***Power Factor Average*** **:** Nilai rata-rata rasio daya kerja dari titik pengukuran dalam kW (float)
12) ***Power Factor_L1, Power Factor_L2,*** dan ***Power Factor_L3*** **:** Nilai rasio daya kerja di titik pengukuran dalam kW (float)
13) **L1_N, L2_N,** dan **L3_N :** Nilai tegangan di titik pengukuran dalam V (float)
14) ***Source Ext Voltage*** **:** Nilai tegangan sumber eksternal dari PLN (float)
15) ***ECU Temperature*** **:** Nilai temperatur unit kontrol elektronik dalam Celsius (integer)
16) **RPM :** Nilai kecepatan rotasi mesin per menit (integer)

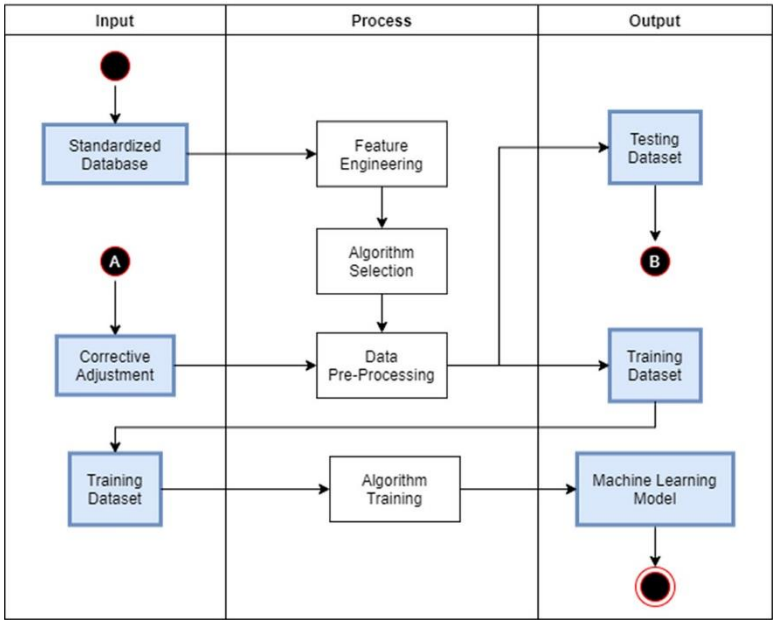
Selain itu juga ada pencatatan kondisi awal dan kondisi akhir saat kereta beroperasi, yang hanya mencatat waktu, lokasi, kecepatan, *ECU temperature* dan *coolant temperature*. Untuk data ini dilakukan monitor sejak 2020-03-07 pukul 16:40:11 hingga 2021-03-08 pukul 09:00:26 atau sekitar 366 hari dengan total sebanyak 643 baris entri data. Untuk penelitian ini, lebih diutamakan pada jenis data pemantauan sensor yang diberikan pada gambar karena informasinya yang lebih lengkap.

Permasalahan dari data ini adalah penggunaan perangkat *data collection* masih dalam tahap percobaan, sehingga data sensor terkadang tidak terbaca atau memiliki nilai yang jauh berbeda dari seharusnya. Selain itu tidak ada label yang diketahui untuk menjadi acuan kondisi dari kereta pembangkit, sehingga data perlu dianalisis lebih lanjut dan membuat asumsi kondisi sebagai label.

Keunggulan data ini jika dibandingkan dengan data *snap log* lokomotif pada bagian sebelumnya, yaitu memiliki sensor yang lebih lengkap. Ada beberapa parameter yang juga tersedia disini seperti Coolant Temperature (Water Temperature pada lokomotif) dan RPM, sehingga kedua parameter tersebut dapat menjadi pertimbangan untuk dipilih menjadi fitur. Ketersediaan dalam bentuk digital, dan semakin bertambahnya arsip data seiring waktu menjadi pertimbangan utama data ini dipilih untuk menjadi basis yang akan digunakan pada perancangan PdM.

III.3 *ML Modelling*

Setara dengan tahap Preskriptif Studi (PS) pada DRM, bertujuan untuk mengoreksi dan menguraikan awal deskripsi situasi yang diinginkan dengan menggunakan peningkatan pemahaman tentang situasi yang ada saat ini. Dibuat berbagai kemungkinan skenario dengan memvariasikan faktor yang ditargetkan untuk mencapai hasil yang dapat mendukung situasi yang diinginkan [30]. Tahap ini mungkin diulangi sampai dapat mencapai hasil yang diinginkan, diharapkan hasilnya mampu lebih baik jika dibandingkan dari penelitian sejenis lainnya.



Gambar III.6 Alur kerja proses desain model ML

1) *Standardized Database*: Sebelum proses dimulai, data mentah yang dimiliki dengan format excel (.xlsx) berisi beberapa *sheet* dengan konten yang berbeda, sehingga perlu mengambil informasi yang diperlukan saja. Data yang dipilih ini digabungkan menjadi satu file sebagai *dataset* utama dan diubah menjadi format *commas separated values* (.csv). Dilakukan pemeriksaan nilai *hashing* memanfaatkan hashlib untuk memastikan integritas file sebelum di upload ke github untuk dapat diakses lebih mudah. Kemudian dalam program akan dibandingkan nilai *hashing* dari file yang di upload untuk memastikan tidak ada perubahan yang terjadi, dan apabila dilakukan revisi dapat dipastikan bahwa menggunakan dataset yang benar.

2) *Feature Engineering*: Langkah ini bertujuan untuk menentukan parameter yang digunakan dalam PdM. Dari 16 parameter berbeda pada tiap kolom,

belum tentu semua cocok digunakan dalam ML. Selain itu, data mentah yang didapatkan tidak memiliki ketentuan kondisi untuk bisa dinyatakan normal atau tidak, sehingga perlu diberikan label untuk mengelompokkannya. Ada dua alternatif pendekatan yang dapat dilakukan: menggunakan algoritma *clustering* atau menggunakan pendekatan statistik. Data akan dibagi menjadi tiga kategori: Outlier, Normal, dan Maintenance. Selain itu juga bisa ditambahkan label secara manual untuk setiap parameter jika diketahui, seperti memberi label kelipatan 100 jam kerja sebagai perawatan berdasarkan *checksheet*.

3) *Algorithm Selection*: Pada tahap ini akan dipilih algoritma yang akan digunakan dalam membuat model ML. Algoritma disediakan oleh *library* pemrograman yang secara public tersedia untuk digunakan, seperti TensorFlow atau Scikit-learn [24]. Pilihan algoritma yang akan diuji kemungkinan adalah Logistic Regression (LR), k-NN, RF, Decision Tree, dan SVM untuk mengklasifikasikan data. Selain itu algoritma clustering seperti Density-Based Spatial Clustering of Applications with Noise (DBSCAN) dan Kmeans juga dipertimbangkan untuk memberi label pada dataset.

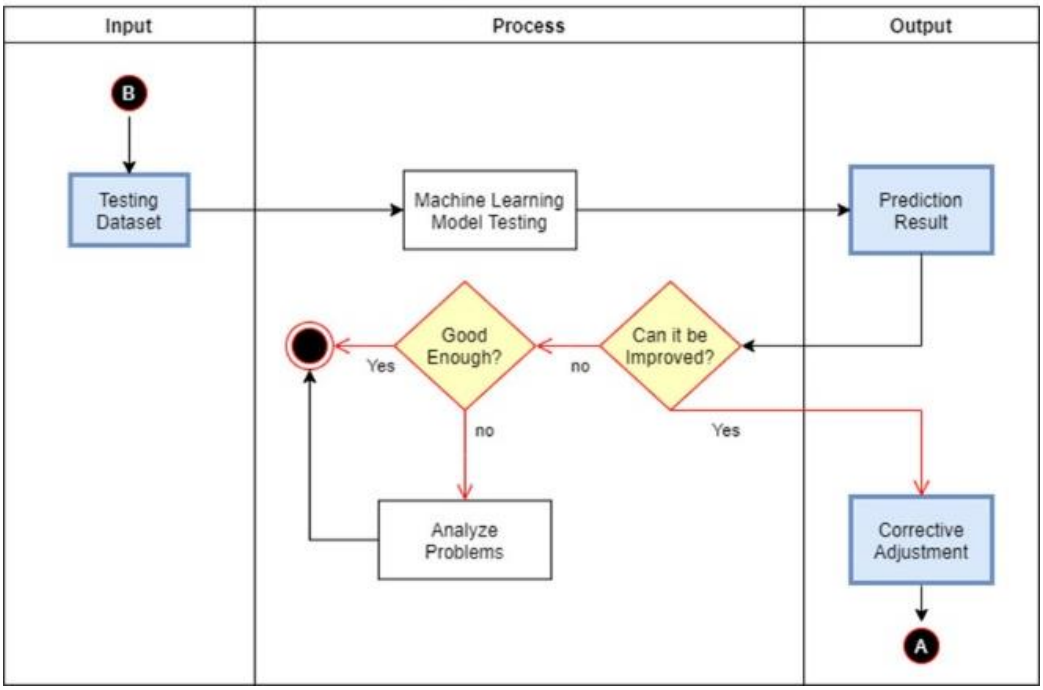
4) *Data Pre-Processing*: Rincian langkah ini akan berbeda tergantung pada algoritma yang digunakan dan akan menyesuaikan dengan situasi. Karena kumpulan data saat ini mungkin perlu informasi yang akan ditambahkan, diubah, atau dihapus. Setelah itu, data akan dibagi menjadi dua: data berlabel digunakan sebagai *training dataset* yang akan menjadi masukan untuk langkah selanjutnya. Sementara itu, ada dua jenis dataset pengujian yaitu: *validation dataset*, yang labelnya diketahui namun disembunyikan serta *testing dataset* yang labelnya tidak diberikan. Nilai rasio pembagian dataset dapat bervariasi antara setiap iterasi.

5) *Algorithm Training*: Pada langkah ini, *training dataset* akan diproses oleh algoritma yang dipilih, menghasilkan model ML. Untuk mengetahui performanya, akan dilakukan *cross validation* menggunakan stratified k-

fold. Setelah model ML dibuat, tahap pemodelan ML ini selesai, dan lanjut ke tahap berikutnya

III.4 *ML Model Evaluation*

Menyerupai tahap Deskriptif Studi II (DS-II) pada DRM, yaitu untuk menyelidiki dampak dari informasi dukungan dan kemampuan desain untuk memenuhi tujuan. Dilakukan dua studi empiris untuk mendapatkan pemahaman, studi pertama dilakukan untuk mengevaluasi penerapan informasi yang mendukung untuk membuat desain menjadi lebih baik. Studi kedua digunakan untuk mengevaluasi kegunaan, yaitu, keberhasilan desain, berdasarkan kriteria yang dikembangkan sebelumnya [30]. Tahap evaluasi akan mengecek hasil data yang didapat dari tahap sebelumnya, kemudian dicari kontrol yang tepat untuk mencapai hasil yang diinginkan. Hasil evaluasi dibandingkan dari keseluruhan proses iterasi untuk mendapatkan informasi nilai mana yang terbaik. Sehingga pada proses berikutnya dapat diterapkan kontrol yang sesuai untuk membuatnya jadi lebih baik.



Gambar III.7 Alur kerja proses evaluasi model ML

1) *ML Model Testing*: Pada langkah ini, model ML akan diuji menggunakan dua dataset. Pertama menggunakan *validation dataset* yang labelnya diketahui namun disembunyikan, untuk mengetahui performa Akurasi dari algoritma. Setelah memiliki hasil yang cukup baik, kemudian model ML

akan dites menggunakan *testing dataset* tanpa label untuk memprediksi klasifikasi Outlier, Normal, atau Maintenance. Sementara itu, *Runtime* diukur dari awal program tes dieksekusi sampai hasil prediksi muncul.

2) Evaluasi Kemungkinan Peningkatan: Langkah ini menentukan apakah model dapat ditingkatkan untuk mencapai hasil yang lebih baik pada iterasi berikutnya. Salah satu cara untuk melakukannya adalah dengan menyesuaikan *Hyperparameter* yang berbeda. Contohnya adalah banyak unit tersembunyi dalam *multilayer perceptron*, k dalam k-NN, batas error di DT, fungsi kernel pada SVM, dan sebagainya. Kemungkinan lain adalah menyesuaikan dataset pelatihan atau menggabungkan beberapa algoritma [24]. Bagaimanapun caranya, penyesuaian dibuat kemudian kembali ke tahap pemodelan ML. Jika tidak ada perbaikan yang diamati setelah mencoba pendekatan yang berbeda, maka akan berlanjut ke langkah berikutnya.
3) Evaluasi Kriteria Keberhasilan: Hasil pengujian dibandingkan dengan Kriteria Sukses dan hasil dari penelitian yang serupa. Selain itu, hasil dari iterasi lain juga akan dibandingkan dan ditentukan mana yang lebih baik. Kriteria sukses untuk Akurasi minimal bernilai 90%, dengan Runtime maksimal sekitar 15 menit. Nilai-nilai ini hanyalah asumsi untuk hasil yang dapat diterima dan dapat berubah sebagai situasi berkembang. Jika hasil saat ini memuaskan keberhasilan kriteria, maka tahap ini selesai. Sementara itu, jika hasilnya masih kurang memuaskan, maka dilanjutkan ke tahap berikutnya.
4) *Analyze Problems*: Mencapai titik ini berarti kemungkinan besar tidak menemukan atau tidak memiliki cara untuk mempengaruhi faktor yang menyebabkan hasil untuk menjadi lebih baik. Permasalahan ini mungkin akan menyebabkan untuk kembali ke tahap sebelumnya dalam Metodologi agar menemukan pendekatan yang berbeda untuk solusi lain. Pada akhirnya, jika masalah tetap ada sampai periode penelitian ini berakhir, maka akan dibahas kemungkinan dan rekomendasi untuk pekerjaan yang akan datang.

BAB IV PROSES DESAIN MODEL PEMBELAJARAN MESIN

Pada Bab ini akan dibahas tahapan dan pertimbangan dalam mendesain model ML dan hasil yang didapatkan. Hasil desain awal merupakan bagian iterasi 0 dari metodologi penelitian ini, kemudian iterasi selanjutnya akan berusaha untuk meningkatkan hasil yang didapat. Tujuan yang ingin dicapai pada tahap ini adalah untuk berhasil membuat model ML klasifikasi yang mampu memberikan prediksi label yang merepresentasikan kondisi keperluan perawatan berdasarkan data sensor dari alat koleksi data yang terpasang pada kereta pembangkit. Kemudian hasil label Maintenance akan diolah untuk melakukan prediksi RUL.

IV.1 Persiapan Data

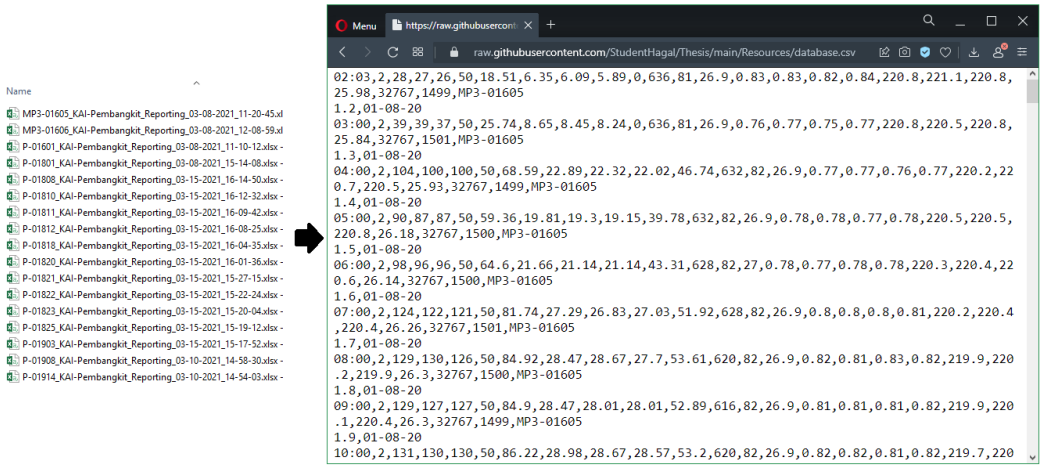
```
# Check the versions of libraries
# !!! WARNING !!!
# Important because model result may be different for other version

print('Python: {}'.format(sys.version))
print('scipy: {}'.format(scipy.__version__))
print('numpy: {}'.format(np.__version__))
print('matplotlib: {}'.format(matplotlib.__version__))
print('pandas: {}'.format(pd.__version__))
print('sklearn: {}'.format(sklearn.__version__))
```

```
Python: 3.7.11 (default, Jul  3 2021, 18:01:19)
[GCC 7.5.0]
scipy: 1.4.1
numpy: 1.19.5
matplotlib: 3.2.2
pandas: 1.1.5
sklearn: 0.22.2.post1
time: 10.1 ms (started: 2021-08-13 07:22:50 +00:00)
```

Gambar IV.1 Kode untuk menampilkan versi *library* yang digunakan

Pada tahap ini, kegiatan awal yang dilakukan adalah untuk membuat program di Google Colab menggunakan Bahasa pemrograman Python, detail versi keseluruhan library ditampilkan pada gambar. Proses yang dijelaskan disini mengacu pada proses desain model ML yang telah dijelaskan pada Bab sebelumnya.

IV.1.1 Membuat database standar



Gambar IV.2 Pemrosesan menjadi database standar

Untuk dapat lebih mudah mengolah data, maka dari 17 file didapatkan untuk jenis kereta yang berbeda data pemantauan sensor diambil lalu disatukan pada file dengan format csv, kemudian diupload ke layanan penyimpanan online untuk lebih mudah diakses. Standar yang dimaksud disini adalah bahwa memastikan pada setiap kolom merupakan entri data dari parameter yang sama, juga menyesuaikan tipe data agar seragam pada tiap kolom. Pada database tidak ada tag nama pada setiap kolom, ini akan ditambahkan saat pemrograman di tahap berikutnya.

```
1 # Checking hash value of a file
2 compare = '10f68054068bc4ac4d7fee65e8fd7a184151e3ce3abce5d85350fc13c412d93b'
3 message = hash_file('database.csv')
4 print('SHA-256 value of your file is :')
5 print(message)
6
7 if (compare == message) :
8   print('Hash check SUCCESS')
9 else :
10  print('Hash check INVALID')
```

```
SHA-256 value of your file is :
10f68054068bc4ac4d7fee65e8fd7a184151e3ce3abce5d85350fc13c412d93b
Hash check SUCCESS
time: 25.2 ms (started: 2021-08-13 07:22:51 +00:00)
```

Gambar IV.3 Pengecekan nilai hash file

Setelah itu juga ada pengecekan nilai Hash untuk memastikan integrasi file saat diunggah, diketahui dari pengecekan file lokal di laptop bahwa nilai SHA-256 file disimpan pada variabel compare. Kemudian nilai ini akan dibandingkan pada program dengan memanfaatkan hashlib yang akan menghitung nilai Hash dari file

yang diunggah. Jika ada perbedaan, maka harus diunggah ulang hingga berhasil memiliki nilai yang sama.

IV.1.2 Merekayasa Fitur

Pada tahap ini, penting sekali untuk memperhatikan urutan pemberian nama setiap kolom, karena jika salah maka nilainya tidak sesuai dengan data mentah yang didapatkan. Kesalahan ini menyebabkan hasil ML tidak merepresentasikan kondisi sesungguhnya, sehingga perlu dipastikan bahwa setiap nama pada setiap kolom sesuai dengan sensor yang tercatat dari dokumen KAI.

	Oil Pressure	Coolant Temp	PF Avg	ECU Temp
count	33832.000000	33832.000000	33832.000000	33832.000000
mean	1533.996364	663.118379	5.953150	6278.678647
std	8092.570600	4389.881127	40.234364	12862.110888
min	0.000000	0.000000	0.000000	0.000000
25%	592.000000	76.000000	0.780000	36.000000
40%	608.000000	79.000000	0.850000	39.000000
50%	612.000000	81.000000	0.870000	41.000000
60%	616.000000	81.000000	0.870000	43.000000
75%	624.000000	82.000000	0.890000	48.000000
max	65535.000000	65531.000000	327.650000	32767.000000

Gambar IV.4 Rekap statistik dari fitur terpilih

Kemudian ditentukan parameter yang akan menjadi fitur untuk digunakan dalam ML, dari 26 kolom untuk setiap sensor berbeda dipilih 4 parameter yang dianggap kritis dan memiliki nilai cukup bervariasi untuk dianalisis. Berdasarkan data mentah yang didapatkan, tidak ada label untuk dapat mengetahui kondisi keperluan perawatan, sehingga perlu memberikan label. Diasumsikan ada 3 label, yaitu Outlier, Normal, dan Maintenance. Dipertimbangkan 2 alternatif pendekatan yang dapat dilakukan untuk memberi label yaitu memanfaatkan algoritma clustering, atau pendekatan statistik.

IV.1.2.1 Pemberian label dengan pendekatan statistik

```
41 #Threshold value variable 40% min, 60% max
42 oil_threshold_40, oil_threshold_60 = df.Oil_Pressure.quantile([0.4 , 0.6])
43 coolant_threshold_40, coolant_threshold_60 = df.Coolant_Temp.quantile([0.4 , 0.6])
44 pf_threshold_40, pf_threshold_60 = df.PF_Avg.quantile([0.4 , 0.6])
45 ecu_threshold_40, ecu_threshold_60 = df.ECU_Temp.quantile([0.4 , 0.6])
46 #Threshold value variable 25% min, 75% max
47 oil_threshold_25, oil_threshold_75 = df.Oil_Pressure.quantile([0.25 , 0.75])
48 coolant_threshold_25, coolant_threshold_75 = df.Coolant_Temp.quantile([0.25 , 0.75])
49 pf_threshold_25, pf_threshold_75 = df.PF_Avg.quantile([0.25 , 0.75])
50 ecu_threshold_25, ecu_threshold_75 = df.ECU_Temp.quantile([0.25 , 0.75])
```

Gambar IV.5 Deklarasi variabel *threshold* kondisi

Untuk opsi awal akan dicoba menggunakan pendekatan statistik. Diambil nilai pada kuartil 25%, 40%, 60%, dan 75% untuk menjadi batasan kondisi, nilai ini dapat diganti berdasarkan standar dari KAI untuk parameter tersebut jika tersedia. Namun karena pada checksheet perawatan dari KAI tidak ada ketentuan batasan nilai pada parameter ini, sehingga diasumsikan menggunakan kondisi berikut.

- Stat_Outlier : x <= 25% or x >= 75%

```
#Stat_Outlier = value <= 25% or value >= 75%
Outlier_Oil = (df['Oil_Pressure'] <= oil_threshold_25) | (df['Oil_Pressure'] >= oil_threshold_75)
Outlier_Coolant =(df['Coolant_Temp'] <= coolant_threshold_25) | (df['Coolant_Temp'] >= coolant_threshold_75)
Outlier_ECU = (df['ECU_Temp'] <= ecu_threshold_25) | (df['ECU_Temp'] >= ecu_threshold_75)
Outlier_PFA = (df['PF_Avg'] <= pf_threshold_25) | (df['PF_Avg'] >= pf_threshold_75)

#All parameter have Stat_Outlier condition TRUE then it is labelled as Outlier
df_Outlier = df[Outlier_Oil & Outlier_Coolant & Outlier_ECU & Outlier_PFA]
df_Outlier.insert(4, 'Label', 'Outlier', True)
print('Outlier data')
print(df_Outlier.head(5))
print('Outlier data size : ', df_Outlier.shape)
print(' ')
```

```
Outlier data
     Oil_Pressure  Coolant_Temp  PF_Avg  ECU_Temp    Label
2           632.0          82.0    0.77   32767.0  Outlier
3           632.0          82.0    0.78   32767.0  Outlier
4           628.0          82.0    0.78   32767.0  Outlier
23            0.0           0.0    0.00       0.0  Outlier
39            0.0           0.0    0.00       0.0  Outlier
Outlier data size :  (7569, 5)
```

Gambar IV.6 Filter data yang memenuhi kondisi Outlier

Untuk label klasifikasi Outlier ditentukan bahwa jika nilai pada sensor berada pada nilai yang telah ditentukan, yaitu kurang dari kuartil 25% atau lebih dari kuartil 75%. Jika semua fitur yang digunakan memenuhi kondisi tersebut maka akan diberi label.

- Stat_Normal : 40% <= x <= 60%

```
#Stat_Normal = 40% <= value <= 60%
Normal_Oil = df['Oil_Pressure'].between(oil_threshold_40, oil_threshold_60)
Normal_Coolant = df['Coolant_Temp'].between(coolant_threshold_40, coolant_threshold_60)
Normal_ECU = df['ECU_Temp'].between(ecu_threshold_40, ecu_threshold_60)
Normal_PFA = df['PF_Avg'].between(pf_threshold_40, pf_threshold_60)

#When any parameter have all Stat_Normal condition TRUE then it is labelled as Normal
df_Normal = df[Normal_Oil & Normal_Coolant & Normal_ECU & Normal_PFA]
df_Normal.insert(4, 'Label', 'Normal', True)
print('Normal data')
print(df_Normal.head(5))
print('Normal data size : ', df_Normal.shape)
print(' ')
```

```
Normal data
      Oil_Pressure  Coolant_Temp  PF_Avg  ECU_Temp   Label
5339         616.0          81.0    0.87      42.0  Normal
5645         616.0          81.0    0.87      39.0  Normal
5822         616.0          81.0    0.86      41.0  Normal
5913         616.0          81.0    0.87      39.0  Normal
6109         616.0          81.0    0.87      39.0  Normal
Normal data size :  (372, 5)
```

Gambar IV.7 Filter data yang memenuhi kondisi Normal

Untuk label klasifikasi Outlier ditentukan bahwa jika nilai pada sensor berada pada nilai yang telah ditentukan, yaitu diantara kuartil 40% hingga kuartil 60% secara inklusif. Jika semua fitur yang digunakan memenuhi kondisi tersebut maka akan diberi label.

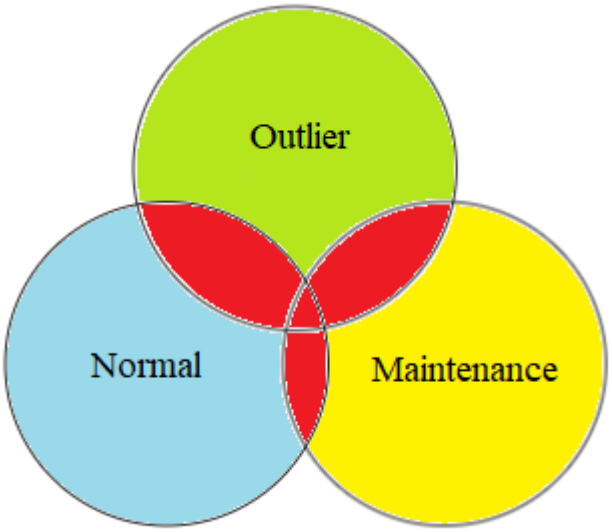
- Stat_Maintenance : 25% < x < 40% or 60% < x < 75%

```
#Stat_Maintenance = 25% < value < 40% or 60% < value < 75%
Maintenance_Oil = (df['Oil_Pressure'].between(oil_threshold_25, oil_threshold_40, inclusive=False)) | (df['Oil_Pressure
Maintenance_Coolant = (df['Coolant_Temp'].between(coolant_threshold_25, coolant_threshold_40, inclusive=False)) | (df[
Maintenance_ECU = (df['ECU_Temp'].between(ecu_threshold_25, ecu_threshold_40, inclusive=False)) | (df['ECU_Temp'].betwe
Maintenance_PFA = (df['PF_Avg'].between(pf_threshold_25, pf_threshold_40, inclusive=False)) | (df['PF_Avg'].between(pf_

#When any parameter have all Stat_Maintenance condition TRUE then it is labelled as Maintenance
df_Maintenance = df[Maintenance_Oil & Maintenance_Coolant & Maintenance_ECU & Maintenance_PFA]
df_Maintenance.insert(4, 'Label', 'Maintenance', True)
print('Maintenance data')
print(df_Maintenance.head(5))
print('Maintenance data size : ', df_Maintenance.shape)
print(' ')
```

```
Maintenance data
       Oil_Pressure  Coolant_Temp  PF_Avg  ECU_Temp        Label
12889         620.0          78.0    0.79      44.0  Maintenance
13010         604.0          78.0    0.81      46.0  Maintenance
13011         604.0          78.0    0.80      44.0  Maintenance
13287         620.0          77.0    0.79      38.0  Maintenance
13309         600.0          77.0    0.88      38.0  Maintenance
Maintenance data size :  (56, 5)

time: 34.7 ms (started: 2021-08-12 20:43:05 +00:00)
```

Gambar IV.8 Filter data yang memenuhi kondisi Maintenance

Untuk label klasifikasi Outlier ditentukan bahwa jika nilai pada sensor berada pada nilai yang telah ditentukan, yaitu diantara kuartil 25% hingga kuartil 40% atau diantara kuartil 60% hingga kuartil 75%. Jika semua fitur yang digunakan memenuhi kondisi tersebut maka akan diberi label.

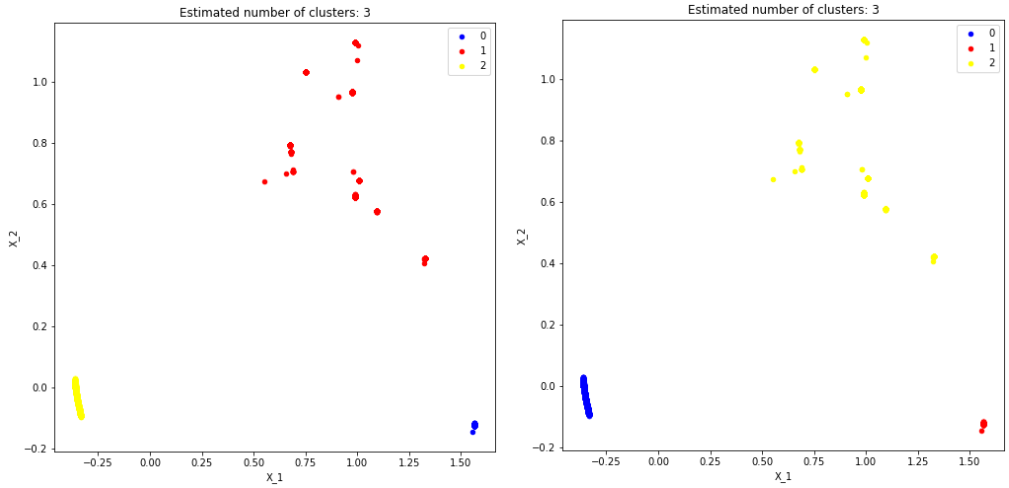


Gambar IV.9 Representasi Diagram Venn data

Pada dasarnya data yang digunakan dapat direpresentasikan dengan Diagram Venn seperti pada gambar tersebut. Rasio jumlah data pada setiap label yaitu 7569:372:56 untuk Outlier:Normal:Maintenance. Bagian berwarna merah tidak diberi label, karena beberapa fitur memiliki kondisi yang berbeda (misalnya Oil Pressure = Normal, Coolant Temp = Maintenance, PF Avg = Outlier, dan ECU Temp = Normal). Dari dataset, ada 9301 jumlah data yang tidak memiliki label. Data tanpa label yang direpresentasikan bagian merah inilah yang akan dicoba diprediksi kategorinya menggunakan model ML untuk PdM dari pendekatan ini.

IV.1.2.2 Pemberian label dengan algoritma clustering

Opsi yang lain adalah penggunaan algoritma clustering untuk memberikan label pada dataset. Ada 2 algoritma yang digunakan, yaitu DBSCAN dan Kmeans. DBSCAN dipertimbangkan karena penggunaan nya yang cocok untuk dapat mendeteksi Outlier sebagai noise dan memfilternya dari cluster, sehingga Outlier tidak diberi label dan dapat diabaikan. Sementara itu Kmeans dipertimbangkan karena kita dapat melakukan inisialisasi centroid sesuai jumlah cluster yang diinginkan, dalam kasus ini berjumlah 3 sesuai dengan label yang direncanakan. Untuk dapat menghasilkan cluster dengan jarak spasial yang konsisten, data perlu melalui *scaling* dan normalisasi. Kemudian dikarenakan ada 4 fitur untuk pertimbangan, perlu dilakukan reduksi dimensionalitas menjadi 2 dimensi untuk dapat divisualisasikan dalam grafik.

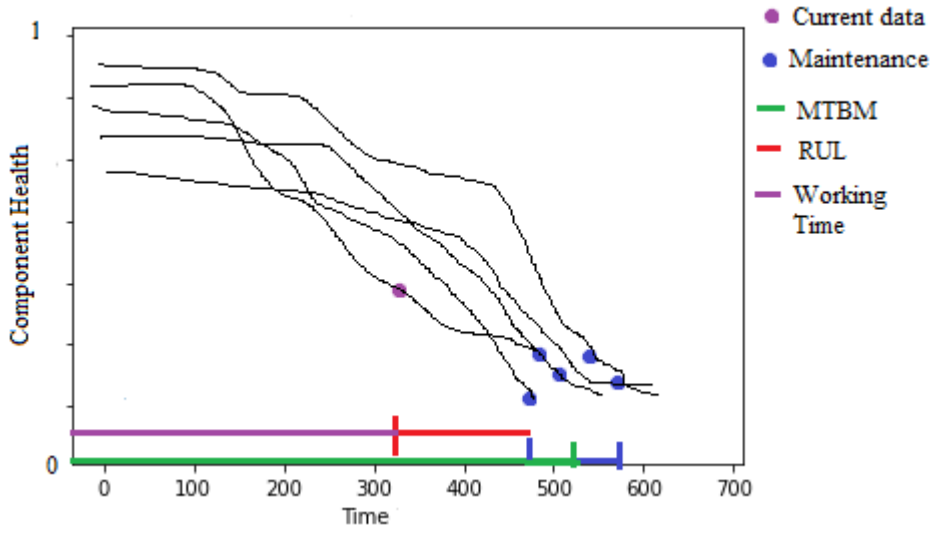


(a) (b)

Gambar IV.10 hasil cluster DBSCAN (a) dan hasil cluster Kmeans (b)

Untuk mendapat hasil ini, pengaturan pada *hyperparameter* untuk DBSCAN adalah eps=0.5 dan min_samples=100, sementara Kmeans yaitu n_clusters=3 dan random_state=0. Semua data pada dataset diberikan label untuk pendekatan ini, dengan rasio jumlah data pada setiap cluster DBSCAN yaitu 5475:1288:27594 dengan format biru:merah:kuning. Sementara pada cluster Kmeans yaitu 27594:5475:1288 dengan format yang sama. Dapat terlihat bahwa kedua algoritma menghasilkan jumlah cluster yang sama, walaupun lokasi cluster (ditunjukan dengan warna) berbeda. Hasil cluster ini berdasarkan posisi data relatif terhadap satu sama lain, sehingga belum jelas terkaitannya dengan kondisi pada kereta pembangkit.

IV.1.2.3 Menentukan RUL



Gambar IV.4 Ilustrasi grafik untuk menentukan RUL

Setelah data diberikan label sesuai kondisi, salah satu manfaat nya adalah untuk digunakan dalam menentukan *Mean Time Between Maintenance* (MTBM) yang diperlukan untuk menghitung RUL. Model regresi akan mengeluarkan prediksi *Working Time* berdasarkan nilai sensor, kemudian menghitung RUL dengan persamaan :

$$RUL = MTBM - Workng\ Time$$

Dengan mengetahui perkiraan RUL maka akan lebih memudahkan untuk menjadwalkan perawatan. Jika sebelumnya pada perawatan yang berbasis waktu, maka MTBM akan konstan karena perawatan selalu dilakukan pada periode yang

sama. Pada strategi perawatan PdM maka keperluan perawatan akan berbasis kondisi komponen, yang dapat memungkinkan waktu perawatan berbeda. Karena hasil MTBM bergantung pada data dengan label kondisi Maintenance, akan diuji beberapa alternatif dalam pemberian label tersebut untuk mengetahui perbedaan dan dampaknya dalam menghitung RUL. Alternatif opsi tersebut adalah sebagai berikut :

1. Semua Kereta kondisi OR

   Pada opsi ini, data diberikan label Maintenance jika salah satu fitur memenuhi kondisi Stat_Maintenance. Dapat diekspresikan sebagai: Fitur1 OR Fitur2 OR Fitur3 OR Fitur4, yang menghasilkan tabel kebenaran TRUE kecuali saat semua fitur tidak memenuhi kondisi Stat_Maintenance.
2. Semua Kereta kondisi AND

   Pada opsi ini, data diberikan label Maintenance jika semua fitur memenuhi kondisi Stat_Maintenance. Dapat diekspresikan sebagai: Fitur1 AND Fitur2 AND Fitur3 AND Fitur4, yang menghasilkan tabel kebenaran TRUE hanya saat semua fitur memenuhi kondisi Stat_Maintenance.
3. Satu Kereta kondisi OR

   Pada opsi ini, data diberikan label Maintenance jika salah satu fitur memenuhi kondisi Stat_Maintenance. Dapat diekspresikan sebagai: Fitur1 OR Fitur2 OR Fitur3 OR Fitur4, yang menghasilkan tabel kebenaran TRUE kecuali saat semua fitur tidak memenuhi kondisi Stat_Maintenance. Namun hanya diambil data satu jenis kereta saja, dipilih kereta dengan kode P-01908 karena memiliki jumlah data yang paling banyak diantara kereta lainnya.
4. Berdasarkan Hasil Model Klasifikasi DT

   Pada opsi ini akan digunakan data hasil klasifikasi oleh model DT, dengan demikian perlu menjalankan program klasifikasi terlebih dulu untuk melatih model, kemudian diberikan data tanpa label untuk mendapatkan hasil klasifikasi Maintenance. Model DT dipilih karena memiliki hasil terbaik dibandingkan algoritma klasifikasi lainnya pada kasus ini.
5. Semua Kereta kondisi OR dikelompokkan berdasarkan Train_code

Pada opsi ini, data diberikan label Maintenance jika salah satu fitur memenuhi kondisi Stat_Maintenance. Dapat diekspresikan sebagai: Fitur1 OR Fitur2 OR Fitur3 OR Fitur4, yang menghasilkan tabel kebenaran TRUE kecuali saat semua fitur tidak memenuhi kondisi Stat_Maintenance. Namun pada opsi ini data akan dikelompokkan berdasarkan kode kereta, dan MTBM dihitung berdasarkan masing-masing kereta.

IV.2 Mendesain model ML

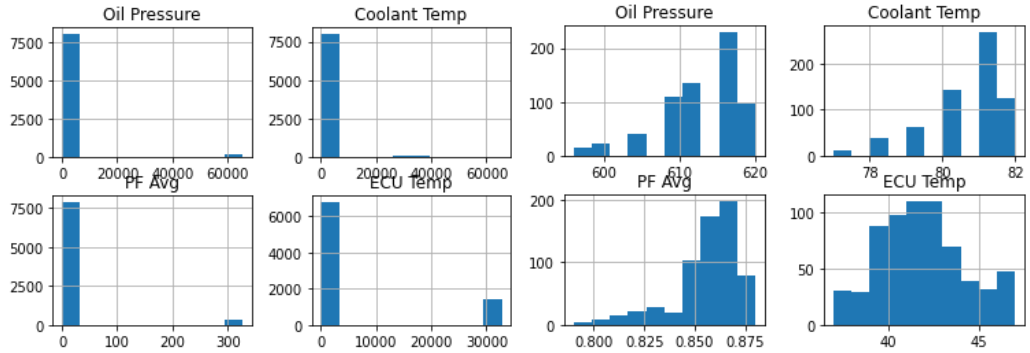
IV.2.1 Menentukan Algoritma Klasifikasi

```
15 # Algorithm considered
16 model_LR = LogisticRegression()
17 model_KNN = KNeighborsClassifier()
18 model_DT = DecisionTreeClassifier()
19 model_SVM = SVC()
20 model_RF = RandomForestClassifier()
21 model_LDA = LinearDiscriminantAnalysis()
22 model_NB = GaussianNB()
```

Gambar IV.11 Deklarasi variabel setiap algoritma model

Dengan pertimbangan berdasarkan survey [12] dan [17] akan dilakukan eksperiman untuk membandingkan beberapa jenis algoritma klasifikasi yang umum digunakan pada sistem PdM, yaitu: DT, SVM, LR, RF dan kNN. LR berbasis probabilitas cocok jika sampel semakin banyak. kNN berbasis jarak cocok untuk sampel sedikit pada penerapan awal. DT hanya butuh sedikit pre-proses data sehingga cocok untuk penerapan awal. SVM efektif pada dimensionalitas tinggi cocok jika menggunakan fitur yang semakin banyak. RF membangkitkan banyak pohon untuk pengambilan keputusan diharapkan dapat meningkatkan nilai Akurasi. Selain itu juga ditambahkan Naïve-Bayes (NB) karena berbasis probabilitas cocok jika sampel semakin banyak serta kemampuan skalabilitasnya dengan waktu pemrosesan yang cepat. *Linear discriminant analysis* (LDA) untuk kemampuan reduksi dimensionalitasnya sehingga akan cocok jika diterapkan dengan fitur yang semakin banyak.

IV.2.2 Melakukan *Pre-Processing* dataset Klasifikasi



(a) (b)

Gambar IV.12 Histogram data dengan Outlier (a) dan tanpa Outlier (b)

Pembuatan dataset bagi data dengan label statistik memiliki 2 opsi alternatif: (1) Dengan Outlier, ini berarti menyertakan data outlier untuk menjadi kelas dalam klasifikasi. (2) Tanpa Outlier, berarti memfilter data outlier sehingga tidak digunakan dalam klasifikasi. Keduanya akan digunakan untuk membandingkan opsi mana yang lebih baik.

```
# Split-out training and test dataset
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.20,
                                                    random_state=1, stratify=y)

print('training data size = ', X_train.shape)
print('testing data size = ', X_test.shape)
print('data split completed')
```

```
training data size =  (27485, 4)
testing data size =  (6872, 4)
data split completed
time: 21.7 ms (started: 2021-08-12 19:49:18 +00:00)
```

Gambar IV.13 Kode untuk memisah dataset

Selanjutnya dataset dengan label dibagi menjadi *training dataset* & *testing dataset* dengan rasio 4:1. Pembagian dilakukan secara acak berdasarkan kondisi random_state=1 untuk memastikan hasilnya dapat direproduksi, parameter stratify=y berguna agar pemecahan data memiliki rasio yang sama pada setiap kelas label.

IV.2.3 Melatih Model Klasifikasi

Setiap algoritma dilatih menggunakan *training dataset* untuk menghasilkan model ML yang dapat memprediksi label kondisi perawatan, serta dilakukan *cross validation* untuk mengetahui performa model.

```
# Cross validation of each model training in turn (with Outlier)
for name, model in models:
  kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
  cv_results = cross_val_score(model, X1_train, Y1_train, cv=kfold,
                               scoring='accuracy')
  results1.append(cv_results)
  names1.append(name)
  print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

# Compare result
pyplot.boxplot(results1, labels=names1)
pyplot.title('Algorithm Accuracy Comparison 1')
pyplot.show()
```

```
LR: 0.627856 (0.003277)
KNN: 0.999945 (0.000165)
DT: 1.000000 (0.000000)
SVM: 1.000000 (0.000000)
RF: 1.000000 (0.000000)
LDA: 0.400110 (0.004415)
NB: 0.989871 (0.002133)
```

Gambar IV.14 Kode untuk melatih model dan keluaran hasil validasi.

Metode *stratified k-fold* akan memecah *training dataset* menjadi 10 bagian yang ditentukan dengan variabel n_split=10, dan memastikan bahwa rasio setiap label akan seimbang. Dari 10 bagian tersebut, sebanyak 9 bagian digunakan untuk melatih dan 1 bagian akan digunakan untuk menguji model, kemudian dicatat nilai Akurasi nya. Proses ini diulangi hingga semua bagian mendapat giliran menjadi data uji, kemudian nilai Akurasi akan dihitung rata-rata nya menghasilkan skor Akurasi untuk algoritma tersebut. Selain itu variabel random_state=1 akan memisah data secara acak namun memastikan bahwa hasilnya tidak berubah jika program dijalankan ulang. Semua algoritma memiliki Hyperparameter yang diatur secara *default* oleh *library* pemrograman, belum dilakukan perubahan apapun untuk iterasi ini.

Tabel IV.1 Data hasil pelatihan awal model klasifikasi

Opsi	Metode	Kriteria	Algoritma

			LR	KNN	DT	SVM	RF	LDA	NB
1	Outlier, Random State=1	Acc	0.9462	0.9995	1	0.9465	1	0.9465	0.9737
		Runtime	11.9 s						
2	No Outlier, Random State=1	Acc	0.9853	0.9941	1	0.8685	1	0.9853	1
		Runtime	2.05 s						
3	DBSCAN, Random State=1	Acc	0.9367	0.9999	1	0.9839	0.9999	0.9945	0.9991
		Runtime	29.6 s						
4	Kmeans, Random State=1	Acc	0.9998	0.9999	0.9999	0.9843	0.9999	0.9940	0.9992
		Runtime	30.9 s						
Rata-Rata Akurasi			0.9670	0.9984	1	0.9458	1	0.9801	0.9930

Berdasarkan tabel tersebut terlihat bahwa Runtime nilainya serupa untuk semua algoritma di opsi yang sama. Hal ini dikarenakan model untuk validasi disimpan pada array, sehingga pemrosesan dilakukan dalam *batch* untuk semua algoritma menyebabkan pengukuran waktu tidak dapat dibedakan untuk algoritma individual. Selain itu juga muncul peringatan untuk meningkatkan Hyperparameter *max iteration* saat menjalankan LR opsi 1, 3, dan 4 sehingga menyebabkan waktu eksekusi meningkat signifikan. Kelas klasifikasi yang multinomial pada opsi tersebut, serta jumlah data yang lebih banyak mungkin juga menjadi penyebabnya. Dari perbandingan akurasi untuk keseluruhan opsi dan algoritma, DT dan RF menunjukan performa terbaik dengan skor Akurasi rata-rata 1 yang menyatakan semua label berhasil diprediksi dengan benar. Hasil terburuk yaitu pada model SVM, terutama pada opsi 2 yang bernilai dibawah kriteria sukses Akurasi.

IV.2.4 Menentukan Algoritma Regresi

```
#Algorithm Selection
model_RFreg = RandomForestRegressor(random_state=1)
model_OLS = LinearRegression()
model_Ridge = Ridge()
```

Gambar IV.15 Deklarasi variabel setiap algoritma model

Ada 3 algoritma Regresi dipertimbangkan: *Ordinary Least Square*, simpel dan umum digunakan cocok untuk menjadi basis awal. *Ridge*, lebih baik dalam mengatasi outlier untuk menutupi kelemahan OLS. *Random Forest Regressor*,

membangkitkan banyak pohon untuk pengambilan keputusan diharapkan dapat meningkatkan ketepatan prediksi.

IV.2.5 Melakukan *Pre-Processing* dataset Regresi

Sebelum mengolah dataset, dilakukan pengecekan nilai hash dari file yang akan digunakan sebagai dataset untuk memastikan bahwa file yang digunakan benar.

```
#load dataset for all vehicles
df_allvehicles = pd.read_csv(process_dir+'all_vehicles.csv', index_col=0)
MTBM_allvehicles = df_allvehicles.Delta_Time_h.mean()
print('Mean Time Between Maintenance : ',MTBM_allvehicles)
df_allvehicles
```

```
ean Time Between Maintenance :  3.914026095060578
```

	Datetime	Oil_Pressure	Coolant_Temp	PF_Avg	ECU_Temp	Label	Train_code	diffs	Delta_Time_h
14136	2020-01-04 00:00:00	624.0	78.0	0.87	37.0	Maintenance	P-01818	0 days 00:00:00	0.000000
14137	2020-01-04 01:00:00	624.0	77.0	0.87	37.0	Maintenance	P-01818	0 days 01:00:00	1.000000
14139	2020-01-04 03:00:00	620.0	78.0	0.89	39.0	Maintenance	P-01818	0 days 02:00:00	2.000000
2936	2020-01-04 07:00:00	620.0	82.0	0.89	43.0	Maintenance	P-01801	0 days 04:00:00	4.000000

Gambar IV.16 Blok kode untuk memuat dataset Opsi 1, 2 dan 5 RUL

Pada kode blok ini dapat dimuat file Opsi 1, 2 dan 5 dalam menentukan RUL, perlu diperhatikan kondisi nilai pada fitur serta Train_code untuk mengetahui opsi yang sedang diuji. File ‘all_vehicles.csv’ dihasilkan dari menjalankan program ‘Dataset_Processing.ipynb’ sehingga untuk mengubah opsi perlu menjalankan program tersebut dahulu dan mengganti kondisi yang digunakan.

```
#Preprocess data of a single vehicle
df_single = df_allvehicles.loc[df_allvehicles['Train_code'] == 'P-01908']
MTBM_single = df_single.Delta_Time_h.mean()
print('Mean Time Between Maintenance : ',MTBM_single)
df_single
```

```
Mean Time Between Maintenance :  7.365023474178405
```

	Datetime	Oil_Pressure	Coolant_Temp	PF_Avg	ECU_Temp	Label	Train_code	diffs	Delta_Time_h
28123	2020-01-10 04:04:00	616.0	81.0	0.86	38.0	Maintenance	P-01908	0 days 02:04:00	2.066667
28124	2020-01-10 05:00:00	616.0	81.0	0.88	39.0	Maintenance	P-01908	0 days 00:56:00	0.933333
28125	2020-01-10 06:00:00	612.0	81.0	0.85	42.0	Maintenance	P-01908	0 days 01:00:00	1.000000

Gambar IV.17 Blok kode untuk memuat dataset Opsi 3 RUL

Pada kode blok ini diolah dataset untuk Opsi 3 dalam menentukan RUL, nilai yang didapat berdasarkan dataset Opsi 1 namun hanya diambil data untuk kereta kode P-01908 saja. Jika file ‘all_vehicles.csv’ yang dihasilkan dari menjalankan program ‘Dataset_Processing.ipynb’ untuk Opsi 2, maka kode blok ini tidak perlu dijalankan

meskipun tidak masalah jika dibiarkan. Karena jumlah data yang didapatkan berjumlah sedikit sekali, maka tidak dipertimbangkan untuk menguji satu kereta kondisi AND dalam menentukan RUL.

```
#Preprocess dataset from ML classifier
df_MLlabel = pd.read_csv(process_dir+'predict1_smote.csv')
dropped_col = ['Unnamed: 0','LR', 'KNN', 'SVM', 'RF', 'LDA', 'NB']
df_MLlabel.drop(labels=dropped_col, axis=1, inplace=True)
df_MLlabel = df_MLlabel.loc[df_MLlabel['DT'] == 'Maintenance']
df_MLlabel['Datetime'] = pd.to_datetime(df_MLlabel.Datetime)
df_MLlabel = df_MLlabel.sort_values(by='Datetime')
df_MLlabel['Delta_Time'] = df_MLlabel.Datetime.diff()
df_MLlabel = df_MLlabel.fillna(pd.Timedelta(seconds=0))
df_MLlabel['Delta_Time_h'] = df_MLlabel['Delta_Time']/np.timedelta64(1,'h')
MTBM_MLlabel = df_MLlabel.Delta_Time_h.mean()
print('Mean Time Between Maintenance : ',MTBM_MLlabel)
df_MLlabel
```

```
Mean Time Between Maintenance :  6.035997120230381
```

	Datetime	Oil_Pressure	Coolant_Temp	PF_Avg	ECU_Temp	Train_code	DT	Delta_Time	Delta_Time_h
20361	2020-01-04 00:00:00	572.0	78.0	0.74	40.0	P-01822	Maintenance	0 days 00:00:00	0.000000
13889	2020-01-04 00:00:00	624.0	78.0	0.87	37.0	P-01818	Maintenance	0 days 00:00:00	0.000000

Gambar IV.18 Blok kode untuk memuat dataset Opsi 4 RUL

Pada kode blok ini dimuat file Opsi 4 dalam menentukan RUL, perlu diperhatikan algoritma yang digunakan dari file ‘predict1_smote.csv’ untuk menggunakan model dengan nilai Akurasi terbaik. Dengan demikian dapat diyakini bahwa hasil klasifikasi sesuai dengan kondisi komponen.

```
#separate feature (X) and predicted (y) parameters for ML classifier
X2 = df_MLlabel[features]
y2 = df_MLlabel['Delta_Time_h']
# Split-out training and test dataset for ML classifier
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X2, y2, test_size=0.20,
                                                        random_state=1)
print('Training data size (ML Classifier) = ', X2_train.shape)
print('Testing data size (ML Classifier) = ', X2_test.shape)

print('Data split completed')
```

```
Training data size =  (3433, 4)
Testing data size =  (859, 4)
Training data size (single vehicle) =  (340, 4)
Testing data size (single vehicle) =  (86, 4)
Training data size (ML Classifier) =  (2222, 4)
Testing data size (ML Classifier) =  (556, 4)
Data split completed
```

Gambar IV.19 Kode untuk memisah dataset RUL

Selanjutnya dataset untuk menentukan RUL dibagi menjadi *training dataset* & *testing dataset* dengan rasio 4:1. Pembagian dilakukan secara acak berdasarkan kondisi random_state=1 untuk memastikan hasilnya dapat direproduksi.

IV.2.6 Melatih Model Regresi

Setiap algoritma regresi dilatih menggunakan *training dataset* untuk menghasilkan waktu sejak perawatan terakhir, kemudian dilakukan *cross validation* dengan metric pengukuran *Mean Absolute Error* (MAE) untuk mengetahui performa model.

```
# Cross validation of each model training in turn
# Change variable X_train and Y_train if using other dataset (Check Dataset Split part for variables used)
for name, model in models:
  kfold = KFold(n_splits=10, random_state=1, shuffle=True)
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='neg_mean_absolute_error')
 #MEA scoring will be shown in negative, due to how cross_val_score scoring function works. The closer to 0
  val_results.append(cv_results)
  names.append(name)
  print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

# Compare result
pyplot.boxplot(val_results, labels=names)
pyplot.title('Algorithm MEA Comparison')
pyplot.show()
```

```
OLS: -4.732938 (1.283825)
Ridge: -4.710381 (1.294073)
RF-reg: -5.253497 (1.606916)
```

Gambar IV.20 Kode untuk melatih model regresi dan keluaran hasil validasi.

Metode *k-fold* akan memecah *training dataset* menjadi 10 bagian yang ditentukan dengan variabel n_split=10. Dari 10 bagian tersebut, sebanyak 9 bagian digunakan untuk melatih dan 1 bagian akan digunakan untuk menguji model, kemudian dicatat nilai MAE nya. Proses ini diulangi hingga semua bagian mendapat giliran menjadi data uji, kemudian nilai MAE akan dihitung rata-rata nya menghasilkan skor MAE akhir untuk algoritma tersebut. Selain itu variabel random_state=1 akan memisah data secara acak namun memastikan bahwa hasilnya tidak berubah jika program dijalankan ulang. Semua algoritma memiliki Hyperparameter yang diatur secara *default* oleh *library* pemrograman.

Tabel IV.2 Data hasil pelatihan model regresi

Opsi	Metode	Kriteria	Algoritma		
			OLS	Ridge	RF-Reg
1	All Vehicles (OR), Data size (6148, 7)	MAE	4.733	4.710	5.252
		Runtime	13.5 ms	12.8 ms	576 ms
2	All Vehicles (AND), Data size (56, 7)	MAE	417.915	405.328	373.485
		Runtime	1.02 s	286 ms	764 ms
3	Single Vehicle (OR), Data size (426, 7)	MAE	54.432	54.015	60.247
		Runtime	15 ms	25.5 ms	231 ms

4	ML Classifier label, Data size (2778, 7)	MAE	9.622	9.621	11.252
		Runtime	17.1 ms	10.9 ms	516 ms
5	All Vehicles (OR), Grouped by Train_code, data size (6148, 7)	MAE	59.408	59.396	64.934
		Runtime	15.2 ms	17.1 ms	616 ms
Rata-rata MAE			109.22	106.614	103.034

Dari perbandingan akurasi untuk keseluruhan opsi dan algoritma, OLS dan Ridge menunjukan performa terbaik dengan skor MAE yang saling berdekatan. Sementara alternatif yang menunjukan hasil terbaik adalah Opsi 1, meskipun Opsi 4 juga tidak berbeda jauh. Dari data yang didapatkan, dapat ditarik kesimpulan bahwa setidaknya dibutuhkan jumlah data sekitar 2700 untuk mendapatkan hasil error pada satu digit, dan mengasumsikan bahwa semua kereta memiliki MTBM yang sama.

IV.3 Mengevaluasi Model ML

Ada dua evaluasi yang akan dilakukan, pertama akan dievaluasi untuk mengecek kemungkinan untuk mendapatkan hasil yang lebih baik, kedua yaitu mengevaluasi hasil yang didapatkan dari *testing dataset* untuk menentukan kesuksesan model ML dengan target sukses yang ditentukan. Target sukses model klasifikasi berdasarkan dua kriteria, Akurasi minimal 90% dan Runtime maksimal 15 menit. Sementara untuk model regresi MAE lebih kecil dari 10 dan Runtime maksimal 15 menit. Ini masih merupakan asumsi untuk hasil yang dapat diterima, dan belum berdasarkan pada acuan tertentu.

IV.3.1 Menguji Model ML

1. Model Klasifikasi

   Pada tahap ini, akan digunakan *testing dataset* untuk menguji performa model ML dengan data yang diketahui labelnya. Dengan demikian dapat diketahui nilai Accuracy serta Runtime untuk dibandingkan dengan hasil training. Jika hasilnya konsisten, atau tidak terlalu menyimpang jauh dengan nilai Akurasi dari hasil *training model*, maka model ML telah terbukti bahwa model dapat digunakan untuk prediksi.

```
pred_KNN = model_KNN.predict(X_test)

print('KNN Result:')
print(accuracy_score(Y_test, pred_KNN))
print(classification_report(Y_test, pred_KNN))
plot_confusion_matrix(model_KNN, X_test, Y_test, values_format='d', cmap=plt.cm.Blues)
plt.show()
```

```
KNN Result:
0.9998544819557625
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1095
         1.0       1.00      1.00      1.00       258
         2.0       1.00      1.00      1.00      5519

    accuracy                           1.00      6872
   macro avg       1.00      1.00      1.00      6872
weighted avg       1.00      1.00      1.00      6872
```

Gambar IV.15 Blok Kode dan keluaran hasil menguji model KNN.

Metrik lain seperti Precision, Recall, dan F1-Score juga disertakan untuk dapat mengoptimalkan performa model, tetapi tidak digunakan untuk evaluasi keberhasilan penelitian. Selain itu juga dibuat kode untuk melakukan plot *confusion matrix* pada setiap model ML untuk mengetahui jumlah label yang berhasil diprediksi.

Tabel IV.3 Data hasil pengujian awal model klasifikasi

Opsi	Metode	Kriteria	Algoritma						
			LR	k-NN	DT	SVM	RF	LDA	NB
1	Statistical label, Outlier, Random State=1	Accuracy	0.05	0.0513	0.0538	0.0469	0.0538	0.0525	0.0538
		Precision	0.15	0.31	0.08	0.02	0.08	0.13	0.13
		Recall	0.48	0.55	0.67	0.33	0.67	0.61	0.67
		F1 Score	0.2	0.33	0.13	0.03	0.13	0.19	0.19
		Runtime (ms)	298	416	267	277	329	262	261
2	Statistical label, No Outlier, Random State=1	Accuracy	0.9302	0.9535	1	0.8721	1	0.9767	1
		Precision	0.96	0.97	1	0.44	1	0.99	1
		Recall	0.73	0.82	1	0.5	1	0.91	1
		F1 Score	0.79	0.88	1	0.47	1	0.94	1
		Runtime (ms)	233	231	216	369	252	269	222
3	DBSCAN, Random State=1	Accuracy	1	0.9999	0.9999	0.9849	0.9999	0.9939	0.9991
		Precision	1	1	1	0.97	1	1	0.99
		Recall	1	1	1	0.87	1	0.95	1

		F1 Score	1	1	1	0.9	1	0.97	1
		Runtime (ms)	214	675	256	459	232	162	186
4	Kmeans, Random State=1	Accuracy	0.9945	1	1	0.9833	1	0.9959	0.9990
		Precision	0.96	1	1	0.97	1	1	0.99
		Recall	0.99	1	1	0.85	1	0.96	1
		F1 Score	0.98	1	1	0.89	1	0.98	0.99
		Runtime (ms)	215	795	283	497	298	191	195
Rata-rata Akurasi			0.7437	0.75115	0.7634	0.7218	0.7634	0.7548	0.763
Rata-rata Runtime (ms)			240	529.25	255.5	400.5	277.75	221	216
Accuracy/Runtime (Normalized)			0.8773	0.4018	0.8459	0.5102	0.7781	0.9669	1

Dari hasil yang didapatkan setelah melakukan pengujian, hasil terbaik terjadi pada opsi 3 dan tidak jauh berbeda yaitu opsi 4, sementara itu opsi 2 sudah berhasil memenuhi kriteria sukses kecuali pada algoritma SVM, dengan hasil terburuk tampil pada opsi 1.

2. Model Regresi

```
predictions_Ridge = model_Ridge.predict(X_test)
print('Ridge MEA Result:')
print(mean_absolute_error(Y_test, predictions_Ridge))
```

```
Ridge MEA Result:
5.663608601447112
```

Gambar IV.16 Blok kode dan keluaran hasil menguji model Ridge.

Pada tahap ini, akan digunakan *testing dataset* untuk menguji performa model ML dengan data yang diketahui labelnya. Dengan demikian dapat diketahui nilai MAE untuk dibandingkan dengan hasil pelatihan. Jika hasilnya konsisten, atau tidak terlalu menyimpang jauh dengan nilai dari hasil pelatihan, maka model ML telah terbukti bahwa model dapat digunakan untuk prediksi. Hasil prediksi tidak langsung menunjukan hasil RUL, melainkan perkiraan waktu dari perawatan terakhir. Untuk mengetahui RUL maka nilai MTBM perlu dikurang hasil prediksi.

Tabel IV.4 Data hasil pengujian model regresi

Opsi	Metode	Kriteria	Algoritma		
			OLS	Ridge	RF-Reg
1	All Vehicles (OR), Data size (6148, 7)	MAE	5.679	5.664	5.841
		Runtime	4.19 ms	8.72 ms	31.3 ms
2	All Vehicles (AND), Data size (56, 7)	MAE	201.43	216.872	555.324
		Runtime	4.66 ms	5.07 ms	16.3 ms
3	Single Vehicle (OR), Data size (426, 7)	MAE	67.19	67.34	95.128
		Runtime	6.64 ms	3.49 ms	23.2 ms
4	ML Classifier label, Data size (2778, 7)	MAE	6.717	6.717	8.262
		Runtime	3.27 ms	4.65 ms	26.9 ms
5	All Vehicles (OR), Grouped by Train_code, data size (6148, 7)	MAE	54.272	54.178	82.691
		Runtime	7.65 ms	4.59 ms	41 ms
Rata-rata MAE			67.058	70.154	149.449
Rata-rata Runtime			5.282	5.304	27.74
MAE/Runtime (Normalized)			0.96	1	0.407

Dari hasil yang didapatkan setelah melakukan pengujian, hasil terbaik terjadi pada opsi 1 dan tidak jauh berbeda yaitu opsi 4, sementara itu opsi lainnya gagal memenuhi kriteria sukses dengan hasil terburuk tampil pada opsi 2.

IV.3.2 Mengevaluasi Kemungkinan Peningkatan Kinerja Model ML

1. Model Klasifikasi

Hasil pemodelan untuk saat ini masih menggunakan *hyperparameter* default yang diatur oleh *library* scikit-learn [31], sehingga masih ada peluang untuk meningkatkan akurasi menjadi lebih baik. Namun berdasarkan hasil dari Tabel IV.2 dapat terlihat bahwa kebanyakan algoritma sudah berhasil memenuhi target kriteria sukses, sehingga tidak perlu difokuskan untuk melakukan optimasi pada algoritma tersebut. Dari opsi yang ada, opsi 1 memberikan hasil yang sangat buruk untuk mengidentifikasi label Outlier. Berdasarkan rasio label yang tidak

seimbang, maka akan dicoba untuk melakukan teknik *oversampling*. Selain itu untuk algritma LR ada peringatan untuk meningkatkan *hyperparameter* max_iteration dari nilai default 100 hingga tidak muncul lagi peringatan tersebut, jika masih gagal akan dicoba untuk mengubah random_state dan *hyperparameter* lainnya. Pada algoritma SVM untuk seluruh opsi juga menunjukan performa paling buruk diantara lainnya, dapat ditingkatkan dengan mengatur *hyperparameter* dalam membuat model.

2. Model Regresi

Hasil pemodelan untuk saat ini masih menggunakan *hyperparameter* default yang diatur oleh *library* scikit-learn [31], sehingga masih ada peluang untuk meningkatkan akurasi menjadi lebih baik. Namun berdasarkan hasil dari Tabel IV.4 dapat terlihat bahwa opsi 1 dan opsi 4 sudah berhasil memenuhi target kriteria sukses, sehingga tidak perlu difokuskan untuk melakukan optimasi. Dari alternatif yang diuji, opsi 2 memberikan hasil MAE yang buruk dan masih bisa ditingkatkan lagi. Sementara opsi 3 dan 5 nilainya juga masih belum mencapai target, walaupun hasilnya tidak terlalu buruk.

IV.3.3 Mengevaluasi Keberhasilan Model ML

1. Model Klasifikasi

Berdasarkan hasil dari Tabel IV.3 dapat terlihat bahwa kebanyakan algoritma sudah berhasil memenuhi target kriteria sukses, dari opsi 2 hanya perlu melakukan optimasi pada algoritma SVM, sedangkan untuk opsi 1 masih sangat jauh dari target. Sementara itu, untuk opsi 3 dan 4 sudah menunjukan hasil hampir sempurna sehingga dapat dikatakan sudah sukses. Meski begitu, ada permasalahan lain yang muncul jika akan menerapkan opsi 3 dan 4 ini yang akan dibahas pada bagian selanjutnya. Dengan demikian keseluruhan opsi 2 kecuali SVM, serta 3 dan 4 dapat digunakan untuk melakukan prediksi pada data tanpa label dari input user atau mengambil nilai acak.

2. Model Regresi

Dari alternatif yang diuji, opsi 2 memberikan hasil MAE yang paling buruk dan masih bisa ditingkatkan lagi. Melihat jumlah data yang sedikit saat dilakukan pengujian, maka diperkirakan bahwa perlu menambahkan jumlah sampel untuk mendapat hasil yang lebih baik pada opsi 2 dan 3 tersebut. Sementara untuk opsi 5 sudah menggunakan jumlah data yang cukup banyak, namun masih mendapatkan nilai yang kurang memuaskan. Kemungkinan lainnya yaitu dengan mengubah *hyperparameter* setiap algoritma.

IV.3.4 Menganalisis Masalah

1. Model Klasifikasi

Hasil pengujian pada Tabel IV.3 menunjukan bahwa opsi 1 memiliki nilai yang Akurasi yang sangat rendah, berdasarkan confusion matrix yang ditampilkan pada program diketahui bahwa semua model gagal untuk memprediksi kondisi Outlier yang merupakan kelas label terbanyak. Karena tidak seimbang nya kelas label, menyebabkan akurasi secara keseluruhan rendah walaupun model ML berhasil mengidentifikasi kelas lainnya dengan tepat. Selain itu pada algoritma LR di opsi 1 ada peringatan untuk meningkatkan *hyperparameter* max_iteration dari nilai default 100, tidak tercapainya konvergensi merupakan indikasi bahwa data mungkin tidak cocok dengan model ML, karena ada terlalu banyak pengamatan yang tidak sesuai. Cara mengatasi masalah-masalah ini telah dibahas pada bagian sebelumnya.

Untuk opsi 2 belum ada masalah yang perlu dibahas detail, sehingga akan lanjut membahas permasalahan jika akan menggunakan opsi 3 dan 4 sebagai pilihan untuk menerapkan PdM. Opsi menggunakan algoritma clustering ini menghasilkan Akurasi yang bagus dalam memprediksi kelas label dari cluster, namun cluster ini tidak diketahui keterkaitannya dengan kondisi kereta pembangkit. Sehingga perlu dianalisis lebih lanjut dengan mempelajari data hasil cluster, dibutuhkan pengetahuan mengenai domain

perawatan kereta pembangkit untuk dapat menjelaskan relasi antara cluster dengan kondisi kereta pembangkit.

2. Model Regresi

Untuk opsi 1 dan 4 sudah menunjukan hasil yang baik, namun menggunakan asumsi bahwa semua kereta memiliki performa yang sama sehingga tidak dipertimbangkan bahwa data berasal dari kereta yang berbeda. Asumsi ini tidak sesuai dengan kondisi nyata, sehingga performa model yang menggunakan opsi ini belum tentu cocok diterapkan.

Pada opsi 2 karena jumlah data yang sedikit, menyebabkan hasil yang buruk. Apabila dikembangkan lebih lanjut dengan menggunakan fitur yang semakin banyak, maka jumlah data yang memenuhi kondisi akan semakin sulit terpenuhi sehingga disimpulkan bahwa opsi ini tidak cocok untuk diterapkan.

Opsi 3 juga memiliki jumlah data yang sedikit, walaupun lebih banyak dari opsi 2. Namun untuk opsi 3 ini jumlah data dapat diperbanyak dengan meningkatkan waktu pengambilan sampel pada kereta yang berkaitan, atau mengumpulkan data pada periode yang lebih lama lagi. Bagaimanapun membutuhkan waktu lebih untuk memperbanyak data yang dimiliki, sehingga untuk penelitian ini dicukupkan sampai sini saja.

Sementara pada opsi 5 data sudah terkumpul dalam jumlah yang cukup banyak, sama dengan jumlah data pada opsi 1. Meski demikian hasil yang didapatkan belum mencapai target yang diinginkan. Ini dikarenakan nilai waktu antar perawatan pada setiap kereta ternyata memiliki perbedaan yang cukup besar, mungkin disebabkan karena usia dan daerah operasi yang berbeda-beda sehingga sulit ditentukan jika menjadi satu dataset. Untuk mengatasi masalah ini diperkirakan perlu menambahkan fitur ‘Train_Code’ agar model dapat mengetahui bahwa data berasal dari kereta yang berbeda.

IV.4 Optimalisasi Model ML Klasifikasi

IV.4.1 Penyesuaian pengaturan model LR dan SVM

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Conve
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Gambar IV.16 Pesan peringatan saat melatih dan validasi model LR

Saat melatih dan menguji model LR pada opsi 1 menggunakan pengaturan *default*, ternyata didapatkan peringatan untuk meningkatkan parameter max_iter dari nilai *default* yaitu 100 karena algoritma belum mencapai konvergensi. Walaupun demikian, program tetap berjalan dan dapat mengeluarkan hasil Akurasi seperti yang ditampilkan pada Tabel IV.1 dan Tabel IV.2. Untuk mengatasi isu ini, maka dilakukan perubahan max_iter dengan mengambil nilai 500 secara asal, kemudian melakukan *binary search* untuk mendapatkan nilai yang tepat. Setelah beberapa kali pengujian hingga tidak muncul lagi peringatan tersebut, didapat nilai sekitar 475 saat peringatan tersebut tidak muncul lagi, sehingga diatur max_iter=475. Namun ternyata ini berkonsekuensi meningkatnya Runtime menjadi 28,3 detik dari 11,9 detik tanpa meningkatkan Akurasi secara signifikan.

```
# Algorithm considered
model_LR = LogisticRegression(max_iter=475)
model_KNN = KNeighborsClassifier()
model_DT = DecisionTreeClassifier()
model_SVM = SVC(gamma='auto')
model_RF = RandomForestClassifier()
model_LDA = LinearDiscriminantAnalysis()
model_NB = GaussianNB()
model_Kmeans = KMeans(n_clusters=3, random_state=0)
model_dbscan = DBSCAN(eps=0.5, min_samples=100)
```

Gambar IV.17 Pengaturan parameter model setiap algoritma

Pada model SVM untuk opsi 2 nilainya masih dibawah kriteria sukses, sehingga perlu dilakukan perubahan juga pengaturannya. Parameter gamma=’scale’ yang merupakan pengaturan *default* diubah menjadi gamma=’auto’ yang mengganti nilai koefisien dari kernel ‘rbf’ menjadi bernilai 1 / n_features. Perubahan ini menghasilkan nilai Akurasi 0.9912 dalam pelatihan dan Akurasi 0.9767 saat pengujian, meningkat hingga memenuhi kriteria sukses dengan mengubah ini saja.

IV.4.2 *Oversampling* data

```
# Pre-process training and test dataset to balance class label (with outlier)
array1 = labeled_dataset1.values
X1 = array1[:,0:4]
y1 = array1[:,4]

print('Original dataset shape %s' % Counter(y1))
balancing_smote = SMOTE(random_state=1)
X1_res, y1_res = balancing_smote.fit_resample(X1, y1)
print('Resampled dataset shape %s' % Counter(y1_res))
```

```
Original dataset shape Counter({'Outlier': 7569, 'Normal': 372, 'Maintenance': 56})
Resampled dataset shape Counter({'Maintenance': 7569, 'Outlier': 7569, 'Normal': 7569})
time: 115 ms (started: 2021-08-12 17:12:14 +00:00)
```

Gambar IV.18 Pengaturan parameter model setiap algoritma

Hasil Akurasi yang buruk saat pengujian pada opsi 1 untuk semua model dikarenakan tidak seimbang nya jumlah data setiap kelas label, menyebabkan akurasi secara keseluruhan rendah walaupun model ML berhasil mengidentifikasi kelas lainnya dengan tepat. Salah satu solusi untuk permasalahan ini adalah dengan teknik *oversampling* untuk menyeimbangkan jumlah data. Berdasarkan rasio jumlah data pada label sebelumnya, ada 7569:372:56 data Outlier:Normal:Maintenance. Dengan menggunakan teknik Synthetic Minority Oversampling Technique (SMOTE) akan dibuat merata mengikuti jumlah data terbanyak, sehingga masing-masing label memiliki 7569 jumlah data. Tahap ini juga dilakukan untuk opsi 3 dan 4, karena hasil cluster memiliki jumlah data tidak seimbang.

Tabel IV.5 Data hasil pelatihan model klasifikasi optimal

Opsi	Metode	Kriteria	Algorithm						
			LR	KNN	DT	SVM	RF	LDA	NB
1	Statistical label, Outlier, Random State=1, SMOTE	Acc	0.6279	0.9999	1	1	1	0.4001	0.9899
		Runtime	42.4 s						
2	Statistical label, No Outlier, Random State=1, SMOTE	Acc	0.99	1	1	1	1	0.9899	1
		Runtime	2.24 s						
3		Acc	1	1	1	0.9964	1	0.9508	0.9987

	DBSCAN, Random State=1, SMOTE	Runtime	1232 s						
4	Kmeans, Random State=1, SMOTE	Acc	1	1	1	0.9963	1	0.9508	0.9987
		Runtime	1185 s						
Rata-Rata Akurasi			0.9045	1	1	0.9982	1	0.8229	0.9968

Berdasarkan hasil optimasi dengan *oversampling* menggunakan SMOTE, perlu diperhatikan bahwa Runtime meningkat untuk opsi 2, 3 dan 4 yang dikarenakan perlunya memproses data yang lebih banyak dari semula. Dapat terlihat juga ada pengurangan akurasi pada model LR dan LDA saat pelatihan, sehingga mungkin tidak cocok untuk digunakan pada kasus ini. Sementara untuk opsi lainnya semua model sudah menunjukan hasil Akurasi yang baik, sehingga tidak perlu dilakukan optimalisasi lebih lanjut.

Tabel IV.6 Data hasil pengujian model klasifikasi optimal

Opsi	Metode	Kriteria	Algoritma						
			LR	k-NN	DT	SVM	RF	LDA	NB
1	Statistical label, Outlier, Random State=1, SMOTE	Accuracy	0.619	0.9998	1	1	1	0.3835	0.9921
		Precision	0.49	1	1	1	1	0.45	0.99
		Recall	0.63	1	1	1	1	0.4	0.99
		F1 Score	0.52	1	1	1	1	0.28	0.99
		Runtime (ms)	369	767	339	596	431	320	290
2	Statistical label, No Outlier, Random State=1, SMOTE	Accuracy	1	1	1	1	1	0.9933	1
		Precision	1	1	1	1	1	0.99	1
		Recall	1	1	1	1	1	0.99	1
		F1 Score	1	1	1	1	1	0.99	1
		Runtime (ms)	207	211	203	329	243	218	213
3	DBSCAN, Random State=1, SMOTE	Accuracy	1	0.9999	0.9999	0.9973	0.9999	0.9522	0.9984
		Precision	1	1	1	1	1	0.96	1
		Recall	1	1	1	1	1	0.95	1
		F1 Score	1	1	1	1	1	0.95	1

		Runtime (ms)	305	2340	703	6370	479	244	264
4	Kmeans, Random State=1, SMOTE	Accuracy	1	0.9999	1	0.9973	1	0.9522	0.9984
		Precision	1	1	1	1	1	0.96	1
		Recall	1	1	1	1	1	0.95	1
		F1 Score	1	1	1	1	1	0.95	1
		Runtime (ms)	268	1840	351	5830	500	278	255
Rata-rata Akurasi			0.9047	0.9999	1	0.9986	1	0.8203	0.9972
Rata-rata Runtime (ms)			287.25	1289.5	399	3281.3	413.25	265	255.5
Accuracy/Runtime (Normalized)			0.807	0.19868	0.642	0.078	0.62	0.7931	1

Hasil optimasi dengan SMOTE pada pengujian untuk opsi 1 dapat terlihat bahwa nilai Akurasi meningkat secara signifikan, seluruh model kecuali LR dan LDA berhasil memprediksi hampir semua label dengan benar. Walaupun ada pengurangan akurasi pada model LR dan LDA, sehingga mungkin tidak cocok untuk digunakan pada kasus ini. Sementara untuk opsi lainnya semua model sudah menunjukan hasil Akurasi yang baik, sehingga optimalisasi dicukupkan. Metrik Accuracy/Runtime ditambahkan untuk mengetahui relasi keduanya pada model ML terkait.

BAB V FINALISASI MODEL PEMBELAJARAN MESIN

Pada Bab ini akan dibahas hasil pengujian model ML yang telah didesain, dengan memberikan data tanpa label untuk diolah. Data tanpa label ini didapatkan dari sisa pendekatan statistik, serta masukan pengguna untuk setiap nilai fitur ataupun berupa file dengan format yang sesuai. Hasil pengujian ini tidak dapat diketahui metrik ketepatan prediksinya, karena tidak ada nilai/label untuk dibandingkan. Pengujian ini merepresentasikan penggunaan model ML di lapangan, dimana data yang didapatkan tidak memiliki label dan masih berdasarkan pengalaman teknisi pada setiap depo perawatan untuk mengetahui kondisi baik atau tidak. Kemudian juga akan dibahas pertimbangan alternatif opsi dan algoritma yang cocok untuk diterapkan menggunakan *decision matrix*.

V.1 Menguji model ML dengan data tanpa label

Pada bagian ini hanya akan diuji model ML opsi 1 dan 2 karena dengan pendekatan statistik, tidak semua data pada dataset diberikan label, sehingga data tanpa label tersebut dapat digunakan untuk menguji hasil prediksi model. Sementara untuk opsi 3, 4 dan model ML Regresi pengujian akan langsung dilakukan dengan masukan pengguna yang tidak akan dibahas pada bagian ini. Karena tidak diketahui label kondisi sesungguhnya, menyebabkan tidak dapat diukurnya Akurasi dari model ML. Oleh karena itu, perlu dipilih model yang memiliki hasil terbaik pada pengujian sebelumnya jika akan diterapkan pada sistem PdM.

```
#Separate unlabelled data
df_Unlabelled = pd.concat([df, df_Outlier, df_Outlier,
                           df_Normal, df_Normal,
                           df_Maintenance, df_Maintenance]).drop_duplicates(keep=False)
df_Unlabelled.drop('Label',axis=1, inplace=True)
print('Unlabelled data')
print(df_Unlabelled.head(5))
print('Unlabelled data size : ', df_Unlabelled.shape)
```

```
Unlabelled data
   Oil_Pressure  Coolant_Temp  PF_Avg  ECU_Temp
0         636.0          81.0    0.83   32767.0
2         632.0          82.0    0.77   32767.0
3         632.0          82.0    0.78   32767.0
5         628.0          82.0    0.80   32767.0
9         616.0          83.0    0.84   32767.0
Unlabelled data size :  (9301, 4)
```

Gambar IV.19 Kode untuk memisahkan data tanpa label dan keluarannya

Pada tahap *feature engineering* saat pemberian label untuk opsi 1 dan 2, dilakukan juga pemisahan data tanpa label dari data lainnya. Jumlah data tanpa label adalah 9301 baris, yang akan digunakan sebagai masukan data untuk prediksi oleh model ML.

```
# Make predictions on unlabelled dataset with outlier
predictions1_LR = model1_LR.predict(df_Unlabelled[features])
predictions1_KNN = model1_KNN.predict(df_Unlabelled[features])
predictions1_DT = model1_DT.predict(df_Unlabelled[features])
predictions1_SVM = model1_SVM.predict(df_Unlabelled[features])
predictions1_RF = model1_RF.predict(df_Unlabelled[features])
predictions1_LDA = model1_LDA.predict(df_Unlabelled[features])
predictions1_NB = model1_NB.predict(df_Unlabelled[features])

df_predict1 =  pd.DataFrame({'LR': predictions1_LR, 'KNN': predictions1_KNN,
                             'DT':predictions1_DT, 'SVM':predictions1_SVM,
                             'RF':predictions1_RF, 'LDA':predictions1_LDA,
                             'NB':predictions1_NB})

df_predict1
```

	LR	KNN	DT	SVM	RF	LDA	NB
0	Outlier	Outlier	Normal	Outlier	Normal	Outlier	Outlier
1	Outlier	Outlier	Normal	Outlier	Outlier	Outlier	Outlier

Gambar IV.20 Potongan kode dan hasil prediksi data tanpa label

Setelah diproses oleh model ML, didapatkan hasil prediksi label kondisi sarana kereta api yang kemudian ditampilkan pada program, hanya terlihat sekitar 5 entri pertama saja dari setiap model ML. Selanjutnya hasil ini disimpan dalam file ‘predict1_smote.csv’ untuk dapat dilihat lebih lengkap. Proses seperti ini diulangi untuk opsi 2, dan hasilnya disimpan dalam file ‘predict2_smote.csv’ untuk dapat dilihat lebih lengkap. Jika hasil dianggap sudah cukup, model kemudian akan disimpan pada file sehingga dapat langsung digunakan untuk tahap selanjutnya.

V.2 Menguji model ML dengan masukan pengguna

Pada bagian ini semua model dari berbagai opsi akan diuji dengan masukan pengguna yang kemudian akan mengeluarkan hasil prediksi label. Untuk mempersingkat waktu pemrosesan, pengujian ini dilakukan dengan memuat model yang sudah tersimpan pada suatu file dari hasil pelatihan sebelumnya.

```
# user input
print('-----Prediction Test on ML Model-----')
print('Enter value for Oil_Pressure sensor:')
feature1 = float(input())
print('Enter value for Coolant_Temp sensor:')
feature2 = float(input())
print('Enter value for PF_Avg sensor:')
feature3 = float(input())
print('Enter value for ECU_Temp sensor:')
feature4 = float(input())

df_input = pd.DataFrame({'Oil_Pressure': [feature1], 'Coolant_Temp': [feature2],
                         'PF_Avg':[feature3], 'ECU_Temp':[feature4]})
```

```
-----Prediction Test on ML Model-----
Enter value for Oil_Pressure sensor:
0
Enter value for Coolant_Temp sensor:
61
Enter value for PF_Avg sensor:
361
Enter value for ECU_Temp sensor:
61
```

Gambar IV.21 Kode untuk menerima masukan nilai sensor dari pengguna

Dibuat kode untuk menerima masukan pengguna dan mengubahnya menjadi tipe data yang serupa dengan fitur yang digunakan yaitu float. Karena ada 4 fitur, maka perlu mengisi semuanya. Kemudian data masukan disimpan pada pandas DataFrame, dengan format yang menyerupai *training dataset* untuk dapat menghasilkan prediksi.

```
#user input path to file
print('-----Prediction Test on ML Model-----')
print('Enter value for File path: ')
file_path = str(input())
```

```
-----Prediction Test on ML Model-----
Enter value for File path:
/content/drive/MyDrive/Colab Notebooks/ML Resources/Raw Data/Format csv/database.csv
```

Gambar IV.22 Kode untuk menerima masukan file dari pengguna

Dibuat kode untuk menerima masukan pengguna berupa file yang berisi banyak data sensor sehingga bisa diolah sekaligus. Ini bermanfaat jika ingin menganalisa data yang terkumpul dalam periode tertentu tanpa harus memasukan nilai secara manual satu demi satu. Setelah data file diolah oleh ML untuk menghasilkan prediksi terkait, kemudian hasilnya akan di eksport pada file lain beserta hasil prediksinya.

```
# print prediction result for each model from user input
predictions_LR = Opsi1_LR.predict(df_file)
predictions_KNN = Opsi1_KNN.predict(df_file)
predictions_DT = Opsi1_DT.predict(df_file)
predictions_SVM = Opsi1_SVM.predict(df_file)
predictions_RF = Opsi1_RF.predict(df_file)
predictions_LDA = Opsi1_LDA.predict(df_file)
predictions_NB = Opsi1_NB.predict(df_file)

df_predict = pd.DataFrame({'LR': predictions_LR, 'KNN': predictions_KNN,
                           'DT':predictions_DT, 'SVM':predictions_SVM,
                           'RF':predictions_RF, 'LDA':predictions_LDA,
                           'NB':predictions_NB})
df_predict
```

	LR	KNN	DT	SVM	RF	LDA	NB
0	Outlier	Outlier	Normal	Outlier	Normal	Outlier	Outlier
1	Outlier	Outlier	Normal	Outlier	Outlier	Outlier	Outlier

Gambar IV.23 Kode dan hasil Klasifikasi dari masukan pengguna

Model klasifikasi yang telah disimpan dalam suatu file dimuat pada program, sehingga tidak perlu menghabiskan waktu untuk melakukan pelatihan model terlebih dulu untuk mengeluarkan hasil prediksi. Selanjutnya model akan mengolah masukan, dan mengeluarkan prediksi label seperti yang ditampilkan pada gambar. Penting untuk memilih model terbaik sehingga hasilnya dapat dipercaya untuk merepresentasikan kondisi sesungguhnya.

```
print('Enter value for MTBM:')
MTBM = float(input())
```

```
Enter value for MTBM:
3.914
time: 7.77 s (started: 2021-10-17 09:10:51 +00:00)
```

```
#RUL result is MTBM_Condition-Prediction (differ with each dataset)
predictions_OLS = RUL_OLS.predict(df_file)
predictions_Ridge = RUL_Ridge.predict(df_file)
predictions_RFreg = RUL_RFreg.predict(df_file)

df_RUL = pd.DataFrame({'OLS': MTBM-predictions_OLS, 'Ridge': MTBM-predictions_Ridge,
                       'RF-reg':MTBM-predictions_RFreg})
df_RUL
```

	OLS	Ridge	RF-reg
0	12398.842019	12523.345901	-2.642500
1	12400.560271	12524.552338	2.112181

Gambar IV.24 Kode dan hasil Regresi dari masukan pengguna

Model regresi yang telah dibuat dan disimpan dalam suatu file, dimuat pada program sehingga tidak perlu menghabiskan waktu untuk melakukan pelatihan model lagi. Selanjutnya model akan mengolah masukan, dan mengeluarkan prediksi label seperti yang ditampilkan pada gambar. Penting untuk memilih model terbaik sehingga hasilnya dapat dipercaya untuk merepresentasikan kondisi sesungguhnya. Nilai MTBM dimasukan oleh pengguna berdasarkan opsi yang digunakan saat melatih model regresi. Perlu diperhatikan saat input nilai fitur sangat besar dan merupakan Outlier, model OLS dan Ridge juga menunjukan hasil yang abnormal, sedangkan model RF-reg masih dapat dikatakan normal. Ini menjadi salah satu keunggulan model RF-reg karena cara kerjanya yang berbasis pohon keputusan dan bukan fungsi linear. Selain itu juga mungkin dikarenakan data fitur tidak dilakukan *StandardScaler* pada algoritma OLS dan Ridge, sehingga hasilnya sensitif dengan nilai yang abnormal.

V.3 Menentukan opsi terbaik Klasifikasi

Kriteria untuk menentukan opsi terbaik berdasarkan 3 kriteria dari sisi keperluan untuk pengguna. *Training Dataset Size* menyatakan banyak data serta ketersediaan data berlabel yang digunakan saat melatih model untuk opsi tersebut, ini memiliki korelasi dengan Runtime pada saat pelatihan model. *Scalability* menilai kemudahan opsi jika ditingkatkan penggunaan fiturnya untuk melakukan prediksi. *Label Clarity* menilai kemudahan menjelaskan hubungan antara label dengan kondisi kebutuhan perawatan. Data ini akan ditabulasi dan diberi nilai untuk menentukan mana yang akan dipilih untuk diterapkan dalam finalisasi desain sistem perawatan prediktif. Untuk sistem penilaian akan diberi poin dari 1 (terendah) hingga 4 (tertinggi), kemudian dilihat total skor dan akan dipilih yang memiliki total terbanyak.

Tabel V.1 *Decision matrix* opsi penerapan prediksi klasifikasi

Criteria \ Alternative	Opsi 1 Outlier + SMOTE	Opsi 2 No Outlier + SMOTE	Opsi 3 DBSCAN + Classifier	Opsi 4 Kmeans + Classifier

Training Dataset Size	3	1	4	4
Scalability	1	1	2	2
Label Clarity	4	3	1	1
Total	8	5	7	7
Ranking	1	4	2	2

Opsi 1 menggunakan data pelatihan sebanyak 18165 sampel setelah dilakukan *oversampling* dengan SMOTE, merupakan urutan ketiga terbanyak dibandingkan opsi lain. Karena menggunakan pendekatan statistic untuk memberi label pada dataset, menyebabkan perlunya untuk menambah variabel secara manual bagi setiap fitur baru yang akan digunakan, selain itu nilai *threshold* mungkin perlu diubah seiring dengan semakin banyaknya data yang disimpan, sehingga diberi skor terendah. Sementara untuk pemberian label, sangat mudah dimengerti hubungan kondisi kereta dengan data sensor yang didapat. Berdasarkan penilaian tersebut, opsi 1 memiliki nilai tertinggi untuk sistem PdM di penelitian ini.

Opsi 2 tampak memiliki performa yang paling baik jika dilihat dari hasil performa model, namun pada opsi ini hanya dilatih menggunakan 595 data meskipun telah dilakukan *oversampling*, sehingga jumlahnya paling rendah dibandingkan opsi lain. Untuk skalabilitas memiliki masalah yang sama dengan opsi 1, yaitu perlunya untuk menambah variabel secara manual bagi setiap fitur baru yang akan digunakan, selain itu nilai *threshold* mungkin perlu diubah seiring dengan semakin banyaknya data yang disimpan. Sementara untuk pemberian label, sangat mudah dimengerti hubungan kondisi kereta dengan data sensor yang didapat, tetapi hanya menggunakan 2 label untuk klasifikasi.

Opsi 3 menggunakan data pelatihan sebanyak 66225 sampel setelah dilakukan *oversampling* dengan SMOTE, merupakan jumlah terbanyak dibandingkan opsi lain. Karena menggunakan pendekatan algoritma clustering untuk memberi label pada dataset, menyebabkan tidak perlu untuk menambah variabel bagi setiap fitur baru yang akan digunakan, hanya perlu mengatur *hyperparameter* untuk menyesuaikan. Sementara untuk pemberian label sulit dimengerti hubungan

kondisi kereta dengan data sensor yang didapat, sehingga perlu dipelajari lebih lanjut hasil clusternya. Keunggulan dengan DBSCAN yaitu kemampuan nya untuk memisahkan data noise dari cluster jika jumlahnya tidak terlalu banyak, sehingga tidak perlu membuat label khusus untuk data tersebut.

Opsi 4 menggunakan data pelatihan sebanyak 66225 sampel setelah dilakukan *oversampling* dengan SMOTE, sama jumlahnya dengan opsi 3 sehingga memiliki skor yang sama. Begitupun untuk kriteria lainnya, karena menggunakan pendekatan algoritma clustering untuk memberi label pada dataset, menyebabkan tidak perlu untuk menambah variabel bagi setiap fitur baru yang akan digunakan, hanya perlu mengatur *hyperparameter* untuk menyesuaikan. Sementara untuk pemberian label sulit dimengerti hubungan kondisi kereta dengan data sensor yang didapat, sehingga perlu dipelajari lebih lanjut hasil clusternya. Keunggulan dari KMeans adalah kemampuan untuk inisialisasi jumlah cluster yang diinginkan, sehingga dapat lebih mudah memberikan label jika mengetahui kondisi data.

V.4 Menentukan opsi terbaik Regresi

Kriteria untuk menentukan opsi terbaik ditentukan berdasarkan 3 kriteria dari sisi keperluan untuk pengguna. *Training Dataset Size* menyatakan banyak data serta ketersediaan data berlabel yang digunakan saat melatih model untuk opsi tersebut, ini memiliki korelasi dengan Runtime pada saat pelatihan model. *Scalability* menilai kemudahan opsi jika ditingkatkan penggunaan fiturnya untuk melakukan prediksi. *MAE Result* menilai hasil yang didapatkan saat tahap pelatihan dan pengujian. Data ini akan ditabulasi dan diberi nilai untuk menentukan mana yang akan dipilih untuk diterapkan dalam finalisasi desain sistem perawatan prediktif. Untuk sistem penilaian akan diberi poin dari 1 (terendah) hingga 5 (tertinggi), kemudian dilihat total skor dan akan dipilih yang memiliki total terbanyak.

Tabel V.2 *Decision matrix* opsi penerapan prediksi RUL

Criteria \ Alternative	Opsi 1 All Vehicles (OR)	Opsi 2 All Vehicles (AND)	Opsi 3 Single Vehicles (OR)	Opsi 4 ML Label (OR)	Opsi 5 All Vehicles (OR), Grouped
Dataset Size	4	1	2	3	4

Scalability	3	1	2	5	3
MAE Result	5	1	3	4	3
Total	12	3	7	12	10
Ranking	1	5	4	1	3

Opsi 1 menggunakan data total sebanyak 6148 sampel, yang merupakan jumlah paling banyak dari semua opsi sehingga mendapatkan nilai yang cukup tinggi. Namun karena ada opsi lain yang memiliki jumlah data lebih sedikit dan mendapat hasil yang hampir sama, menyebabkan tidak mendapat nilai tertinggi. Dalam melatih model, semakin banyak data akan membuat hasil lebih akurat namun apabila terlalu banyak maka akan menambah waktu pemrosesan. Untuk kemampuan skalabilitas, diberi nilai sedang karena menggunakan kondisi statistik untuk memberi label pada dataset, menyebabkan perlunya untuk menambah variabel secara manual bagi setiap fitur baru yang akan digunakan, selain itu nilai *threshold* mungkin perlu diubah seiring dengan semakin banyaknya data yang disimpan. Sementara untuk hasil MAE, sudah berhasil mencapai target yaitu kurang dari 10 bagi semua algoritma model, sehingga mendapat nilai tertinggi. Berdasarkan penilaian keseluruhan, opsi ini seimbang dengan opsi 4 sebagai alternatif terbaik untuk prediksi RUL.

Opsi 2 memiliki total 56 data dan memiliki hasil MAE yang sangat buruk, sehingga mendapat nilai terendah untuk keduanya. Pada skalabilitas memiliki masalah yang sama dengan opsi 1, ditambah juga karena menggunakan kondisi AND pada setiap fitur untuk memenuhi kondisi Maintenance, maka jumlah data akan semakin sedikit karena peluang kondisi terpenuhi mengecil.

Opsi 3 menggunakan data sebanyak 426 sampel, dengan hasil MAE yang masih kurang memuaskan. Untuk skalabilitas memiliki masalah yang sama dengan opsi 1, ditambah dengan perlunya memisahkan setiap kode kereta menyebabkan perlunya tambahan baris kode seiring dengan meningkatnya jumlah data dari kereta lain.

Opsi 4 menggunakan data sebanyak 2778 sampel, dan memberikan hasil MAE yang telah memenuhi target saat menggunakan data uji. Sementara untuk skalabilitas

mendapatkan nilai tertinggi, dikarenakan opsi ini unggul dengan berbasis hasil klasifikasi yang didapat dari model ML, sehingga tidak banyak penyesuaian yang diperlukan seiring dengan bertambahnya fitur maupun jumlah data. Berdasarkan penilaian keseluruhan, opsi ini seimbang dengan opsi 1 sebagai alternatif terbaik untuk prediksi RUL.

Opsi 5 menggunakan data total sebanyak 6148 sampel, yang merupakan jumlah terbanyak beserta dengan opsi 1 sehingga mendapat skor seimbang antar keduanya. Meski demikian, hasil MAE masih belum memenuhi target walaupun tidak terlalu buruk. Pada skalabilitas memiliki masalah yang sama dengan opsi 1, sehingga mendapatkan skor yang seimbang.

V.5 Menentukan algoritma terbaik Klasifikasi

Hasil terbaik yang didapatkan dari keseluruhan opsi dalam pengujian model ML akan dipertimbangkan mana yang cocok untuk dipilih dengan membandingkan kriteria suksesnya. Data ini akan ditabulasi dan diberi nilai untuk menentukan mana yang akan dipilih untuk diterapkan dalam finalisasi desain sistem perawatan prediktif. Untuk sistem penilaian akan diberi poin dari 1 (terendah) hingga 7 (tertinggi) berdasarkan performa, kemudian dilihat total skor dan akan dipilih yang memiliki total terbanyak.

Tabel V.3 *Decision matrix* algoritma model ML terbaik

Criteria \ Algorithm	LR	KNN	DT	SVM	RF	LDA	NB
Accuracy	2	5	6	3	6	1	4
Runtime	5	1	4	2	3	6	7
Accuracy/Runtime	4	1	5	2	3	6	7
Total	11	7	15	7	12	13	18
Ranking	5	6	2	6	4	3	1

Nilai tersebut berdasarkan hasil terbaik model yang didapatkan dari keseluruhan opsi, perlu dipertimbangkan lebih lanjut opsi mana yang akan digunakan sebelum menentukan penggunaan Algoritma. Secara keseluruhan untuk algoritma NB memiliki keunggulan pada Akurasi/Runtime, sementara DT dan RF unggul pada Akurasi, sedangkan LDA dan NB memiliki Runtime pemrosesan yang paling cepat.

Jika mengacu pada opsi 1 yang merupakan opsi terbaik saat pembahasan sebelumnya, maka disarankan untuk menghindari LR dan LDA karena performanya yang buruk untuk opsi tersebut. Dalam konteks penelitian ini, dipilih DT sebagai algoritma terbaik karena model ML hanya perlu melakukan prediksi berdasarkan satu input dari pengguna sehingga lebih diutamakan Akurasi daripada Runtime.

V.6 Menentukan algoritma terbaik Regresi

Hasil terbaik yang didapatkan dari keseluruhan opsi dalam pengujian model ML akan dipertimbangkan mana yang cocok untuk dipilih dengan membandingkan kriteria suksesnya dan hasil pengujian. Data ini akan ditabulasi dan diberi nilai untuk menentukan mana yang akan dipilih untuk diterapkan dalam finalisasi desain sistem perawatan prediktif. Untuk sistem penilaian akan diberi poin dari 1 (terendah) hingga 3 (tertinggi) berdasarkan performa, kemudian dilihat total skor dan akan dipilih yang memiliki total terbanyak.

Tabel V.4 *Decision matrix* algoritma model ML regresi terbaik

Criteria \ Algorithm	OLS	Ridge	RF-reg
MAE	3	2	1
Runtime	2	3	1
MAE/Runtime	3	2	1
Total	8	7	3
Ranking	1	2	3

Nilai tersebut berdasarkan hasil terbaik model yang didapatkan dari keseluruhan opsi, perlu dipertimbangkan lebih lanjut opsi mana yang akan digunakan sebelum menentukan penggunaan Algoritma. Hasil model OLS dan Ridge cukup dekat diantara semua opsi, dan setelah dilakukan penilaian memiliki skor total yang sama. Perhatian khusus perlu diberikan pada model RF-reg untuk kemampuan prediksinya, seperti yang ditampilkan saat pengujian menggunakan masukan file bahwa model ini dapat memberikan estimasi yang cukup normal meskipun nilai fitur merupakan Outlier. Dalam konteks penelitian ini, dipilih OLS sebagai algoritma terbaik karena model ML lebih diutamakan MAE daripada Runtime.

BAB VI Penutup

VI.1 Kesimpulan

Berikut ini adalah kesimpulan yang didapatkan dari penelitian yang telah dilakukan dalam pengerjaan tesis.

1. Proses bisnis perawatan sarana dari KAI untuk saat ini telah dibuat alur kerjanya dengan bantuan program *Enterprise Resource Management* berupa SAP. Meski demikian, dalam pelaksanaan untuk mengecek kondisi sarana masih dilakukan manual berdasarkan *checksheet* dari kertas dan hasilnya belum disimpan dalam bentuk digital. Lokasi perbaikan ringan yang berbasis di depo masing-masing tempat kereta api beroperasi menyebabkan data yang tersedia tersebar dan sulit diakses secara menyeluruh, sementara itu balai yasa hanya melayani perawatan besar (*overhaul*) sehingga tidak memiliki data inspeksi rutin.

2. Penelitian ini menghasilkan model ML untuk mendeteksi kemungkinan keperluan perawatan berdasarkan 4 fitur dari nilai sensor (Oil_Pressure, Coolant_Temp, PF_Avg, ECU_Temp) pada kereta pembangkit yang mewakilkan kondisi komponen pelumas dan pendingin. Proses mendesain model tersebut meliputi metode menentukan opsi paling tepat, optimasi performa, hingga pengujian hasil dalam program sederhana yang dapat menentukan Klasifikasi dan RUL dari masukan secara langsung oleh pengguna.

3. Berbagai opsi untuk penerapan sistem PdM telah diuji pada penelitian ini, dengan 4 opsi berbeda pada model Klasifikasi. Untuk menyeimbangkan jumlah data untuk setiap label, keseluruhan opsi menggunakan SMOTE. Opsi 1 menggunakan pendekatan statistik untuk memberi label, dengan 3 label (Outlier, Normal, Maintenance). Opsi 2 juga menggunakan pendekatan statistik untuk memberi label, tetapi hanya dengan 2 label (Normal, Maintenance). Opsi 3 menggunakan DBSCAN untuk memberikan label pada dataset. Sementara Opsi 4 menggunakan Kmeans untuk memberikan label pada dataset. Berdasarkan penilaian menggunakan *decision matrix* didapatkan bahwa Opsi 1 menjadi solusi terbaik.

Sementara untuk model Regresi, Opsi 1 menggunakan pendekatan statistik untuk memberi label, dengan kondisi OR antara setiap fitur. Opsi 2 juga menggunakan pendekatan statistik untuk memberi label, tetapi dengan kondisi AND pada setiap fitur. Opsi 3 menggunakan pendekatan statistik untuk memberi label, tapi hanya diambil data dari satu kereta saja. Sementara Opsi 4 menggunakan hasil model ML klasifikasi untuk memberikan label pada dataset. Terakhir pada Opsi 5, juga menggunakan pendekatan statistik untuk memberi label, namun data dikelompokkan berdasarkan kode kereta. Berdasarkan penilaian menggunakan *decision matrix* didapatkan bahwa Opsi 1 dan Opsi 4 memiliki skor seimbang untuk menjadi solusi terbaik.

4. Secara keseluruhan untuk model Klasifikasi algoritma NB memiliki keunggulan pada Akurasi/Runtime sehingga cocok digunakan jika mengolah data yang banyak. Sementara DT dan RF unggul pada Akurasi, namun memiliki Runtime yang cukup lama. Sedangkan LDA dan NB memiliki Runtime pemrosesan yang paling cepat. Untuk detail nilai yang didapatkan, dapat melihat Tabel IV.6. Dalam konteks penelitian ini, dipilih DT sebagai algoritma terbaik karena model ML hanya perlu melakukan prediksi berdasarkan satu input dari pengguna sehingga lebih diutamakan Akurasi daripada Runtime.

   Sementara untuk model Regresi, algoritma OLS unggul pada MAE meskipun algoritma Ridge cukup dekat perbedaannya. Begitupun dengan Runtime bagi kedua algoritma tersebut tidak berbeda jauh. Sedangkan RF-reg memiliki MAE dan Runtime yang kurang memuaskan, meskipun demikian perhatian khusus perlu diberikan karena model ini dapat memberikan estimasi yang cukup normal meskipun nilai fitur merupakan Outlier. Untuk detail nilai yang didapatkan, dapat melihat Tabel IV.4. Dalam konteks penelitian ini, dipilih OLS sebagai algoritma terbaik karena model ML Regresi lebih diutamakan MAE daripada Runtime.

VI.2 Saran

Berikut ini adalah saran yang diberikan untuk pengembangan lanjutan dari penelitian ini.

1. Dibutuhkan data yang sudah memiliki label untuk mendukung data sensor. Contohnya seperti hasil perawatan Harian dan Bulanan, terutama saat terjadi perbaikan pada komponen tertentu. Sebaiknya sudah tersedia dalam bentuk digital untuk lebih mudah diolah menjadi dataset yang dapat dipercaya untuk digunakan pada ML dalam sistem PdM.

2. Penambahan fitur yang digunakan sebagai basis prediksi juga diperlukan untuk dapat meningkatkan ketepatan dan melakukan monitor kondisi komponen lannya pada sarana KAI.

3. Pengembangan aplikasi dalam penelitian ini harus dilakukan secara lebih lanjut untuk dapat memberikan manfaat nyata, yakni meningkatkan ketersediaan sarana KAI dalam beroperasi. Contohnya adalah membuat antarmuka untuk pengguna, ataupun *Application Programming Interface* (API) untuk dapat mengambil data sensor secara otomatis.

4. Perlu memastikan bahwa dataset yang telah digunakan dalam model terus diperbarui seiring dengan bertambahnya data yang terkumpul, kemudian melakukan pelatihan ulang model, sehingga ketepatan model tidak menurun dari masa ke masa.

5. Melakukan uji banding antara hasil prediksi model ML dengan kondisi nyata saat pengamatan langsung dilapangan. Kemudian memperbaiki model berdasarkan hasil yang didapatkan, sehingga model ML menjadi lebih baik lagi dalam melakukan prediksinya.

DAFTAR PUSTAKA

[1] Menteri Perhubungan Republik Indonesia, *PM 153 Tentang Standar Spesifikasi Teknis Lokomotif Sarana Perkeretaapian,* Jakarta: Menteri Perhubungan Republik Indonesia, 2016.

[2] Menteri Perhubungan Republik Indonesia, *PM 30 Tentang Tata Cara Pengujian dan Pemberian Sertifikat Prasarana Perkeretaapian,* Jakarta: Menteri Perhubungan Republik Indonesia, 2011.

[3] Menteri Perhubungan Republik Indonesia, *PM 15 Tentang Standar, Tata Cara Pengujian dan Sertifikasi Kelaikan,* Jakarta: Menteri Perhubungan Republik Indonesia, 2011.

[4] Menteri Perhubungan Republik Indonesia, *KM 41 Tentang Standar Spesifikasi Teknis Kereta Yang Ditarik Lokomotif,* Jakarta: Menteri Perhubungan Republik Indonesia, 2010.

[5] Menteri Perhubungan Republik Indonesia, *KM 43 Tahun 2010 Tentang Standar Spesifikasi Teknis Gerbong,* Jakarta: Menteri Perhubungan Republik Indonesia, 2010.

[6] Menteri Perhubungan Republik Indonesia, *PM 18 Tentang Standar Tempat dan Peralatan Perawatan Sarana Perkeretaapian,* Jakarta: Menteri Perhubungan Republik Indonesia, 2019.

[7] W. Septiani, D. Suhardini and E. Sari, "PENGUKURAN KINERJA PERAWATAN LOKOMOTIF PT. KERETA API INDONESIA (PERSERO) BERDASARKAN MODEL MAINTENANCE SCORECARD," *J@ti Undip : Jurnal Teknik Industri,* vol. 7, no. 3, pp. 191-198, 2013.

[8] D. Burrough, "Digital transformation improves SNCF’s maintenance systems," International Railway Journal, 25 06 2020. [Online]. Available: https://www.railjournal.com/in_depth/digital-transformation-improves-sncfs-maintenance-systems. [Accessed 29 09 2021].

[9] The British Standards Institution, *Maintenance - Maintenance terminology,* BSI Standards, 2018.

[10] D. Jeffrey, Principles of Machine Operation and Maintenance, Oxford: Butterworth Heinemann Ltd, 1991.

[11] B. Schmidt and L. Wang, "Cloud-enhanced predictive maintenance," *International Journal Advanced Manufacturing Technology,* 2016.

[12] Y. Ran, X. Zhou, P. Lin, Y. Wen and R. Deng, "A Survey of Predictive Maintenance: Systems,," *IEEE COMMUNICATIONS SURVEYS & TUTORIALS,* 2019.

[13] D. Justus, J. Brennan, S. Bonner and A. S. McGough, "Predicting the Computational Cost of Deep Learning Models," in *IEEE International Conference on Big Data*, 2018.

[14] C. Krupitzer, T. Wagenhals, M. Zufle, V. Lesch, D. Schafer, A. Mozaffarin, J. Edinger, C. Becker and S. Kounev, "A Survey on Predictive Maintenance for Industry 4.0," 2020. [Online]. Available: https://www.semanticscholar.org/paper/A-Survey-on-Predictive-Maintenance-for-Industry-4.0-Krupitzer-Wagenhals/ef0cec144f3132f47f3a3e39b96aa8067bee90af. [Accessed 1 December 2020].

[15] M. C. Garcia, M. A. Sanz-Bobi and J. d. Pico, "SIMAP: Intelligent System for Predictive Maintenance Application to the health condition monitoring of a windturbine gearbox," *Computers in Industry,* pp. 552-568, 2006.

[16] K. A. Kaiser and N. Z. Gebraeel, "Predictive Maintenance Management Using Sensor-Based Degradation Models," *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS,* vol. 39, no. 4, p. 840, 2009.

[17] W. Zhang, D. Yang and H. Wang, "Data-Driven Methods for Predictive Maintenance of Industrial Equipment: A Survey," *IEEE SYSTEMS JOURNAL,* vol. 13, no. 3, pp. 2213-2227, 2019.

[18] MathWorks, "Designing Algorithms for Condition Monitoring and Predictive Maintenance," [Online]. Available: https://www.mathworks.com/help/predmaint/gs/designing-algorithms-for-condition-monitoring-and-predictive-maintenance.html?s_eid=PSM_15028. [Accessed 11 10 2021].

[19] MathWorks, "Decision Models for Fault Detection and Diagnosis," [Online]. Available: https://www.mathworks.com/help/predmaint/ug/decision-models-for-fault-detection-and-diagnosis.html. [Accessed 12 10 2021].

[20] MathWorks, "RUL Estimation Using RUL Estimator Models," [Online]. Available: https://www.mathworks.com/help/predmaint/ug/rul-estimation-using-rul-estimator-models.html. [Accessed 11 10 2021].

[21] K. Schwab, The Fourth Industrial Revolution, New York: Crown Publishing Group, 2016.

[22] S. J. Russel and P. Norvig, Artificial Intelligence: A Modern Approach (3rd Ed), New Jersey: Prentice Hall, 2009.

[23] E. Alpaydin, Introduction to Machine Learning (Second ed.), London: The MIT Press, 2010.

[24] A. Geron, Hands-On Machine Learning with Scikit-Learn & TensorFlow, California: O'Reilly Media, Inc, 2017.

[25] A. Barnes, "Increased Locomotive performance using Condition Based Maintenance," in *6th IET Conference on Railway Condition Monitoring (RCM 2014)*, Birmingham, 2014.

[26] T. Y. En, T. J. Jie, M. S. Ki, N. T. Hui and M. Ashrof, "Predictive Maintenance of a Train System Using a Multilayer Perceptron Artificial Neural Network," in *2018 International Conference on Intelligent Rail Transportation (ICIRT),* Singapore, 2018.

[27] Q. Wang, S. Bu and Z. He, "Achieving Predictive and Proactive Maintenance for High-Speed Railway Power Equipment with LSTM-RNN," *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS,* 2020.

[28] N. N. Hakim, Pengembangan Model Machine Learning untuk Analisis Prediksi Kemungkinan Kejadian Patah Rel Kereta Api di Indonesia, Bandung: ITB, 2020.

[29] R. Nappi, G. Cutrera, A. Vigliotti and G. Franzè, "A Predictive-based Maintenance Approach for Rolling Stocks Vehicles," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Vienna, 2020.

[30] A. Chakrabarti and L. T. Blessing, DRM, a Design Research Methodology, London: Springer, 2009.

[31] Pedregosa and et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

LAMPIRAN

1. Hasil Pelatihan Model

Iteration	Option	Success Criteria	Algorithm							Highest Value	
Initial			LR	k-NN	DT	SVM	RF	LDA	NB		Lowest Value
1	Statistical label, Outlier, Random State=1	Accuracy	0.946225	0.999531	1	0.946538	1	0.946538	0.973738	1	0.946225
		Runtime	11.9	11.9	11.9	11.9	11.9	11.9	11.9	11.9	11.9
2	Statistical label, No Outlier, Random State=1	Accuracy	0.985294	0.994118	1	0.868487	1	0.985294	1	1	0.868487
		Runtime	2.05	2.05	2.05	2.05	2.05	2.05	2.05	2.05	2.05
3	DBSCAN, Random State=1	Accuracy	0.936704	0.999927	1	0.983846	0.999964	0.99447	0.999127	1	0.936704
		Runtime	29.6	29.6	29.6	29.6	29.6	29.6	29.6	29.6	29.6
4	Kmeans, Random State=1	Accuracy	0.999818	0.999891	0.999927	0.984246	0.999927	0.99396	0.9992	0.999927	0.984246
		Runtime	30.9	30.9	30.9	30.9	30.9	30.9	30.9	30.9	30.9
Avg Accuracy			0.9670102 5	0.99836675	0.99998175	0.94577925	0.99997275	0.9800655	0.9930162 5	0.99998175	0.94577925
Avg Runtime			18.6125	18.6125	18.6125	18.6125	18.6125	18.6125	18.6125	18.6125	18.6125
Acc/Time			5.1954882 47	5.36395836 1	5.372635326	5.08141974 5	5.37258697 1	5.265630625	5.3352115 51	5.37263532 6	5.08141974 5
Normalized Acc/Time			0.9670278 983	0.99838497 05	1	0.94579651 08	0.99999099 98	0.980083386 5	0.9930343 729	1	0.94579651 08
Tune Hyperparameter			LR(max_iter=475)			SVC(gamma='auto')					
1		Accuracy	0.946538	N/A	N/A	0.999531	N/A	N/A	N/A	0.999531	0.946538

	Statistical label, Outlier, Random State=1	Runtime	28.3	N/A	N/A	28.3	N/A	N/A	N/A	28.3	28.3
2	Statistical label, No Outlier, Random State=1	Accuracy	0.985294	N/A	N/A	0.991176	N/A	N/A	N/A	0.991176	0.985294
		Runtime (s)	2.16	N/A	N/A	2.16	N/A	N/A	N/A	2.16	2.16
Avg Accuracy			0.965916	N/A	N/A	0.9953535	N/A	N/A	N/A	0.9953535	0.965916
Avg Runtime			15.23	N/A	N/A	15.23	N/A	N/A	N/A	15.23	15.23
Acc/Time			6.34219304	N/A	N/A	6.535479317	N/A	N/A	N/A	6.535479317	6.34219304
Normalized Acc/Time			0.416427645	N/A	N/A	0.4291187995	N/A	N/A	N/A	0.4291187995	0.4164276454
Use SMOTE			LR(max_iter=475)			SVC(gamma='auto')					
1	Statistical label, Outlier, Random State=1, SMOTE	Accuracy	0.627856	0.999945	1	1	1	0.40011	0.989871	1	0.40011
		Runtime	42.4	42.4	42.4	42.4	42.4	42.4	42.4	42.4	42.4
2	Statistical label, No Outlier, Random State=1, SMOTE	Accuracy	0.99	1	1	1	1	0.989915	1	1	0.989915
		Runtime	2.24	2.24	2.24	2.24	2.24	2.24	2.24	2.24	2.24
3	DBSCAN, Random State=1, SMOTE	Accuracy	1	1	1	0.9964	1	0.9508	0.9987	1	0.9508
		Runtime (s)	1232	1232	1232	1232	1232	1232	1232	1232	1232
4	Kmeans, Random State=1, SMOTE	Accuracy	1	1	1	0.996391	1	0.950804	0.998671	1	0.950804
		Runtime (s)	1185	1185	1185	1185	1185	1185	1185	1185	1185
Avg Accuracy			0.904464	0.99998625	1	0.99819775	1	0.82290725	0.9968105	1	0.82290725

Avg Runtime (s)	615.41	615.41	615.41	615.41	615.41	615.41	615.41	615.41	615.41
Acc/Time	1.469693375	1.624910629	1.624932972	1.622004436	1.624932972	1.337169123	1.619750248	1.624932972	1.337169123
Normalized Acc/Time	0.904464	0.99998625	1	0.99819775	1	0.82290725	0.9968105	1	0.82290725

2. Hasil Pengujian model

Iteration	Methods	Success Criteria	Algorithm							Highest Value	Lowest Value	Average
Initial			LR	k-NN	DT	SVM	RF	LDA	NB			
0	Statistical label, Outlier, Random State=1	Accuracy	0.05	0.05125	0.05375	0.046875	0.05375	0.0525	0.05375	0.05375	0.046875	0.05169642857
		Precision	0.15	0.31	0.08	0.02	0.08	0.13	0.13	0.31	0.02	0.1285714286
		Recall	0.48	0.55	0.67	0.33	0.67	0.61	0.67	0.67	0.33	0.5685714286
		F1 Score	0.2	0.33	0.13	0.03	0.13	0.19	0.19	0.33	0.03	0.1714285714
		Runtime (ms)	298	416	267	277	329	262	261	416	261	301.4285714
	Statistical label, No Outlier, Random State=1	Accuracy	0.9302325581	0.9534883721	1	0.8720930233	1	0.976744186	1	1	0.8720930233	0.9617940199
		Precision	0.96	0.97	1	0.44	1	0.99	1	1	0.44	0.9085714286
		Recall	0.73	0.82	1	0.5	1	0.91	1	1	0.5	0.8514285714
		F1 Score	0.79	0.88	1	0.47	1	0.94	1	1	0.47	0.8685714286
		Runtime (ms)	233	231	216	369	252	269	222	369	216	256
	DBSCAN, Random State=1	Accuracy	1	0.999854482	0.99985	0.98486	0.99985	0.99388	0.99912	1	0.98486	0.9967734974
		Precision	1	1	1	0.97	1	1	0.99	1	0.97	0.9942857143

		Recall	1	1	1	0.87	1	0.95	1	1	0.87	0.9742857143
		F1 Score	1	1	1	0.9	1	0.97	1	1	0.9	0.9814285714
		Runtime (ms)	214	675	256	459	232	162	186	675	162	312
	Kmeans, Random State=1	Accuracy	0.9944703143	1	1	0.9832654249	1	0.9959254948	0.9989813737	1	0.9832654249	0.9960918011
		Precision	0.96	1	1	0.97	1	1	0.99	1	0.96	0.9885714286
		Recall	0.99	1	1	0.85	1	0.96	1	1	0.85	0.9714285714
		F1 Score	0.98	1	1	0.89	1	0.98	0.99	1	0.89	0.9771428571
		Runtime (ms)	215	795	283	497	298	191	195	795	191	353.4285714
Avg Accuracy			0.7436757181	0.7511482135	0.7634	0.721773362	0.7634	0.7547624202	0.7629628434	0.7634	0.721773362	0.7515889368
Avg Precision			0.7675	0.82	0.77	0.6	0.77	0.78	0.7775	0.82	0.6	0.755
Avg Recall			0.8	0.8425	0.9175	0.6375	0.9175	0.8575	0.9175	0.9175	0.6375	0.8414285714
Avg F1 Score			0.7425	0.8025	0.7825	0.5725	0.7825	0.77	0.795	0.8025	0.5725	0.7496428571
Avg Runtime			240	529.25	255.5	400.5	277.75	221	216	529.25	216	305.7142857
Acc/Time			3.098648825	1.41926918	2.987866928	1.802180679	2.748514851	3.415214571	3.532235386	3.532235386	1.41926918	2.714847203
Normalized Acc/Time			0.8772486787	0.4018048133	0.8458855656	0.5102096781	0.7781233556	0.9668706067	1	1	0.4018048133	0.768591814

Tune Hyperparameter			LR(max_iter=475)			SVC(gamma='auto')						
1	Statistical label, Outlier, Random State=1	Accuracy	0.05	-	-	0.0525	-	-	-	0.0525	0.05	0.05125
		Precision	0.15	-	-	0.35	-	-	-	0.35	0.15	0.25
		Recall	0.48	-	-	0.61	-	-	-	0.61	0.48	0.545
		F1 Score	0.2	-	-	0.33	-	-	-	0.33	0.2	0.265
		Runtime (ms)	263	-	-	275	-	-	-	275	263	269
	Statistical label, No Outlier, Random State=1	Accuracy	0.9302325581	-	-	0.976744186	-	-	-	0.976744186	0.930232558	0.9534883721
		Precision	0.96	-	-	0.99	-	-	-	0.99	0.96	0.975
		Recall	0.73	-	-	0.91	-	-	-	0.91	0.73	0.82
		F1 Score	0.79	-	-	0.94	-	-	-	0.94	0.79	0.865
		Runtime (ms)	218	-	-	330	-	-	-	330	218	274
Avg Accuracy			0.4901162791	-	-	0.514622093	-	-	-	0.514622093	0.490116279	0.502369186
Avg Precision			0.555	-	-	0.67	-	-	-	0.67	0.555	0.6125
Avg Recall			0.605	-	-	0.76	-	-	-	0.76	0.605	0.6825
Avg F1 Score			0.495	-	-	0.635	-	-	-	0.635	0.495	0.565
Avg Runtime			240.5	-	-	302.5	-	-	-	302.5	240.5	271.5
Acc/Time			2.037905526	-	-	1.70123006	-	-	-	2.037905526	1.70123006	1.869567793

Normalized Acc/Time			1	-	-	0.8347933885	-	-	-	1	0.8347933885	0.9173966943
Use SMOTE			LR(max_iter=475)			SVC(gamma='auto')						
2	Statistical label, Outlier, Random State=1, SMOTE	Accuracy	0.6188903567	0.99977983 27	1	1	1	0.38353	0.99207	1	0.38353	0.8563243128
		Precision	0.49	1	1	1	1	0.45	0.99	1	0.45	0.8471428571
		Recall	0.63	1	1	1	1	0.4	0.99	1	0.4	0.86
		F1 Score	0.52	1	1	1	1	0.28	0.99	1	0.28	0.8271428571
		Runtime (ms)	369	767	339	596	431	320	290	767	290	444.5714286
	Statistical label, No Outlier, Random State=1, SMOTE	Accuracy	1	1	1	1	1	0.99328	1	1	0.99328	0.99904
		Precision	1	1	1	1	1	0.99	1	1	0.99	0.9985714286
		Recall	1	1	1	1	1	0.99	1	1	0.99	0.9985714286
		F1 Score	1	1	1	1	1	0.99	1	1	0.99	0.9985714286
		Runtime (ms)	207	211	203	329	243	218	213	329	203	232
	DBSCAN, Random State=1, SMOTE	Accuracy	1	0.9999	0.99985	0.99728	0.99985	0.95216	0.99836	1	0.95216	0.9924857143
		Precision	1	1	1	1	1	0.96	1	1	0.96	0.9942857143
		Recall	1	1	1	1	1	0.95	1	1	0.95	0.9928571429

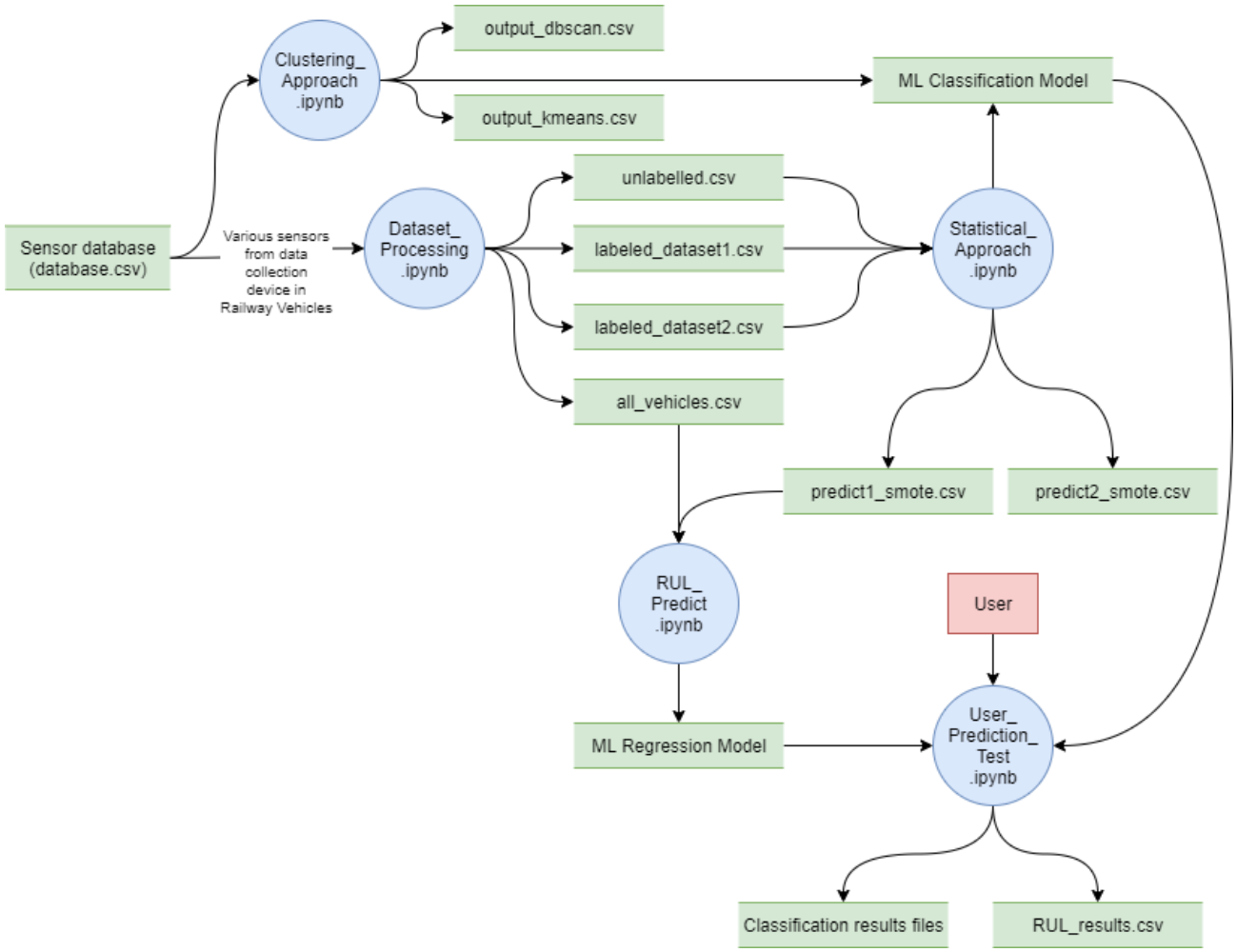
		F1 Score	1	1	1	1	1	0.95	1	1	0.95	0.9928571429
		Runtime (ms)	305	2340	703	6370	479	244	264	6370	244	1529.285714
	Kmeans, Random State=1, SMOTE	Accuracy	1	0.99993	1	0.99728	1	0.95216	0.99836	1	0.95216	0.9925328571
		Precision	1	1	1	1	1	0.96	1	1	0.96	0.9942857143
		Recall	1	1	1	1	1	0.95	1	1	0.95	0.9928571429
		F1 Score	1	1	1	1	1	0.95	1	1	0.95	0.9928571429
		Runtime (ms)	268	1840	351	5830	500	278	255	5830	255	1331.714286
Avg Accuracy			0.9047225892	0.9999024582	0.9999625	0.99864	0.9999625	0.8202825	0.9971975	0.9999625	0.8202825	0.960095721
Avg Precision			0.8725	1	1	1	1	0.84	0.9975	1	0.84	0.9585714286
Avg Recall			0.9075	1	1	1	1	0.8225	0.9975	1	0.8225	0.9610714286
Avg F1 Score			0.88	1	1	1	1	0.7925	0.9975	1	0.7925	0.9528571429
Avg Runtime			287.25	1289.5	399	3281.25	413.25	265	255.5	3281.25	255.5	884.3928571
Acc/Time			3.149599962	0.7754187345	2.506171679	0.3043474286	2.419751966	3.09540566	3.902925636	3.902925636	0.3043474286	2.307660152
Normalized Acc/Time			0.806984364	0.1986762769	0.6421264233	0.077979305	0.6199841329	0.7930988056	1	1	0.077979305	0.5912641868

3. Hasil Prediksi Input Pengguna

No		Input				Model Prediction Option 1						
		Oil_Pressure	Coolant_Temp	PF_Avg	ECU_Temp	LR	KNN	DT	SVM	RF	LDA	NB
1	Value	0	61	361	61							
	Status	Outlier	Outlier	Outlier	Outlier	Outlier	Outlier	Outlier	Outlier	Outlier	Normal	Outlier
2	Value	613	80	327	32767							
	Status	Normal	Normal	Outlier	Outlier	Outlier	Outlier	Normal	Outlier	Normal	Outlier	Outlier
3	Value	613	80	0.88	32000							
	Status	Normal	Normal	Maintenance	Outlier	Outlier	Outlier	Normal	Outlier	Normal	Outlier	Outlier
4	Value	610	79.5	0.86	47							
	Status	Normal	Normal	Normal	Maintenance	Normal	Normal	Normal	Outlier	Normal	Normal	Maintenance
5	Value	620	81.5	0.7	0							
	Status	Maintenance	Maintenance	Outlier	Outlier	Normal	Outlier	Normal	Outlier	Normal	Normal	Outlier
6	Value	620	81.5	0.87	10							
	Status	Maintenance	Maintenance	Normal	Outlier	Normal	Outlier	Normal	Outlier	Normal	Normal	Outlier
7	Value	600	81.5	0.88	41							
	Status	Maintenance	Maintenance	Maintenance	Normal	Normal	Maintenance	Normal	Outlier	Normal	Normal	Normal
8	Value	600	81.5	0.86	41							
	Status	Maintenance	Maintenance	Normal	Normal	Normal	Maintenance	Normal	Outlier	Normal	Normal	Normal

9	Value	610	82	0.86	47							
	Status	Normal	Maintenance	Normal	Maintenance	Normal	Normal	Outlier	Outlier	Outlier	Normal	Normal
10	Value	65500	65500	0	47							
	Status	Outlier	Outlier	Outlier	Maintenance	Outlier	Outlier	Outlier	Outlier	Outlier	Outlier	Outlier

4. Data Flow Diagram



5. Program Persiapan Dataset

Semua kode dan file dapat diakses pada https://github.com/StudentHagal/Thesis

```
# -*- coding: utf-8 -*-
"""Dataset_Processing.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/19MvHUBVDHEO8GOxz7n2EijTCBViy2z-V

# 0) Preparation

## Load Requirements
"""

# Commented out IPython magic to ensure Python compatibility.
# Libraries
import sys
import pandas as pd
import hashlib
import matplotlib
import numpy as np
import pandas as pd
import sklearn
import scipy
import joblib
import matplotlib.pyplot as plt
from collections import Counter
from imblearn.over_sampling import SMOTE
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from sklearn.decomposition import PCA
from sklearn.metrics import r2_score

#Extension
!pip install ipython-autotime
# %load_ext autotime

print('Load completed')

"""## Check Lib version"""

# Check the versions of libraries
# !!! WARNING !!!
```

```
# Important because model result may be different for other version

print('Python: {}'.format(sys.version))
print('scipy: {}'.format(scipy.__version__))
print('numpy: {}'.format(np.__version__))
print('matplotlib: {}'.format(matplotlib.__version__))
print('pandas: {}'.format(pd.__version__))
print('sklearn: {}'.format(sklearn.__version__))
print('joblib: {}'.format(joblib.__version__))

"""## Global Variables declaration"""

# Global Variable

#known hash value of file
compare = '10f68054068bc4ac4d7fee65e8fd7a184151e3ce3abce5d85350fc13c412d93b'
#Stored file path
raw_file = "/content/drive/MyDrive/Colab Notebooks/ML Resources/Raw Data/Format
csv/database.csv"
#environment path
code_dir = '/content/drive/MyDrive/Colab Notebooks/Thesis Repo/'
model_dir = '/content/drive/MyDrive/Colab Notebooks/ML Resources/models/'
process_dir = '/content/drive/MyDrive/Colab Notebooks/ML Resources/Processed Data/'
#Column names for dataset
col_names = ['No', 'Datetime', 'Control_Mode', 'L1', 'L2', 'L3', 'Frequency', 'kVA_Total',
          'kVA_L1', 'kVA_L2', 'kVA_L3', 'KVAr', 'Oil_Pressure', 'Coolant_Temp',
          'Charger_Alternator', 'PF_Avg', 'PF_L1', 'PF_L2',
          'PF_L3', 'L1_N', 'L2_N', 'L3_N', 'Source_Ext_Voltage', 'ECU_Temp',
          'RPM', 'Train_code']
features = ['Oil_Pressure', 'Coolant_Temp', 'PF_Avg', 'ECU_Temp']
df = pd.read_csv(raw_file, names=col_names)
#Unused columns in dataset
unused_col = ['No', 'Control_Mode', 'L1', 'L2', 'L3', 'Frequency',
           'kVA_Total', 'kVA_L1', 'kVA_L2', 'kVA_L3', 'KVAr', 'Charger_Alternator',
           'RPM', 'PF_L1', 'PF_L2', 'PF_L3', 'L1_N','L2_N', 'L3_N',
           'Source_Ext_Voltage']

results = []
names = []
array_predict = []

# Algorithm considered
model_LinearReg = LinearRegression()
model_BayesRidge = linear_model.BayesianRidge()
model_SVR = SVR()
models = []
models.append(('LinearReg', model_LinearReg))
models.append(('BR', model_BayesRidge))
models.append(('SVR', model_SVR))

#Threshold value variable 40% min, 60% max
oil_threshold_40, oil_threshold_60 = df.Oil_Pressure.quantile([0.4 , 0.6])
coolant_threshold_40, coolant_threshold_60 = df.Coolant_Temp.quantile([0.4 , 0.6])
pf_threshold_40, pf_threshold_60 = df.PF_Avg.quantile([0.4 , 0.6])
ecu_threshold_40, ecu_threshold_60 = df.ECU_Temp.quantile([0.4 , 0.6])
```

```
#Threshold value variable 25% min, 75% max
oil_threshold_25, oil_threshold_75 = df.Oil_Pressure.quantile([0.25 , 0.75])
coolant_threshold_25, coolant_threshold_75 = df.Coolant_Temp.quantile([0.25 , 0.75])
pf_threshold_25, pf_threshold_75 = df.PF_Avg.quantile([0.25 , 0.75])
ecu_threshold_25, ecu_threshold_75 = df.ECU_Temp.quantile([0.25 , 0.75])

"""## Hash Check Function"""

# Function that returns the SHA-2 hash of the file
def hash_file(filename):

  # make a hash object with SHA-2
  h = hashlib.sha256()

  # open file for reading in binary mode
  with open('/content/drive/MyDrive/Colab Notebooks/ML Resources/Raw Data/Format
csv/database.csv','rb') as file:
      # loop till the end of the file
      chunk = 0
      while chunk != b'':
          # read only 1024 bytes at a time
          chunk = file.read(1024)
          h.update(chunk)

  # return the hex representation of digest
  return h.hexdigest()

"""# 1) Processing Dataset

### Hash Validation
"""

# Checking hash value of a file
compare = '10f68054068bc4ac4d7fee65e8fd7a184151e3ce3abce5d85350fc13c412d93b'
message = hash_file('database.csv')
print('SHA-256 value of your file is :')
print(message)

if (compare == message) :
  print('Hash check SUCCESS')
else :
  print('Hash check INVALID')

"""## Feature Selection"""

# Summarize raw dataset
pd.set_option('display.max_columns', None)
# size of (row, column) data
print('Total data size : ', df.shape)
# show sample entry of data
df.head(5)

# statistical value of data
df.describe()
```

```
df.info()

# class distribution
df.groupby('Train_code').size()

# convert Datetime Dtype
df['Datetime'] = pd.to_datetime(df.Datetime)

df.info()

# Feature selection
# delete columns with condition
df.drop(labels=unused_col, axis=1, inplace=True)
df

# statistical value of data
df.describe()

# visualisasi dalam grafik
df.hist()
pyplot.show()

#amount of Outlier data
df[(df[features] >= df.quantile(q=0.75)) | (df[features] <= df.quantile(q=0.25))].count()

"""## Data Processing

### Label data for Classification
"""

#Print quantile value for label threshold

print('Threshold values (40%, 60%) :')
print('Oil_Pressure  = ', [oil_threshold_40, oil_threshold_60])
print('Coolant_Temp  = ', [coolant_threshold_40, coolant_threshold_60])
print('PF_Avg        = ', [pf_threshold_40, pf_threshold_60])
print('ECU_Temp      = ', [ecu_threshold_40, ecu_threshold_60])
print('')
print('Threshold values (25%, 75%) :')
print('Oil_Pressure  = ', [oil_threshold_25, oil_threshold_75])
print('Coolant_Temp  = ', [coolant_threshold_25, coolant_threshold_75])
print('PF_Avg        = ', [pf_threshold_25, pf_threshold_75])
print('ECU_Temp      = ', [ecu_threshold_25, ecu_threshold_75])

#Stat_Outlier = value <= 25% or value >= 75%
Outlier_Oil = (df['Oil_Pressure'] <= oil_threshold_25) | (df['Oil_Pressure'] >=
oil_threshold_75)
Outlier_Coolant =(df['Coolant_Temp'] <= coolant_threshold_25) | (df['Coolant_Temp'] >=
coolant_threshold_75)
Outlier_ECU = (df['ECU_Temp'] <= ecu_threshold_25) | (df['ECU_Temp'] >=
ecu_threshold_75)
Outlier_PFA = (df['PF_Avg'] <= pf_threshold_25) | (df['PF_Avg'] >= pf_threshold_75)

#All parameter have Stat_Outlier condition TRUE then it is labelled as Outlier
df_Outlier = df[Outlier_Oil & Outlier_Coolant & Outlier_ECU & Outlier_PFA]
```

```
df_Outlier.insert(5, 'Label', 'Outlier', True)
print('Outlier data')
print(df_Outlier.head(5))
print('Outlier data size : ', df_Outlier.shape)
print(' ')

#Stat_Normal = 40% <= value <= 60%
Normal_Oil = df['Oil_Pressure'].between(oil_threshold_40, oil_threshold_60)
Normal_Coolant = df['Coolant_Temp'].between(coolant_threshold_40, coolant_threshold_60)
Normal_ECU = df['ECU_Temp'].between(ecu_threshold_40, ecu_threshold_60)
Normal_PFA = df['PF_Avg'].between(pf_threshold_40, pf_threshold_60)

#When any parameter have all Stat_Normal condition TRUE then it is labelled as Normal
df_Normal = df[Normal_Oil & Normal_Coolant & Normal_ECU & Normal_PFA]
df_Normal.insert(5, 'Label', 'Normal', True)
print('Normal data')
print(df_Normal.head(5))
print('Normal data size : ', df_Normal.shape)
print(' ')

#Stat_Maintenance = 25% < value < 40% or 60% < value < 75%
Maintenance_Oil = (df['Oil_Pressure'].between(oil_threshold_25, oil_threshold_40,
inclusive=False)) | (df['Oil_Pressure'].between(oil_threshold_60, oil_threshold_75,
inclusive=False))
Maintenance_Coolant = (df['Coolant_Temp'].between(coolant_threshold_25,
coolant_threshold_40, inclusive=False)) |
(df['Coolant_Temp'].between(coolant_threshold_60, coolant_threshold_75, inclusive=False))
Maintenance_ECU = (df['ECU_Temp'].between(ecu_threshold_25, ecu_threshold_40,
inclusive=False)) | (df['ECU_Temp'].between(ecu_threshold_60, ecu_threshold_75,
inclusive=False))
Maintenance_PFA = (df['PF_Avg'].between(pf_threshold_25, pf_threshold_40,
inclusive=False)) | (df['PF_Avg'].between(pf_threshold_60, pf_threshold_75,
inclusive=False))

#When any feature have Stat_Maintenance condition TRUE then it is labelled as Maintenance
df_Maintenance = df[Maintenance_Oil & Maintenance_Coolant & Maintenance_ECU &
Maintenance_PFA]
df_Maintenance.insert(5, 'Label', 'Maintenance', True)
print('Maintenance data')
print(df_Maintenance.head(5))
print('Maintenance data size : ', df_Maintenance.shape)
print(' ')

#Separate unlabelled data then export
df_Unlabelled = pd.concat([df, df_Outlier, df_Outlier,
                           df_Normal, df_Normal,
                           df_Maintenance, df_Maintenance]).drop_duplicates(keep=False)
df_Unlabelled.drop('Label',axis=1, inplace=True)
print('Unlabelled data')
print(df_Unlabelled.head(5))
print('Unlabelled data size : ', df_Unlabelled.shape)
df_Unlabelled.to_csv(process_dir+'unlabelled.csv')

#Training dataset 1 summary (include outlier)
```

```
#merge labelled data into training dataset
combine = pd.merge(df_Maintenance, df_Outlier, how = 'outer')
labeled_dataset1 = pd.merge(combine, df_Normal, how = 'outer')
print(labeled_dataset1)

#Visualize data
print('Training dataset size with outlier : ', labeled_dataset1.shape)
# class distribution
print(labeled_dataset1.groupby('Label').size())
print(' ')
# statistical value of data
print(labeled_dataset1.describe(percentiles=[.25, .4, .5, .6, .75]))
# box and whisker plots
labeled_dataset1.plot(kind='box', subplots=True, layout=(2,3), sharex=False, sharey=False)
pyplot.show()
# histograms
labeled_dataset1.hist()
pyplot.show()
#export df to csv
labeled_dataset1.to_csv(process_dir+'labeled_dataset1.csv')

#Training dataset 2 Summary (exclude outlier)

# merge labelled data into training dataset
labeled_dataset2 = pd.merge(df_Maintenance, df_Normal, how = 'outer')
print(labeled_dataset2)

#Visualize data
print('Training dataset size excluding outlier : ', labeled_dataset2.shape)
# class distribution
print(labeled_dataset2.groupby('Label').size())
print(' ')
# statistical value of data
print(labeled_dataset2.describe(percentiles=[.25, .4, .5, .6, .75]))
# box and whisker plots
labeled_dataset2.plot(kind='box', subplots=True, layout=(2,3), sharex=False, sharey=False)
pyplot.show()
# histograms
labeled_dataset2.hist()
pyplot.show()
#export df to csv
labeled_dataset2.to_csv(process_dir+'labeled_dataset2.csv')

"""### Filtered data for RUL"""

# Pre-processing
print('Total data size : ', df.shape)
df.drop_duplicates()
print('Removed duplicates data size : ', df.shape)

#Filter Outlier data from each feature
Filtered_Oil = df['Oil_Pressure'].between(oil_threshold_25, oil_threshold_75)
Filtered_Coolant = df['Coolant_Temp'].between(coolant_threshold_25,
coolant_threshold_75)
Filtered_ECU = df['ECU_Temp'].between(ecu_threshold_25, ecu_threshold_75)
```

```
Filtered_PFA = df['PF_Avg'].between(pf_threshold_25, pf_threshold_75)

df_Filtered = df[Filtered_Coolant & Filtered_ECU & Filtered_Oil & Filtered_PFA]
df_Filtered

# visualisasi dalam grafik
df_Filtered.hist()
pyplot.show()

# Create dataframe ONLY for Maintenance condition
#Stat_Maintenance = 25% < value < 40% or 60% < value < 75%
Maintenance_Oil = (df_Filtered['Oil_Pressure'].between(oil_threshold_25, oil_threshold_40,
inclusive=False)) | (df_Filtered['Oil_Pressure'].between(oil_threshold_60, oil_threshold_75,
inclusive=False))
Maintenance_Coolant = (df_Filtered['Coolant_Temp'].between(coolant_threshold_25,
coolant_threshold_40, inclusive=False)) |
(df_Filtered['Coolant_Temp'].between(coolant_threshold_60, coolant_threshold_75,
inclusive=False))
Maintenance_ECU = (df_Filtered['ECU_Temp'].between(ecu_threshold_25,
ecu_threshold_40, inclusive=False)) | (df_Filtered['ECU_Temp'].between(ecu_threshold_60,
ecu_threshold_75, inclusive=False))
Maintenance_PFA = (df_Filtered['PF_Avg'].between(pf_threshold_25, pf_threshold_40,
inclusive=False)) | (df_Filtered['PF_Avg'].between(pf_threshold_60, pf_threshold_75,
inclusive=False))

#When parameter have Stat_Maintenance condition TRUE then it is labelled as Maintenance
df_Maintenance = df_Filtered[Maintenance_Oil | Maintenance_Coolant | Maintenance_ECU |
Maintenance_PFA] # OR condition (choose one only)
#df_Maintenance = df_Filtered[Maintenance_Oil & Maintenance_Coolant &
Maintenance_ECU & Maintenance_PFA] # AND condition (choose one only)

df_Maintenance.insert(5, 'Label', 'Maintenance', True)
df_Maintenance = df_Maintenance.drop_duplicates(subset='Datetime')
print('Maintenance data size : ', df_Maintenance.shape)
df_Maintenance.head(5)

# Dataset for All vehicles
#calculate diffs from each row

#df_Maintenance['diffs'] =
df_Maintenance.sort_values(['Train_code','Datetime']).groupby('Train_code')['Datetime'].diff(
) #grouped by train code (Choose one)
df_Maintenance = df_Maintenance.sort_values(by='Datetime')
#NOT grouped (Choose one)
df_Maintenance['diffs'] = df_Maintenance['Datetime'].diff()
#NOT grouped (Choose one)
df_Maintenance['diffs'] = df_Maintenance['diffs'].fillna(pd.Timedelta(seconds=0))
df_Maintenance['Delta_Time_h'] = df_Maintenance.diffs/np.timedelta64(1,'h')
df_Maintenance

# Visualize Mean Time Between Maintenance (MTBM)
print(df_Maintenance.groupby('Train_code').Delta_Time_h.mean())
df_Maintenance.groupby('Train_code').Delta_Time_h.mean().plot(kind='bar')

#Export Dataset for All vehicles grouped by train code
```

```
df_Maintenance.to_csv(process_dir+'all_vehicles.csv')
```

6. Program Pemodelan dan Pengujian ML Klasifikasi Opsi 1 dan Opsi 2
Semua kode dan file dapat diakses pada https://github.com/StudentHagal/Thesis

```
# -*- coding: utf-8 -*-
"""Statistical Approach.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1w9aDVfndbYjgalX9jcHFQYknExy4-F4-

# ML Modelling

## 0) Preparation

### Load Requirements
"""

# Commented out IPython magic to ensure Python compatibility.
# Github repository
!git clone https://github.com/StudentHagal/Thesis.git

# Libraries
import sys
import pandas as pd
import hashlib
import matplotlib
import numpy as np
import pandas as pd
import sklearn
import scipy
import joblib
import matplotlib.pyplot as plt
from collections import Counter
from imblearn.over_sampling import SMOTE
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import plot_confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import DBSCAN
from sklearn.cluster import KMeans
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from sklearn.decomposition import PCA

#Extension
!pip install ipython-autotime
# %load_ext autotime

print('Load completed')

# Check the versions of libraries
# !!! WARNING !!!
# Important because model result may be different for other version

print('Python: {}'.format(sys.version))
print('scipy: {}'.format(scipy.__version__))
print('numpy: {}'.format(np.__version__))
print('matplotlib: {}'.format(matplotlib.__version__))
print('pandas: {}'.format(pd.__version__))
print('sklearn: {}'.format(sklearn.__version__))
print('joblib: {}'.format(joblib.__version__))

"""### Global variables"""

# Global Variable
compare = '10f68054068bc4ac4d7fee65e8fd7a184151e3ce3abce5d85350fc13c412d93b'
#known hash value of file
url = "https://raw.githubusercontent.com/StudentHagal/Thesis/main/Resources/database.csv"
#Stored file path
dir = '/content/drive/MyDrive/Colab Notebooks/ML Resources/models/'
#path to export built models
process_dir = '/content/drive/MyDrive/Colab Notebooks/ML Resources/Processed Data/'
col_names = ['No', 'Datetime', 'Control_Mode', 'L1', 'L2', 'L3', 'Frequency', 'kVA_Total',
#Column names for dataset
            'kVA_L1', 'kVA_L2', 'kVA_L3', 'KVAr', 'Oil_Pressure', 'Coolant_Temp',
            'Charger_Alternator', 'PF_Avg', 'PF_L1', 'PF_L2',
            'PF_L3', 'L1_N', 'L2_N', 'L3_N', 'Source_Ext_Voltage', 'ECU_Temp',
            'RPM', 'Train_code']
unused_col = ['No', 'Control_Mode', 'L1', 'L2', 'L3', 'Frequency',                    #Unused
columns in dataset
             'kVA_Total', 'kVA_L1', 'kVA_L2', 'kVA_L3', 'KVAr', 'Charger_Alternator',
             'RPM', 'PF_L1', 'PF_L2', 'PF_L3', 'L1_N','L2_N', 'L3_N',
             'Source_Ext_Voltage', 'Train_code']
df = pd.read_csv(url, names=col_names)
features = ['Oil_Pressure', 'Coolant_Temp', 'PF_Avg', 'ECU_Temp']
results1 = []
names1 = []
results2 = []
names2 = []

# Algorithm considered
model_LR = LogisticRegression(max_iter=475)
model_KNN = KNeighborsClassifier()
model_DT = DecisionTreeClassifier()
model_SVM = SVC(gamma='auto')
```

```
model_RF = RandomForestClassifier()
model_LDA = LinearDiscriminantAnalysis()
model_NB = GaussianNB()
model_Kmeans = KMeans(n_clusters=3, random_state=0)
model_dbscan = DBSCAN(eps=0.5, min_samples=100)
balancing_smote = SMOTE(random_state=1)
########################################################################
models = []
models.append(('LR', model_LR))
models.append(('KNN', model_KNN))
models.append(('DT', model_DT))
models.append(('SVM', model_SVM))
models.append(('RF', model_RF))
models.append(('LDA', model_LDA))
models.append(('NB', model_NB))

#Threshold value variable 10% min, 90% max
oil_threshold_10, oil_threshold_90 = df.Oil_Pressure.quantile([0.10 , 0.90])
coolant_threshold_10, coolant_threshold_90 = df.Coolant_Temp.quantile([0.10 , 0.90])
pf_threshold_10, pf_threshold_90 = df.PF_Avg.quantile([0.10 , 0.90])
ecu_threshold_10, ecu_threshold_90 = df.ECU_Temp.quantile([0.10 , 0.90])
#Threshold value variable 40% min, 60% max
oil_threshold_40, oil_threshold_60 = df.Oil_Pressure.quantile([0.4 , 0.6])
coolant_threshold_40, coolant_threshold_60 = df.Coolant_Temp.quantile([0.4 , 0.6])
pf_threshold_40, pf_threshold_60 = df.PF_Avg.quantile([0.4 , 0.6])
ecu_threshold_40, ecu_threshold_60 = df.ECU_Temp.quantile([0.4 , 0.6])
#Threshold value variable 25% min, 75% max
oil_threshold_25, oil_threshold_75 = df.Oil_Pressure.quantile([0.25 , 0.75])
coolant_threshold_25, coolant_threshold_75 = df.Coolant_Temp.quantile([0.25 , 0.75])
pf_threshold_25, pf_threshold_75 = df.PF_Avg.quantile([0.25 , 0.75])
ecu_threshold_25, ecu_threshold_75 = df.ECU_Temp.quantile([0.25 , 0.75])

"""### Hash Check Function"""

# Function that returns the SHA-2 hash of the file
def hash_file(filename):

  # make a hash object with SHA-2
  h = hashlib.sha256()

  # open file for reading in binary mode
  with open('/content/Thesis/Resources/database.csv','rb') as file:
    # loop till the end of the file
    chunk = 0
    while chunk != b'':
      # read only 1024 bytes at a time
      chunk = file.read(1024)
      h.update(chunk)

  # return the hex representation of digest
  return h.hexdigest()

"""### Show cluster result Function"""

#Function to show cluster result
```

```
def show_clusters(dataset, labels):
  df = pd.DataFrame(dict(x=X_principal['P1'], y=X_principal['P2'], label=labels))
  colors = {-1:'black', 0:'blue', 1:'red', 2:'yellow' ,3:'green', 4:'orange'}
  fig, ax = plt.subplots(figsize=(8,8))
  grouped = df.groupby('label')
  for key, group in grouped:
    group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color=colors[key])
  plt.xlabel('X_1')
  plt.ylabel('X_2')
  plt.title('Estimated number of clusters: %d' % n_clusters_)
  plt.show()

"""## 1) Standardized Dataset"""

# Checking hash value of a file
compare = '10f68054068bc4ac4d7fee65e8fd7a184151e3ce3abce5d85350fc13c412d93b'
message = hash_file('database.csv')
print('SHA-256 value of your file is :')
print(message)

if (compare == message) :
  print('Hash check SUCCESS')
else :
  print('Hash check INVALID')

# Summarize raw dataset
pd.set_option('display.max_columns', None)
# size of (row, column) data
print('Total data size : ', df.shape)

# show sample entry of data
df.head(5)

# statistical value of data
df.describe()

# class distribution
df.groupby('Train_code').size()

"""## 2) Feature Engineering

### Feature selection
"""

# Feature selection
# delete columns with condition
df.drop(labels=unused_col, axis=1, inplace=True)
df

# statistical value of data
print(df.describe(percentiles=[.1, .25, .4, .6, .75, .9]))

# histograms
df.hist()
pyplot.show()
```

```
#Print quantile value for features

print('Threshold values (10%, 90%) :')
print('Oil_Pressure  = ', [oil_threshold_10, oil_threshold_90])
print('Coolant_Temp  = ', [coolant_threshold_10, coolant_threshold_90])
print('PF_Avg        = ', [pf_threshold_10, pf_threshold_90])
print('ECU_Temp      = ', [ecu_threshold_10, ecu_threshold_90])
print('')
print('Threshold values (40%, 60%) :')
print('Oil_Pressure  = ', [oil_threshold_40, oil_threshold_60])
print('Coolant_Temp  = ', [coolant_threshold_40, coolant_threshold_60])
print('PF_Avg        = ', [pf_threshold_40, pf_threshold_60])
print('ECU_Temp      = ', [ecu_threshold_40, ecu_threshold_60])
print('')
print('Threshold values (25%, 75%) :')
print('Oil_Pressure  = ', [oil_threshold_25, oil_threshold_75])
print('Coolant_Temp  = ', [coolant_threshold_25, coolant_threshold_75])
print('PF_Avg        = ', [pf_threshold_25, pf_threshold_75])
print('ECU_Temp      = ', [ecu_threshold_25, ecu_threshold_75])

"""### Data Labelling"""

# Pre-processing
print('Total data size : ', df.shape)
df.drop_duplicates()
print('Removed duplicates data size : ', df.shape)
print(' ')

#Stat_Outlier = value <= 25% or value >= 75%
Outlier_Oil = (df['Oil_Pressure'] <= oil_threshold_25) | (df['Oil_Pressure'] >=
oil_threshold_75)
Outlier_Coolant =(df['Coolant_Temp'] <= coolant_threshold_25) | (df['Coolant_Temp'] >=
coolant_threshold_75)
Outlier_ECU = (df['ECU_Temp'] <= ecu_threshold_25) | (df['ECU_Temp'] >=
ecu_threshold_75)
Outlier_PFA = (df['PF_Avg'] <= pf_threshold_25) | (df['PF_Avg'] >= pf_threshold_75)

#All parameter have Stat_Outlier condition TRUE then it is labelled as Outlier
df_Outlier = df[Outlier_Oil & Outlier_Coolant & Outlier_ECU & Outlier_PFA]
df_Outlier.insert(4, 'Label', 'Outlier', True)
print('Outlier data')
print(df_Outlier.head(5))
print('Outlier data size : ', df_Outlier.shape)
print(' ')

#Stat_Normal = 40% <= value <= 60%
Normal_Oil = df['Oil_Pressure'].between(oil_threshold_40, oil_threshold_60)
Normal_Coolant = df['Coolant_Temp'].between(coolant_threshold_40, coolant_threshold_60)
Normal_ECU = df['ECU_Temp'].between(ecu_threshold_40, ecu_threshold_60)
Normal_PFA = df['PF_Avg'].between(pf_threshold_40, pf_threshold_60)

#When any parameter have all Stat_Normal condition TRUE then it is labelled as Normal
df_Normal = df[Normal_Oil & Normal_Coolant & Normal_ECU & Normal_PFA]
df_Normal.insert(4, 'Label', 'Normal', True)
```

```
print('Normal data')
print(df_Normal.head(5))
print('Normal data size : ', df_Normal.shape)
print(' ')

#Stat_Maintenance = 25% < value < 40% or 60% < value < 75%
Maintenance_Oil = (df['Oil_Pressure'].between(oil_threshold_25, oil_threshold_40,
inclusive=False)) | (df['Oil_Pressure'].between(oil_threshold_60, oil_threshold_75,
inclusive=False))
Maintenance_Coolant = (df['Coolant_Temp'].between(coolant_threshold_25,
coolant_threshold_40, inclusive=False)) |
(df['Coolant_Temp'].between(coolant_threshold_60, coolant_threshold_75, inclusive=False))
Maintenance_ECU = (df['ECU_Temp'].between(ecu_threshold_25, ecu_threshold_40,
inclusive=False)) | (df['ECU_Temp'].between(ecu_threshold_60, ecu_threshold_75,
inclusive=False))
Maintenance_PFA = (df['PF_Avg'].between(pf_threshold_25, pf_threshold_40,
inclusive=False)) | (df['PF_Avg'].between(pf_threshold_60, pf_threshold_75,
inclusive=False))

#When any parameter have all Stat_Maintenance condition TRUE then it is labelled as
Maintenance
df_Maintenance = df[Maintenance_Oil & Maintenance_Coolant & Maintenance_ECU &
Maintenance_PFA]
df_Maintenance.insert(4, 'Label', 'Maintenance', True)
print('Maintenance data')
print(df_Maintenance.head(5))
print('Maintenance data size : ', df_Maintenance.shape)
print(' ')

#Separate unlabelled data
df_Unlabelled = pd.concat([df, df_Outlier, df_Outlier,
                    df_Normal, df_Normal,
                    df_Maintenance, df_Maintenance]).drop_duplicates(keep=False)
df_Unlabelled.drop('Label',axis=1, inplace=True)
print('Unlabelled data')
print(df_Unlabelled.head(5))
print('Unlabelled data size : ', df_Unlabelled.shape)

df_Unlabelled.to_csv(process_dir+'unlabelled.csv')

"""## 3) Algorithm Selection
Differ for each iteration, by default all consideration will be selected to build ML model. But
after the best choice is identified, there may be only a few algorithm to be selected.
Algorithm declaration code is in Global variables part.

Algorithm considered:
LogisticRegression(), KNeighborsClassifier(), DecisionTreeClassifier(), SVC(),
RandomForestClassifier(), LinearDiscriminantAnalysis(), GaussianNB()

## 4) Data Pre Processing

### Outlier
"""

#Training data summary 1 (include outlier)
```

```
#merge labelled data into training dataset
combine = pd.merge(df_Maintenance, df_Outlier, how = 'outer')
labeled_dataset1 = pd.merge(combine, df_Normal, how = 'outer')
print(labeled_dataset1)

#Visualize data
print('Training dataset size with outlier : ', labeled_dataset1.shape)
# class distribution
print(labeled_dataset1.groupby('Label').size())
print(' ')
# statistical value of data
print(labeled_dataset1.describe(percentiles=[.1, .4, .5, .6, .9]))
# box and whisker plots
labeled_dataset1.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
pyplot.show()
# histograms
labeled_dataset1.hist()
pyplot.show()

# Pre-process training and test dataset to balance class label (with outlier)

X1 = labeled_dataset1[features]
y1 = labeled_dataset1['Label']

print('Original dataset shape %s' % Counter(y1))
X1_res, y1_res = balancing_smote.fit_resample(X1, y1)
print('Resampled dataset shape %s' % Counter(y1_res))

# Split-out validation and test dataset (with outlier)
X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1_res, y1_res, test_size=0.20,
random_state=1)

print('training data size = ', X1_train.shape)
print('testing data size = ', X1_test.shape)
print('data split completed')

"""### No Outlier"""

#Training data Summary 2 (exclude outlier)

# merge labelled data into training dataset
labeled_dataset2 = pd.merge(df_Maintenance, df_Normal, how = 'outer')
print(labeled_dataset2)

#Visualize data
print('Training dataset size excluding outlier : ', labeled_dataset2.shape)
# class distribution
print(labeled_dataset2.groupby('Label').size())
print(' ')
# statistical value of data
print(labeled_dataset2.describe(percentiles=[.1, .4, .5, .6, .9]))
# box and whisker plots
labeled_dataset2.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
pyplot.show()
```

```
# histograms
labeled_dataset2.hist()
pyplot.show()

# Pre-process training and test dataset to balance class label (with outlier)

X2 = labeled_dataset2[features]
y2 = labeled_dataset2['Label']

print('Original dataset shape %s' % Counter(y2))
X2_res, y2_res = balancing_smote.fit_resample(X2, y2)
print('Resampled dataset shape %s' % Counter(y2_res))

# Split-out validation dataset (exclude outlier)
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X2_res, y2_res, test_size=0.20,
random_state=1)

print('training data size = ', X2_train.shape)
print('testing data size = ', X2_test.shape)
print('data split completed')

"""## 5) Algorithm Training & Validation

### Outlier
"""

# Cross validation of each model training in turn (with Outlier)
for name, model in models:
        kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
        cv_results = cross_val_score(model, X1_train, Y1_train, cv=kfold,
                scoring='accuracy')
        results1.append(cv_results)
        names1.append(name)
        print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

# Compare result
pyplot.boxplot(results1, labels=names1)
pyplot.title('Algorithm Accuracy Comparison 1')
pyplot.show()

# Re-train each algorithm using whole training dataset with Outlier for better model
# Each model is separated in different code block for easier tracking of Runtime
model1_LR = model_LR.fit(X1_train, Y1_train)
joblib.dump(model1_LR,dir+'Opsi1_LR.joblib')

model1_KNN = model_KNN.fit(X1_train, Y1_train)
joblib.dump(model1_KNN,dir+'Opsi1_KNN.joblib')

model1_DT = model_DT.fit(X1_train, Y1_train)
joblib.dump(model1_DT,dir+'Opsi1_DT.joblib')

model1_SVM = model_SVM.fit(X1_train, Y1_train)
joblib.dump(model1_SVM,dir+'Opsi1_SVM.joblib')

model1_RF = model_RF.fit(X1_train, Y1_train)
```

```
joblib.dump(model1_RF,dir+'Opsi1_RF.joblib')

model1_LDA = model_LDA.fit(X1_train, Y1_train)
joblib.dump(model1_LDA,dir+'Opsi1_LDA.joblib')

model1_NB = model_NB.fit(X1_train, Y1_train)
joblib.dump(model1_NB,dir+'Opsi1_NB.joblib')

print('Model with outlier training completed!')

"""### No Outlier"""

# Algorithm considered
model_LR = LogisticRegression(max_iter=475)
model_KNN = KNeighborsClassifier()
model_DT = DecisionTreeClassifier()
model_SVM = SVC(gamma='auto')
model_RF = RandomForestClassifier()
model_LDA = LinearDiscriminantAnalysis()
model_NB = GaussianNB()
model_Kmeans = KMeans(n_clusters=3, random_state=0)
model_dbscan = DBSCAN(eps=0.5, min_samples=100)
#########################################################################
models = []
models.append(('LR', model_LR))
models.append(('KNN', model_KNN))
models.append(('DT', model_DT))
models.append(('SVM', model_SVM))
models.append(('RF', model_RF))
models.append(('LDA', model_LDA))
models.append(('NB', model_NB))

# evaluate each model training in turn (exclude Outlier)
for name, model in models:
        kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
        cv_results = cross_val_score(model, X2_train, Y2_train, cv=kfold,
                scoring='accuracy')
        results2.append(cv_results)
        names2.append(name)
        print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

# Compare result
pyplot.boxplot(results2, labels=names2)
pyplot.title('Algorithm Accuracy Comparison 2')
pyplot.show()

# Re-train each algorithm using whole training dataset excluding Outlier for better model
# Each model is separated in different code block for easier tracking of Runtime
model2_LR = model_LR.fit(X2_train, Y2_train)
joblib.dump(model2_LR,dir+'Opsi2_LR.joblib')

model2_KNN = model_KNN.fit(X2_train, Y2_train)
joblib.dump(model2_KNN,dir+'Opsi2_KNN.joblib')

model2_DT = model_DT.fit(X2_train, Y2_train)
```

```
joblib.dump(model2_DT,dir+'Opsi2_DT.joblib')

model2_SVM = model_SVM.fit(X2_train, Y2_train)
joblib.dump(model2_SVM,dir+'Opsi2_SVM.joblib')

model2_RF = model_RF.fit(X2_train, Y2_train)
joblib.dump(model2_RF,dir+'Opsi2_RF.joblib')

model2_LDA = model_LDA.fit(X2_train, Y2_train)
joblib.dump(model2_LDA,dir+'Opsi2_LDA.joblib')

model2_NB = model_NB.fit(X2_train, Y2_train)
joblib.dump(model2_NB,dir+'Opsi2_NB.joblib')

print('Model no outlier training completed!')

"""# ML Model Evaluation
Ada dua evaluasi yang akan dilakukan, pertama akan dievaluasi untuk mengecek
kemungkinan untuk mendapatkan hasil yang lebih baik, kedua yaitu mengevaluasi hasil yang
didapatkan untuk menentukan kesuksesan model ML dengan target sukses yang ditentukan.
Target sukses berdasarkan dua kriteria, Akurasi minimal 90% dan Runtime maksimal 15
menit.

## 1) ML Model Testing
Pada tahap ini, akan digunakan test dataset untuk menguji performa model ML dengan data
yang diketahui labelnya, namun disembunyikan saat pengujian. Dengan demikian dapat
diketahui nilai Accuracy, Precision, Recall serta F1-Score untuk dibandingkan dengan hasil
training. Jika hasilnya konsisten, atau tidak terlalu menyimpang jauh dengan nilai Akurasi
dari hasil training model, maka model ML telah tervalidasi bahwa model dapat digunakan
untuk prediksi.

### Test With Outlier
"""

#Correct label
print('Correct label distribution:')
print(accuracy_score(Y1_test, Y1_test))
print(confusion_matrix(Y1_test, Y1_test))

# Make predictions on validation dataset (with Outlier)
pred_LR = model1_LR.predict(X1_test)

print('LR Result:')
print(accuracy_score(Y1_test, pred_LR))
print(classification_report(Y1_test, pred_LR))
plot_confusion_matrix(model1_LR, X1_test, Y1_test, values_format='d', cmap=plt.cm.Blues)
plt.show()

pred_KNN = model1_KNN.predict(X1_test)

print('KNN Result:')
print(accuracy_score(Y1_test, pred_KNN))
print(classification_report(Y1_test, pred_KNN))
plot_confusion_matrix(model1_KNN, X1_test, Y1_test, values_format='d',
cmap=plt.cm.Blues)
```

```
plt.show()

pred_DT = model1_DT.predict(X1_test)

print('DT Result:')
print(accuracy_score(Y1_test, pred_DT))
print(classification_report(Y1_test, pred_DT))
plot_confusion_matrix(model1_DT, X1_test, Y1_test, values_format='d', cmap=plt.cm.Blues)
plt.show()

pred_SVM = model1_SVM.predict(X1_test)

print('SVM Result:')
print(accuracy_score(Y1_test, pred_SVM))
print(classification_report(Y1_test, pred_SVM))
plot_confusion_matrix(model1_SVM, X1_test, Y1_test, values_format='d',
cmap=plt.cm.Blues)
plt.show()

pred_RF = model1_RF.predict(X1_test)

print('RF Result:')
print(accuracy_score(Y1_test, pred_RF))
print(classification_report(Y1_test, pred_RF))
plot_confusion_matrix(model1_RF, X1_test, Y1_test, values_format='d', cmap=plt.cm.Blues)
plt.show()

pred_LDA = model1_LDA.predict(X1_test)

print('LDA Result:')
print(accuracy_score(Y1_test, pred_LDA))
print(classification_report(Y1_test, pred_LDA))
plot_confusion_matrix(model1_LDA, X1_test, Y1_test, values_format='d',
cmap=plt.cm.Blues)
plt.show()

pred_NB = model1_NB.predict(X1_test)

print('NB Result:')
print(accuracy_score(Y1_test, pred_NB))
print(classification_report(Y1_test, pred_NB))
plot_confusion_matrix(model1_NB, X1_test, Y1_test, values_format='d',
cmap=plt.cm.Blues)
plt.show()

"""### Test No Outlier"""

# Each model is separated in different code block for easier tracking of Runtime
#Correct label
print('Correct label distribution:')
print(accuracy_score(Y2_test, Y2_test))
print(confusion_matrix(Y2_test, Y2_test))

# Make predictions on validation dataset (with Outlier)
pred_LR = model2_LR.predict(X2_test)
```

```
print('LR Result:')
print(accuracy_score(Y2_test, pred_LR))
print(classification_report(Y2_test, pred_LR))
plot_confusion_matrix(model2_LR, X2_test, Y2_test, values_format='d', cmap=plt.cm.Blues)
plt.show()

pred_KNN = model2_KNN.predict(X2_test)

print('KNN Result:')
print(accuracy_score(Y2_test, pred_KNN))
print(classification_report(Y2_test, pred_KNN))
plot_confusion_matrix(model2_KNN, X2_test, Y2_test, values_format='d',
cmap=plt.cm.Blues)
plt.show()

pred_DT = model2_DT.predict(X2_test)

print('DT Result:')
print(accuracy_score(Y2_test, pred_DT))
print(classification_report(Y2_test, pred_DT))
plot_confusion_matrix(model2_DT, X2_test, Y2_test, values_format='d', cmap=plt.cm.Blues)
plt.show()

pred_SVM = model2_SVM.predict(X2_test)

print('SVM Result:')
print(accuracy_score(Y2_test, pred_SVM))
print(classification_report(Y2_test, pred_SVM))
plot_confusion_matrix(model2_SVM, X2_test, Y2_test, values_format='d',
cmap=plt.cm.Blues)
plt.show()

pred_RF = model2_RF.predict(X2_test)

print('RF Result:')
print(accuracy_score(Y2_test, pred_RF))
print(classification_report(Y2_test, pred_RF))
plot_confusion_matrix(model2_RF, X2_test, Y2_test, values_format='d', cmap=plt.cm.Blues)
plt.show()

pred_LDA = model2_LDA.predict(X2_test)

print('LDA Result:')
print(accuracy_score(Y2_test, pred_LDA))
print(classification_report(Y2_test, pred_LDA))
plot_confusion_matrix(model2_LDA, X2_test, Y2_test, values_format='d',
cmap=plt.cm.Blues)
plt.show()

pred_NB = model2_NB.predict(X2_test)

print('NB Result:')
print(accuracy_score(Y2_test, pred_NB))
print(classification_report(Y2_test, pred_NB))
```

```
plot_confusion_matrix(model2_NB, X2_test, Y2_test, values_format='d',
cmap=plt.cm.Blues)
plt.show()

"""### Prediction Test with Outlier"""

# Make predictions on unlabelled dataset with outlier
predictions1_LR = model1_LR.predict(df_Unlabelled[features])
predictions1_KNN = model1_KNN.predict(df_Unlabelled[features])
predictions1_DT = model1_DT.predict(df_Unlabelled[features])
predictions1_SVM = model1_SVM.predict(df_Unlabelled[features])
predictions1_RF = model1_RF.predict(df_Unlabelled[features])
predictions1_LDA = model1_LDA.predict(df_Unlabelled[features])
predictions1_NB = model1_NB.predict(df_Unlabelled[features])

df_predict1 =  pd.DataFrame({'LR': predictions1_LR, 'KNN': predictions1_KNN,
                  'DT':predictions1_DT, 'SVM':predictions1_SVM,
                  'RF':predictions1_RF, 'LDA':predictions1_LDA,
                  'NB':predictions1_NB})

df_predict1

"""### Prediction Test no Outlier"""

# Make predictions on unlabelled dataset no outlier
predictions2_LR = model2_LR.predict(df_Unlabelled[features])
predictions2_KNN = model2_KNN.predict(df_Unlabelled[features])
predictions2_DT = model2_DT.predict(df_Unlabelled[features])
predictions2_SVM = model2_SVM.predict(df_Unlabelled[features])
predictions2_RF = model2_RF.predict(df_Unlabelled[features])
predictions2_LDA = model2_LDA.predict(df_Unlabelled[features])
predictions2_NB = model2_NB.predict(df_Unlabelled[features])

df_predict2 =  pd.DataFrame({'LR': predictions2_LR, 'KNN': predictions2_KNN,
                  'DT':predictions2_DT, 'SVM':predictions2_SVM,
                  'RF':predictions2_RF, 'LDA':predictions2_LDA,
                  'NB':predictions2_NB})


df_predict2

"""## 2) Result Documenting"""

df_result1 = pd.concat([df_Unlabelled.reset_index(drop=True),
df_predict1.reset_index(drop=True)],axis=1)
df_result1

df_result2 = pd.concat([df_Unlabelled.reset_index(drop=True),
df_predict2.reset_index(drop=True)],axis=1)
df_result2

df_result1.to_csv(process_dir+'predict1_smote.csv')
df_result2.to_csv(process_dir+'predict2_smote.csv')
```

7. Program Pemodelan dan Pengujian ML Klasifikasi Opsi 3 dan Opsi 4
Semua kode dan file dapat diakses pada https://github.com/StudentHagal/Thesis

```
# -*- coding: utf-8 -*-
"""Dataset_Processing.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/19MvHUBVDHEO8GOxz7n2EijTCBViy2z-V

# 0) Preparation

## Load Requirements
"""

# Commented out IPython magic to ensure Python compatibility.
# Libraries
import sys
import pandas as pd
import hashlib
import matplotlib
import numpy as np
import pandas as pd
import sklearn
import scipy
import joblib
import matplotlib.pyplot as plt
from collections import Counter
from imblearn.over_sampling import SMOTE
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from sklearn.decomposition import PCA
from sklearn.metrics import r2_score

#Extension
!pip install ipython-autotime
# %load_ext autotime

print('Load completed')

"""## Check Lib version"""

# Check the versions of libraries
```

```
# !!! WARNING !!!
# Important because model result may be different for other version

print('Python: {}'.format(sys.version))
print('scipy: {}'.format(scipy.__version__))
print('numpy: {}'.format(np.__version__))
print('matplotlib: {}'.format(matplotlib.__version__))
print('pandas: {}'.format(pd.__version__))
print('sklearn: {}'.format(sklearn.__version__))
print('joblib: {}'.format(joblib.__version__))

"""## Global Variables declaration"""

# Global Variable

#known hash value of file
compare = '10f68054068bc4ac4d7fee65e8fd7a184151e3ce3abce5d85350fc13c412d93b'
#Stored file path
raw_file = "/content/drive/MyDrive/Colab Notebooks/ML Resources/Raw Data/Format
csv/database.csv"
#environment path
code_dir = '/content/drive/MyDrive/Colab Notebooks/Thesis Repo/'
model_dir = '/content/drive/MyDrive/Colab Notebooks/ML Resources/models/'
process_dir = '/content/drive/MyDrive/Colab Notebooks/ML Resources/Processed Data/'
#Column names for dataset
col_names = ['No', 'Datetime', 'Control_Mode', 'L1', 'L2', 'L3', 'Frequency', 'kVA_Total',
          'kVA_L1', 'kVA_L2', 'kVA_L3', 'KVAr', 'Oil_Pressure', 'Coolant_Temp',
          'Charger_Alternator', 'PF_Avg', 'PF_L1', 'PF_L2',
          'PF_L3', 'L1_N', 'L2_N', 'L3_N', 'Source_Ext_Voltage', 'ECU_Temp',
          'RPM', 'Train_code']
features = ['Oil_Pressure', 'Coolant_Temp', 'PF_Avg', 'ECU_Temp']
df = pd.read_csv(raw_file, names=col_names)
#Unused columns in dataset
unused_col = ['No', 'Control_Mode', 'L1', 'L2', 'L3', 'Frequency',
           'kVA_Total', 'kVA_L1', 'kVA_L2', 'kVA_L3', 'KVAr', 'Charger_Alternator',
           'RPM', 'PF_L1', 'PF_L2', 'PF_L3', 'L1_N','L2_N', 'L3_N',
           'Source_Ext_Voltage']

results = []
names = []
array_predict = []

# Algorithm considered
model_LinearReg = LinearRegression()
model_BayesRidge = linear_model.BayesianRidge()
model_SVR = SVR()
models = []
models.append(('LinearReg', model_LinearReg))
models.append(('BR', model_BayesRidge))
models.append(('SVR', model_SVR))

#Threshold value variable 40% min, 60% max
oil_threshold_40, oil_threshold_60 = df.Oil_Pressure.quantile([0.4 , 0.6])
coolant_threshold_40, coolant_threshold_60 = df.Coolant_Temp.quantile([0.4 , 0.6])
pf_threshold_40, pf_threshold_60 = df.PF_Avg.quantile([0.4 , 0.6])
```

```
ecu_threshold_40, ecu_threshold_60 = df.ECU_Temp.quantile([0.4 , 0.6])
#Threshold value variable 25% min, 75% max
oil_threshold_25, oil_threshold_75 = df.Oil_Pressure.quantile([0.25 , 0.75])
coolant_threshold_25, coolant_threshold_75 = df.Coolant_Temp.quantile([0.25 , 0.75])
pf_threshold_25, pf_threshold_75 = df.PF_Avg.quantile([0.25 , 0.75])
ecu_threshold_25, ecu_threshold_75 = df.ECU_Temp.quantile([0.25 , 0.75])

"""## Hash Check Function"""

# Function that returns the SHA-2 hash of the file
def hash_file(filename):

  # make a hash object with SHA-2
  h = hashlib.sha256()

  # open file for reading in binary mode
  with open('/content/drive/MyDrive/Colab Notebooks/ML Resources/Raw Data/Format
csv/database.csv','rb') as file:
      # loop till the end of the file
      chunk = 0
      while chunk != b'':
          # read only 1024 bytes at a time
          chunk = file.read(1024)
          h.update(chunk)

  # return the hex representation of digest
  return h.hexdigest()

"""# 1) Processing Dataset

### Hash Validation
"""

# Checking hash value of a file
compare = '10f68054068bc4ac4d7fee65e8fd7a184151e3ce3abce5d85350fc13c412d93b'
message = hash_file('database.csv')
print('SHA-256 value of your file is :')
print(message)

if (compare == message) :
  print('Hash check SUCCESS')
else :
  print('Hash check INVALID')

"""## Feature Selection"""

# Summarize raw dataset
pd.set_option('display.max_columns', None)
# size of (row, column) data
print('Total data size : ', df.shape)
# show sample entry of data
df.head(5)

# statistical value of data
df.describe()
```

```
df.info()

# class distribution
df.groupby('Train_code').size()

# convert Datetime Dtype
df['Datetime'] = pd.to_datetime(df.Datetime)

df.info()

# Feature selection
# delete columns with condition
df.drop(labels=unused_col, axis=1, inplace=True)
df

# statistical value of data
df.describe()

# visualisasi dalam grafik
df.hist()
pyplot.show()

#amount of Outlier data
df[(df[features] >= df.quantile(q=0.75)) | (df[features] <= df.quantile(q=0.25))].count()

"""## Data Processing

### Label data for Classification
"""

#Print quantile value for label threshold

print('Threshold values (40%, 60%) :')
print('Oil_Pressure  = ', [oil_threshold_40, oil_threshold_60])
print('Coolant_Temp  = ', [coolant_threshold_40, coolant_threshold_60])
print('PF_Avg        = ', [pf_threshold_40, pf_threshold_60])
print('ECU_Temp      = ', [ecu_threshold_40, ecu_threshold_60])
print('')
print('Threshold values (25%, 75%) :')
print('Oil_Pressure  = ', [oil_threshold_25, oil_threshold_75])
print('Coolant_Temp  = ', [coolant_threshold_25, coolant_threshold_75])
print('PF_Avg        = ', [pf_threshold_25, pf_threshold_75])
print('ECU_Temp      = ', [ecu_threshold_25, ecu_threshold_75])

#Stat_Outlier = value <= 25% or value >= 75%
Outlier_Oil = (df['Oil_Pressure'] <= oil_threshold_25) | (df['Oil_Pressure'] >=
oil_threshold_75)
Outlier_Coolant =(df['Coolant_Temp'] <= coolant_threshold_25) | (df['Coolant_Temp'] >=
coolant_threshold_75)
Outlier_ECU = (df['ECU_Temp'] <= ecu_threshold_25) | (df['ECU_Temp'] >=
ecu_threshold_75)
Outlier_PFA = (df['PF_Avg'] <= pf_threshold_25) | (df['PF_Avg'] >= pf_threshold_75)

#All parameter have Stat_Outlier condition TRUE then it is labelled as Outlier
```

```
df_Outlier = df[Outlier_Oil & Outlier_Coolant & Outlier_ECU & Outlier_PFA]
df_Outlier.insert(5, 'Label', 'Outlier', True)
print('Outlier data')
print(df_Outlier.head(5))
print('Outlier data size : ', df_Outlier.shape)
print(' ')

#Stat_Normal = 40% <= value <= 60%
Normal_Oil = df['Oil_Pressure'].between(oil_threshold_40, oil_threshold_60)
Normal_Coolant = df['Coolant_Temp'].between(coolant_threshold_40, coolant_threshold_60)
Normal_ECU = df['ECU_Temp'].between(ecu_threshold_40, ecu_threshold_60)
Normal_PFA = df['PF_Avg'].between(pf_threshold_40, pf_threshold_60)

#When any parameter have all Stat_Normal condition TRUE then it is labelled as Normal
df_Normal = df[Normal_Oil & Normal_Coolant & Normal_ECU & Normal_PFA]
df_Normal.insert(5, 'Label', 'Normal', True)
print('Normal data')
print(df_Normal.head(5))
print('Normal data size : ', df_Normal.shape)
print(' ')

#Stat_Maintenance = 25% < value < 40% or 60% < value < 75%
Maintenance_Oil = (df['Oil_Pressure'].between(oil_threshold_25, oil_threshold_40,
inclusive=False)) | (df['Oil_Pressure'].between(oil_threshold_60, oil_threshold_75,
inclusive=False))
Maintenance_Coolant = (df['Coolant_Temp'].between(coolant_threshold_25,
coolant_threshold_40, inclusive=False)) |
(df['Coolant_Temp'].between(coolant_threshold_60, coolant_threshold_75, inclusive=False))
Maintenance_ECU = (df['ECU_Temp'].between(ecu_threshold_25, ecu_threshold_40,
inclusive=False)) | (df['ECU_Temp'].between(ecu_threshold_60, ecu_threshold_75,
inclusive=False))
Maintenance_PFA = (df['PF_Avg'].between(pf_threshold_25, pf_threshold_40,
inclusive=False)) | (df['PF_Avg'].between(pf_threshold_60, pf_threshold_75,
inclusive=False))

#When any feature have Stat_Maintenance condition TRUE then it is labelled as Maintenance
df_Maintenance = df[Maintenance_Oil & Maintenance_Coolant & Maintenance_ECU &
Maintenance_PFA]
df_Maintenance.insert(5, 'Label', 'Maintenance', True)
print('Maintenance data')
print(df_Maintenance.head(5))
print('Maintenance data size : ', df_Maintenance.shape)
print(' ')

#Separate unlabelled data then export
df_Unlabelled = pd.concat([df, df_Outlier, df_Outlier,
                    df_Normal, df_Normal,
                    df_Maintenance, df_Maintenance]).drop_duplicates(keep=False)
df_Unlabelled.drop('Label',axis=1, inplace=True)
print('Unlabelled data')
print(df_Unlabelled.head(5))
print('Unlabelled data size : ', df_Unlabelled.shape)
df_Unlabelled.to_csv(process_dir+'unlabelled.csv')

#Training dataset 1 summary (include outlier)
```

```
#merge labelled data into training dataset
combine = pd.merge(df_Maintenance, df_Outlier, how = 'outer')
labeled_dataset1 = pd.merge(combine, df_Normal, how = 'outer')
print(labeled_dataset1)

#Visualize data
print('Training dataset size with outlier : ', labeled_dataset1.shape)
# class distribution
print(labeled_dataset1.groupby('Label').size())
print(' ')
# statistical value of data
print(labeled_dataset1.describe(percentiles=[.25, .4, .5, .6, .75]))
# box and whisker plots
labeled_dataset1.plot(kind='box', subplots=True, layout=(2,3), sharex=False, sharey=False)
pyplot.show()
# histograms
labeled_dataset1.hist()
pyplot.show()
#export df to csv
labeled_dataset1.to_csv(process_dir+'labeled_dataset1.csv')

#Training dataset 2 Summary (exclude outlier)

# merge labelled data into training dataset
labeled_dataset2 = pd.merge(df_Maintenance, df_Normal, how = 'outer')
print(labeled_dataset2)

#Visualize data
print('Training dataset size excluding outlier : ', labeled_dataset2.shape)
# class distribution
print(labeled_dataset2.groupby('Label').size())
print(' ')
# statistical value of data
print(labeled_dataset2.describe(percentiles=[.25, .4, .5, .6, .75]))
# box and whisker plots
labeled_dataset2.plot(kind='box', subplots=True, layout=(2,3), sharex=False, sharey=False)
pyplot.show()
# histograms
labeled_dataset2.hist()
pyplot.show()
#export df to csv
labeled_dataset2.to_csv(process_dir+'labeled_dataset2.csv')

"""### Filtered data for RUL"""

# Pre-processing
print('Total data size : ', df.shape)
df.drop_duplicates()
print('Removed duplicates data size : ', df.shape)

#Filter Outlier data from each feature
Filtered_Oil = df['Oil_Pressure'].between(oil_threshold_25, oil_threshold_75)
Filtered_Coolant = df['Coolant_Temp'].between(coolant_threshold_25,
coolant_threshold_75)
```

```
Filtered_ECU = df['ECU_Temp'].between(ecu_threshold_25, ecu_threshold_75)
Filtered_PFA = df['PF_Avg'].between(pf_threshold_25, pf_threshold_75)

df_Filtered = df[Filtered_Coolant & Filtered_ECU & Filtered_Oil & Filtered_PFA]
df_Filtered

# visualisasi dalam grafik
df_Filtered.hist()
pyplot.show()

# Create dataframe ONLY for Maintenance condition
#Stat_Maintenance = 25% < value < 40% or 60% < value < 75%
Maintenance_Oil = (df_Filtered['Oil_Pressure'].between(oil_threshold_25, oil_threshold_40,
inclusive=False)) | (df_Filtered['Oil_Pressure'].between(oil_threshold_60, oil_threshold_75,
inclusive=False))
Maintenance_Coolant = (df_Filtered['Coolant_Temp'].between(coolant_threshold_25,
coolant_threshold_40, inclusive=False)) |
(df_Filtered['Coolant_Temp'].between(coolant_threshold_60, coolant_threshold_75,
inclusive=False))
Maintenance_ECU = (df_Filtered['ECU_Temp'].between(ecu_threshold_25,
ecu_threshold_40, inclusive=False)) | (df_Filtered['ECU_Temp'].between(ecu_threshold_60,
ecu_threshold_75, inclusive=False))
Maintenance_PFA = (df_Filtered['PF_Avg'].between(pf_threshold_25, pf_threshold_40,
inclusive=False)) | (df_Filtered['PF_Avg'].between(pf_threshold_60, pf_threshold_75,
inclusive=False))

#When parameter have Stat_Maintenance condition TRUE then it is labelled as Maintenance
df_Maintenance = df_Filtered[Maintenance_Oil | Maintenance_Coolant | Maintenance_ECU |
Maintenance_PFA] # OR condition (choose one only)
#df_Maintenance = df_Filtered[Maintenance_Oil & Maintenance_Coolant &
Maintenance_ECU & Maintenance_PFA] # AND condition (choose one only)

df_Maintenance.insert(5, 'Label', 'Maintenance', True)
df_Maintenance = df_Maintenance.drop_duplicates(subset='Datetime')
print('Maintenance data size : ', df_Maintenance.shape)
df_Maintenance.head(5)

# Dataset for All vehicles
#calculate diffs from each row

#df_Maintenance['diffs'] =
df_Maintenance.sort_values(['Train_code','Datetime']).groupby('Train_code')['Datetime'].diff(
) #grouped by train code (Choose one)
df_Maintenance = df_Maintenance.sort_values(by='Datetime')
#NOT grouped (Choose one)
df_Maintenance['diffs'] = df_Maintenance['Datetime'].diff()
#NOT grouped (Choose one)
df_Maintenance['diffs'] = df_Maintenance['diffs'].fillna(pd.Timedelta(seconds=0))
df_Maintenance['Delta_Time_h'] = df_Maintenance.diffs/np.timedelta64(1,'h')
df_Maintenance

# Visualize Mean Time Between Maintenance (MTBM)
print(df_Maintenance.groupby('Train_code').Delta_Time_h.mean())
df_Maintenance.groupby('Train_code').Delta_Time_h.mean().plot(kind='bar')
```

```
#Export Dataset for All vehicles grouped by train code
df_Maintenance.to_csv(process_dir+'all_vehicles.csv')
```

8. Program Prediksi RUL

Semua kode dan file dapat diakses pada https://github.com/StudentHagal/Thesis

```
# -*- coding: utf-8 -*-
"""RUL Predict.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1bUkxdbRNquGV-Up7t5usmkwaiGNNvtnz

# 0) Preparation

## Load Requirements
"""

# Commented out IPython magic to ensure Python compatibility.
# Github repository
!git clone https://github.com/StudentHagal/Thesis.git

# Libraries
import sys
import pandas as pd
import hashlib
import matplotlib
import numpy as np
import pandas as pd
import sklearn
import joblib
import scipy
import matplotlib.pyplot as plt
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor


#Extension
!pip install ipython-autotime
# %load_ext autotime

print('load completed')

"""## Check Lib Version"""

# Check the versions of libraries
# !!! WARNING !!!
```

```
# Important because model result may be different for other version

print('Python: {}'.format(sys.version))
print('scipy: {}'.format(scipy.__version__))
print('numpy: {}'.format(np.__version__))
print('matplotlib: {}'.format(matplotlib.__version__))
print('pandas: {}'.format(pd.__version__))
print('sklearn: {}'.format(sklearn.__version__))
print('joblib: {}'.format(joblib.__version__))

"""## Global variables"""

# Global Variables declaration

#known hash value of file
compare = '10f68054068bc4ac4d7fee65e8fd7a184151e3ce3abce5d85350fc13c412d93b'
#Stored file path
url = "https://raw.githubusercontent.com/StudentHagal/Thesis/main/Resources/database.csv"
#environment path
code_dir = '/content/drive/MyDrive/Colab Notebooks/Thesis Repo/'
model_dir = '/content/drive/MyDrive/Colab Notebooks/ML Resources/models/'
process_dir = '/content/drive/MyDrive/Colab Notebooks/ML Resources/Processed Data/'
#Column names for dataset
col_names = ['No', 'Datetime', 'Control_Mode', 'L1', 'L2', 'L3', 'Frequency', 'kVA_Total',
            'kVA_L1', 'kVA_L2', 'kVA_L3', 'KVAr', 'Oil_Pressure', 'Coolant_Temp',
            'Charger_Alternator', 'PF_Avg', 'PF_L1', 'PF_L2',
            'PF_L3', 'L1_N', 'L2_N', 'L3_N', 'Source_Ext_Voltage', 'ECU_Temp',
            'RPM', 'Train_code']
df = pd.read_csv(url, names=col_names)
features = ['Oil_Pressure', 'Coolant_Temp', 'PF_Avg', 'ECU_Temp']
#Unused columns in dataset
unused_col = ['No', 'Control_Mode', 'L1', 'L2', 'L3', 'Frequency',
             'kVA_Total', 'kVA_L1', 'kVA_L2', 'kVA_L3', 'KVAr', 'Charger_Alternator',
             'RPM', 'PF_L1', 'PF_L2', 'PF_L3', 'L1_N','L2_N', 'L3_N',
             'Source_Ext_Voltage']

#Threshold value variable 40% min, 60% max for labelling
oil_threshold_40, oil_threshold_60 = df.Oil_Pressure.quantile([0.4 , 0.6])
coolant_threshold_40, coolant_threshold_60 = df.Coolant_Temp.quantile([0.4 , 0.6])
pf_threshold_40, pf_threshold_60 = df.PF_Avg.quantile([0.4 , 0.6])
ecu_threshold_40, ecu_threshold_60 = df.ECU_Temp.quantile([0.4 , 0.6])
#Threshold value variable 25% min, 75% max for labelling
oil_threshold_25, oil_threshold_75 = df.Oil_Pressure.quantile([0.25 , 0.75])
coolant_threshold_25, coolant_threshold_75 = df.Coolant_Temp.quantile([0.25 , 0.75])
pf_threshold_25, pf_threshold_75 = df.PF_Avg.quantile([0.25 , 0.75])
ecu_threshold_25, ecu_threshold_75 = df.ECU_Temp.quantile([0.25 , 0.75])

#Algorithm Selection
model_RFreg = RandomForestRegressor(random_state=1)
model_OLS = LinearRegression()
model_Ridge = Ridge()

#Model Evaluation var
val_results = []
names = []
```

```
models = []
models.append(('OLS', model_OLS))
models.append(('Ridge', model_Ridge))
models.append(('RF-reg', model_RFreg))

"""## Hash Check Function"""

# Function that returns the SHA-2 hash of the file
def hash_file(filepath):

  # make a hash object with SHA-2
  h = hashlib.sha256()

  # open file for reading in binary mode
  with open(filepath,'rb') as file:
     # loop till the end of the file
     chunk = 0
     while chunk != b'':
        # read only 1024 bytes at a time
        chunk = file.read(1024)
        h.update(chunk)

  # return the hex representation of digest
  return h.hexdigest()

"""# 1) Processing Dataset
Calculate Mean Time Between Maintenance (MTBM)

## Hash Validation
"""

# Checking hash value of all vehicles dataset file
message = hash_file(process_dir+'all_vehicles.csv')
print('SHA-256 value of all vehicles dataset file is :')
print(message)

# Checking hash value of ML labeled dataset file
message = hash_file(process_dir+'predict1_smote.csv')
print('SHA-256 value of ML labeled dataset file is :')
print(message)

"""## All vehicles"""

#load dataset for all vehicles
df_allvehicles = pd.read_csv(process_dir+'all_vehicles.csv', index_col=0)
MTBM_allvehicles = df_allvehicles.Delta_Time_h.mean()
print('Mean Time Between Maintenance : ',MTBM_allvehicles)
df_allvehicles

df_allvehicles.describe()

# Simple Scatterplot
plt.scatter(df_allvehicles.Delta_Time_h, df_allvehicles.Oil_Pressure)
plt.title('Scatter plot')
plt.xlabel('Delta_Time (h)')
```

```
plt.ylabel('Oil Pressure')
plt.show()

"""## Single vehicle"""

#Preprocess data of a single vehicle
df_single = df_allvehicles.loc[df_allvehicles['Train_code'] == 'P-01908']
MTBM_single = df_single.Delta_Time_h.mean()
print('Mean Time Between Maintenance : ',MTBM_single)
df_single

df_single.describe()

# Simple Scatterplot
plt.scatter( df_single.Delta_Time_h, df_single.Oil_Pressure)
plt.title('Scatter plot P-01908')
plt.xlabel('Delta_Time (h)')
plt.ylabel('Oil Pressure')
plt.show()

"""## ML Label"""

#Preprocess dataset from ML classifier
df_MLlabel = pd.read_csv(process_dir+'predict1_smote.csv')
dropped_col = ['Unnamed: 0','LR', 'KNN', 'SVM', 'RF', 'LDA', 'NB']
df_MLlabel.drop(labels=dropped_col, axis=1, inplace=True)
df_MLlabel = df_MLlabel.loc[df_MLlabel['DT'] == 'Maintenance']
df_MLlabel['Datetime'] = pd.to_datetime(df_MLlabel.Datetime)
df_MLlabel = df_MLlabel.sort_values(by='Datetime')
df_MLlabel['Delta_Time'] = df_MLlabel.Datetime.diff()
df_MLlabel = df_MLlabel.fillna(pd.Timedelta(seconds=0))
df_MLlabel['Delta_Time_h'] = df_MLlabel['Delta_Time']/np.timedelta64(1,'h')
MTBM_MLlabel = df_MLlabel.Delta_Time_h.mean()
print('Mean Time Between Maintenance : ',MTBM_MLlabel)
df_MLlabel

df_MLlabel.describe()

# Simple Scatterplot
plt.scatter(df_MLlabel.Delta_Time_h,df_MLlabel.Oil_Pressure )
plt.title('Scatter plot ML Classifier')
plt.xlabel('Timedelta (h)')
plt.ylabel('Oil Pressure')
plt.show()

"""## Dataset Split"""

#Data splitting

#separate feature (X) and predicted (y) parameters for all vehicles
X = df_allvehicles[features]
y = df_allvehicles['Delta_Time_h']
# Split-out training and test dataset for all vehicles
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.20,
                                    random_state=1)
```

```
print('Training data size = ', X_train.shape)
print('Testing data size = ', X_test.shape)


#separate feature (X) and predicted (y) parameters for single vehicle
X1 = df_single[features]
y1 = df_single['Delta_Time_h']
# Split-out training and test dataset for single vehicle
X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1, y1, test_size=0.20,
                                    random_state=1)
print('Training data size (single vehicle) = ', X1_train.shape)
print('Testing data size (single vehicle) = ', X1_test.shape)


#separate feature (X) and predicted (y) parameters for ML classifier
X2 = df_MLlabel[features]
y2 = df_MLlabel['Delta_Time_h']
# Split-out training and test dataset for ML classifier
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X2, y2, test_size=0.20,
                                    random_state=1)
print('Training data size (ML Classifier) = ', X2_train.shape)
print('Testing data size (ML Classifier) = ', X2_test.shape)

print('Data split completed')

"""# 3) Model Training

## Model Training & Validation
"""

# Cross validation of each model training in turn
# Change variable X_train and Y_train if using other dataset (Check Dataset Split part for
variables used)
for name, model in models:
        kfold = KFold(n_splits=10, random_state=1, shuffle=True)
        cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
scoring='neg_mean_absolute_error')
 #MEA scoring will be shown in negative, due to how cross_val_score scoring function
works. The closer to 0 is better
        val_results.append(cv_results)
        names.append(name)
        print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

# Compare result
pyplot.boxplot(val_results, labels=names)
pyplot.title('Algorithm MEA Comparison')
pyplot.show()

# Re-train each algorithm using whole training dataset for better model
# Change variable X_train and Y_train if using other dataset
# Each model is separated in different code block for easier tracking of Runtime
model_OLS.fit(X_train, Y_train)
joblib.dump(model_OLS, model_dir+'model_OLS.joblib')

model_Ridge.fit(X_train, Y_train)
```

```
joblib.dump(model_Ridge, model_dir+'model_Ridge.joblib')

model_RFreg.fit(X_train, Y_train)
joblib.dump(model_RFreg, model_dir+'model_RFreg.joblib')

"""# 4) Model Testing"""

#RUL result is MTBM_dataset-Prediction
#Change MTBM_dataset and Y_test variable depend on dataset:
#MTBM_allvehicles, Y_test for All vehicles dataset,
#MTBM_single, Y1_test for Single vehicle dataset,
#MTBM_MLlabel, Y2_test for ML labelled dataset

predictions_OLS = model_OLS.predict(X_test)
print('OLS MEA Result:')
print(mean_absolute_error(Y_test, predictions_OLS))

predictions_Ridge = model_Ridge.predict(X_test)
print('Ridge MEA Result:')
print(mean_absolute_error(Y_test, predictions_Ridge))

predictions_RFreg = model_RFreg.predict(X_test)
print('RF-Reg MEA Result:')
print(mean_absolute_error(Y_test, predictions_RFreg))

df_RUL = pd.DataFrame({'OLS': MTBM_allvehicles-predictions_OLS, 'Ridge':
MTBM_allvehicles-predictions_Ridge,
                       'RF-reg':MTBM_allvehicles-predictions_RFreg})
df_RUL
```

9. Program Prediksi Input Pengguna
Semua kode dan file dapat diakses pada https://github.com/StudentHagal/Thesis

```
# -*- coding: utf-8 -*-
"""User Prediction Test.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/13bHySJTqdBA2xuDUINkJ2EvBnR3m-dVx

# 0) Preparation

## Load Requirements
"""

# Commented out IPython magic to ensure Python compatibility.
# Libraries
import sys
import numpy as np
import pandas as pd
import sklearn
import joblib
import hashlib
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

#Extension
!pip install ipython-autotime
# %load_ext autotime

print('Load completed')

# Check the versions of libraries
# !!! WARNING !!!
# Important because model result may be different for other version

print('Python: {}'.format(sys.version))
print('numpy: {}'.format(np.__version__))
print('pandas: {}'.format(pd.__version__))
print('sklearn: {}'.format(sklearn.__version__))
print('joblib: {}'.format(joblib.__version__))

"""## Global Variables"""

#Global variables declaration

#Column names for dataset
```

```
dir = '/content/drive/MyDrive/Colab Notebooks/ML Resources/models/'
process_dir = '/content/drive/MyDrive/Colab Notebooks/ML Resources/Processed Data/'
col_names = ['No', 'Datetime', 'Control_Mode', 'L1', 'L2', 'L3', 'Frequency', 'kVA_Total',
           'kVA_L1', 'kVA_L2', 'kVA_L3', 'KVAr', 'Oil_Pressure', 'Coolant_Temp',
           'Charger_Alternator', 'PF_Avg', 'PF_L1', 'PF_L2',
           'PF_L3', 'L1_N', 'L2_N', 'L3_N', 'Source_Ext_Voltage', 'ECU_Temp',
           'RPM', 'Train_code']
features = ['Oil_Pressure', 'Coolant_Temp', 'PF_Avg', 'ECU_Temp']
#Unused columns in dataset
unused_col = ['No', 'Datetime', 'Control_Mode', 'L1', 'L2', 'L3', 'Frequency',
            'kVA_Total', 'kVA_L1', 'kVA_L2', 'kVA_L3', 'KVAr', 'Charger_Alternator',
            'RPM', 'PF_L1', 'PF_L2', 'PF_L3', 'L1_N','L2_N', 'L3_N',
            'Source_Ext_Voltage', 'Train_code']

#Import and load Model
Opsi1_LR = joblib.load(dir+'Opsi1_LR.joblib')
Opsi1_KNN = joblib.load(dir+'Opsi1_KNN.joblib')
Opsi1_DT = joblib.load(dir+'Opsi1_DT.joblib')
Opsi1_SVM = joblib.load(dir+'Opsi1_SVM.joblib')
Opsi1_RF = joblib.load(dir+'Opsi1_RF.joblib')
Opsi1_LDA = joblib.load(dir+'Opsi1_LDA.joblib')
Opsi1_NB = joblib.load(dir+'Opsi1_NB.joblib')

Opsi2_LR = joblib.load(dir+'Opsi2_LR.joblib')
Opsi2_KNN = joblib.load(dir+'Opsi2_KNN.joblib')
Opsi2_DT = joblib.load(dir+'Opsi2_DT.joblib')
Opsi2_SVM = joblib.load(dir+'Opsi2_SVM.joblib')
Opsi2_RF = joblib.load(dir+'Opsi2_RF.joblib')
Opsi2_LDA = joblib.load(dir+'Opsi2_LDA.joblib')
Opsi2_NB = joblib.load(dir+'Opsi2_NB.joblib')

Opsi3_LR = joblib.load(dir+'Opsi3_LR.joblib')
Opsi3_KNN = joblib.load(dir+'Opsi3_KNN.joblib')
Opsi3_DT = joblib.load(dir+'Opsi3_DT.joblib')
Opsi3_SVM = joblib.load(dir+'Opsi3_SVM.joblib')
Opsi3_RF = joblib.load(dir+'Opsi3_RF.joblib')
Opsi3_LDA = joblib.load(dir+'Opsi3_LDA.joblib')
Opsi3_NB = joblib.load(dir+'Opsi3_NB.joblib')

Opsi4_LR = joblib.load(dir+'Opsi4_LR.joblib')
Opsi4_KNN = joblib.load(dir+'Opsi4_KNN.joblib')
Opsi4_DT = joblib.load(dir+'Opsi4_DT.joblib')
Opsi4_SVM = joblib.load(dir+'Opsi4_SVM.joblib')
Opsi4_RF = joblib.load(dir+'Opsi4_RF.joblib')
Opsi4_LDA = joblib.load(dir+'Opsi4_LDA.joblib')
Opsi4_NB = joblib.load(dir+'Opsi4_NB.joblib')

RUL_OLS = joblib.load(dir+'model_OLS.joblib')
RUL_Ridge = joblib.load(dir+'model_Ridge.joblib')
RUL_RFreg = joblib.load(dir+'model_RFreg.joblib')

"""## Function Declaration"""

# Function that returns the SHA-2 hash of the file
def hash_file(filepath):
```

```
    # make a hash object with SHA-2
    h = hashlib.sha256()

    # open file for reading in binary mode
    with open(filepath,'rb') as file:
        # loop till the end of the file
        chunk = 0
        while chunk != b'':
            # read only 1024 bytes at a time
            chunk = file.read(1024)
            h.update(chunk)

    # return the hex representation of digest
    return h.hexdigest()

"""# 1) Prediction Test

## Manual Input
"""

# user input manually for each sensor value
print('-----Prediction Test on ML Model-----')
print('Enter value for Oil_Pressure sensor:')
feature1 = float(input())
print('Enter value for Coolant_Temp sensor:')
feature2 = float(input())
print('Enter value for PF_Avg sensor:')
feature3 = float(input())
print('Enter value for ECU_Temp sensor:')
feature4 = float(input())

df_input = pd.DataFrame({'Oil_Pressure': [feature1], 'Coolant_Temp': [feature2],
                         'PF_Avg':[feature3], 'ECU_Temp':[feature4]})

df_input

"""### Opsi 1"""

# print prediction result for each model from user input
predictions_LR = Opsi1_LR.predict(df_input)
predictions_KNN = Opsi1_KNN.predict(df_input)
predictions_DT = Opsi1_DT.predict(df_input)
predictions_SVM = Opsi1_SVM.predict(df_input)
predictions_RF = Opsi1_RF.predict(df_input)
predictions_LDA = Opsi1_LDA.predict(df_input)
predictions_NB = Opsi1_NB.predict(df_input)

df_predict = pd.DataFrame({'LR': predictions_LR, 'KNN': predictions_KNN,
                           'DT':predictions_DT, 'SVM':predictions_SVM,
                           'RF':predictions_RF, 'LDA':predictions_LDA,
                           'NB':predictions_NB})
df_predict

"""### Opsi 2"""
```

```
# print prediction result for each model from user input
predictions_LR = Opsi2_LR.predict(df_input)
predictions_KNN = Opsi2_KNN.predict(df_input)
predictions_DT = Opsi2_DT.predict(df_input)
predictions_SVM = Opsi2_SVM.predict(df_input)
predictions_RF = Opsi2_RF.predict(df_input)
predictions_LDA = Opsi2_LDA.predict(df_input)
predictions_NB = Opsi2_NB.predict(df_input)

df_predict = pd.DataFrame({'LR': predictions_LR, 'KNN': predictions_KNN,
                    'DT':predictions_DT, 'SVM':predictions_SVM,
                    'RF':predictions_RF, 'LDA':predictions_LDA,
                    'NB':predictions_NB})
df_predict

"""### Opsi 3"""

# print prediction result for each model from user input
predictions_LR = Opsi3_LR.predict(df_input)
predictions_KNN = Opsi3_KNN.predict(df_input)
predictions_DT = Opsi3_DT.predict(df_input)
predictions_SVM = Opsi3_SVM.predict(df_input)
predictions_RF = Opsi3_RF.predict(df_input)
predictions_LDA = Opsi3_LDA.predict(df_input)
predictions_NB = Opsi3_NB.predict(df_input)

df_predict = pd.DataFrame({'LR': predictions_LR, 'KNN': predictions_KNN,
                    'DT':predictions_DT, 'SVM':predictions_SVM,
                    'RF':predictions_RF, 'LDA':predictions_LDA,
                    'NB':predictions_NB})
df_predict

"""### Opsi 4"""

# print prediction result for each model from user input
predictions_LR = Opsi4_LR.predict(df_input)
predictions_KNN = Opsi4_KNN.predict(df_input)
predictions_DT = Opsi4_DT.predict(df_input)
predictions_SVM = Opsi4_SVM.predict(df_input)
predictions_RF = Opsi4_RF.predict(df_input)
predictions_LDA = Opsi4_LDA.predict(df_input)
predictions_NB = Opsi4_NB.predict(df_input)

df_predict = pd.DataFrame({'LR': predictions_LR, 'KNN': predictions_KNN,
                    'DT':predictions_DT, 'SVM':predictions_SVM,
                    'RF':predictions_RF, 'LDA':predictions_LDA,
                    'NB':predictions_NB})
df_predict

"""### RUL"""

#RUL result is MTBF_Condition-Prediction (differ with each dataset)
predictions_OLS = RUL_OLS.predict(df_input)
predictions_Ridge = RUL_Ridge.predict(df_input)
```

```
predictions_RFreg = RUL_RFreg.predict(df_input)

df_RUL = pd.DataFrame({'OLS': 3.914026-predictions_OLS, 'Ridge': 3.914026-
predictions_Ridge,
                       'RF-reg':3.914026-predictions_RFreg})
df_RUL

"""## From csv file"""

#user input path to file
print('-----Prediction Test on ML Model-----')
print('Enter value for File path: ')
file_path = str(input())

hash_file(file_path)

df_file = pd.read_csv(file_path, names=col_names)
df_file

df_file.drop(labels=unused_col, axis=1, inplace=True)
df_file

df_file = df_file.dropna()

df_file.info()

"""### Opsi 1"""

# print prediction result for each model from user input
predictions_LR = Opsi1_LR.predict(df_file)
predictions_KNN = Opsi1_KNN.predict(df_file)
predictions_DT = Opsi1_DT.predict(df_file)
predictions_SVM = Opsi1_SVM.predict(df_file)
predictions_RF = Opsi1_RF.predict(df_file)
predictions_LDA = Opsi1_LDA.predict(df_file)
predictions_NB = Opsi1_NB.predict(df_file)

df_predict = pd.DataFrame({'LR': predictions_LR, 'KNN': predictions_KNN,
                       'DT':predictions_DT, 'SVM':predictions_SVM,
                       'RF':predictions_RF, 'LDA':predictions_LDA,
                       'NB':predictions_NB})
df_predict

#Export prediction result
df_result1 = pd.concat([df_file.reset_index(drop=True),
df_predict.reset_index(drop=True)],axis=1)
df_result1.to_csv(process_dir+'opsi1_classification.csv')
df_result1

"""### Opsi 2"""

# print prediction result for each model from user input
predictions_LR = Opsi2_LR.predict(df_file)
predictions_KNN = Opsi2_KNN.predict(df_file)
predictions_DT = Opsi2_DT.predict(df_file)
```

```
predictions_SVM = Opsi2_SVM.predict(df_file)
predictions_RF = Opsi2_RF.predict(df_file)
predictions_LDA = Opsi2_LDA.predict(df_file)
predictions_NB = Opsi2_NB.predict(df_file)

df_predict = pd.DataFrame({'LR': predictions_LR, 'KNN': predictions_KNN,
                    'DT':predictions_DT, 'SVM':predictions_SVM,
                    'RF':predictions_RF, 'LDA':predictions_LDA,
                    'NB':predictions_NB})
df_predict

#Export prediction result
df_result2 = pd.concat([df_file.reset_index(drop=True),
df_predict.reset_index(drop=True)],axis=1)
df_result2.to_csv(process_dir+'opsi2_classification.csv')
df_result2

"""### Opsi 3"""

# print prediction result for each model from user input
predictions_LR = Opsi3_LR.predict(df_file)
predictions_KNN = Opsi3_KNN.predict(df_file)
predictions_DT = Opsi3_DT.predict(df_file)
predictions_SVM = Opsi3_SVM.predict(df_file)
predictions_RF = Opsi3_RF.predict(df_file)
predictions_LDA = Opsi3_LDA.predict(df_file)
predictions_NB = Opsi3_NB.predict(df_file)

df_predict = pd.DataFrame({'LR': predictions_LR, 'KNN': predictions_KNN,
                    'DT':predictions_DT, 'SVM':predictions_SVM,
                    'RF':predictions_RF, 'LDA':predictions_LDA,
                    'NB':predictions_NB})
df_predict

#Export prediction result
df_result3 = pd.concat([df_file.reset_index(drop=True),
df_predict.reset_index(drop=True)],axis=1)
df_result3.to_csv(process_dir+'opsi3_classification.csv')
df_result3

"""### Opsi 4"""

# print prediction result for each model from user input
predictions_LR = Opsi4_LR.predict(df_file)
predictions_KNN = Opsi4_KNN.predict(df_file)
predictions_DT = Opsi4_DT.predict(df_file)
predictions_SVM = Opsi4_SVM.predict(df_file)
predictions_RF = Opsi4_RF.predict(df_file)
predictions_LDA = Opsi4_LDA.predict(df_file)
predictions_NB = Opsi4_NB.predict(df_file)

df_predict = pd.DataFrame({'LR': predictions_LR, 'KNN': predictions_KNN,
                    'DT':predictions_DT, 'SVM':predictions_SVM,
                    'RF':predictions_RF, 'LDA':predictions_LDA,
                    'NB':predictions_NB})
```

```
df_predict

#Export prediction result
df_result4 = pd.concat([df_file.reset_index(drop=True),
df_predict.reset_index(drop=True)],axis=1)
df_result4.to_csv(process_dir+'opsi4_classification.csv')
df_result4

"""### RUL"""

print('Enter value for MTBM:')
MTBM = float(input())

#RUL result is MTBM_Condition-Prediction (differ with each dataset)
predictions_OLS = RUL_OLS.predict(df_file)
predictions_Ridge = RUL_Ridge.predict(df_file)
predictions_RFreg = RUL_RFreg.predict(df_file)

df_RUL = pd.DataFrame({'OLS': MTBM-predictions_OLS, 'Ridge': MTBM-
predictions_Ridge,
                       'RF-reg': MTBM-predictions_RFreg})
df_RUL

#Export prediction result
df_resultRUL = pd.concat([df_file.reset_index(drop=True),
df_RUL.reset_index(drop=True)],axis=1)
df_resultRUL.to_csv(process_dir+'RUL_regression.csv')
df_resultRUL
```