IMPLEMENTASI SPARK MLLIB PADA PENGOLAHAN DATA STREAM UNTUK PRAKIRAAN TINGKAT KONSENTRASI CO_2

SKRIPSI

Diajukan sebagai salah satu syarat untuk menyelesaikan
pendidikan diploma empat (D-4) Program Studi Teknik Komputer dan Jaringan
Jurusan Teknik Elektro
Politeknik Negeri Ujung Pandang

ARNI SEPTIANI
42518005

PROGRAM STUDI D-4 TEKNIK KOMPUTER DAN JARINGAN

JURUSAN TEKNIK ELEKTRO

POLITEKNIK NEGERI UJUNG PANDANG MAKASSAR

2023

HALAMAN PENGESAHAN

Skripsi dengan judul **Implementasi Spark Mllib Pada Pengolahan Data Stream Untuk Prakiraan Tingkat Konsentrasi CO_2** oleh Arni Septiani NIM 42518005 dinyatakan layak untuk diujikan.

Makassar, 27 April 2023

Pembimbing I,

Zawiyah Saharuna,S.T., M.Eng
NIP 198309032014042001

Pembimbing II,

Ir.Dahlia,M.T.
NIP 196412311991032003

Mengetahui
Ketua Program Studi,
Teknik Komputer dan Jaringan
Politeknik Negeri Ujung Pandang

Eddy Tungadi,S.T., M.T.
NIP 197908232010121001

HALAMAN PENERIMAAN

Pada hari ini, Jumat tanggal 12 Mei 2023, Tim Penguji Ujian Sidang Skripsi telah menerima dengan baik skripsi oleh mahasiswa: Arni Septiani NIM 42518005 dengan judul **Implementasi Spark Mllib Pada Pengolahan Data Stream Untuk Prakiraan Tingkat Konsentrasi CO_2.**

Makassar, 12 Mei 2023

Tim Penguji Ujian Sidang Skripsi:

1. Iin Karmila Yusri, S.ST., M.Eng., Ph.D.	Ketua	(.....)
2. Meylanie Olivya,S.T.,M.T.	Sekretaris	(.....)
3. Rini Nur,S.T., M.T.	Anggota	(.....)
4. Drs.Kasim,M.T.	Anggota	(.....)
5. Zawiyah Saharuna,S.T., M.Eng.	Anggota	(.....)
6. Ir.Dahlia,M.T.	Anggota	(.....)

KATA PENGANTAR

Puji dan syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa yang telah melimpahkan rahmat dan karunia-Nya, sehingga penulisan skripsi ini yang berjudul "IMPLEMENTASI SPARK MLLIB PADA PENGOLAHAN DATA STREAM UNTUK PRAKIRAAN TINGKAT KONSENTRASI CO_2" dapat diselesaikan dengan baik.

Skripsi ini disusun guna memenuhi salah satu syarat untuk menyelesaikan studi serta dalam rangka memperoleh gelar Diploma IV (D-4/S1 Terapan) pada Program Studi Teknik Komputer dan Jaringan Jurusan Teknik Elektro Politeknik Negeri Ujung Pandang.

Penulis menyadari bahwa keberhasilan penyusunan skripsi ini tidak lepas dari bantuan berbagai pihak baik secara langsung maupun tidak langsung. Oleh karena itu, dengan rendah hati penulis mengucapkan terima kasih kepada:

1. Kedua orang tua penulis yakni Pelipus, S.Pd dan Monika Mani', Saudara penulis yakni Venina Mani' dan Tirsa Triani serta keluarga besar penulis yang telah memberikan semangat, motivasi, dukungan, bimbingan, dan doa kepada penulis.
2. Teman-teman penulis yang sudah menyemangati bahkan ikut membantu penyelesaian skripsi ini. Terlebih khusus kepada Kisma, Bima, dan Elu yang selalu meluangkan waktunya kepada penulis.

3. Ibu Zawiyah Saharuna, S.T., M.Eng. selaku pembimbing I dan Ir. Dahlia Nur, M.T. selaku pembimbing II yang telah mencurahkan waktu dan kesempatannya untuk mengarahkan penulis dalam menyelesaikan skripsi ini.
4. Bapak Ahmad Rizal Sultan, S.T., M.T., Ph.D. selaku Ketua Jurusan Teknik Elektro Politeknik Negeri Ujung Pandang.
5. Bapak Eddy Tungadi, S.T., M.T. selaku Koordinator Program Studi Teknik Komputer dan Jaringan.
6. Seluruh dosen dan Staf Jurusan Teknik Elektro, Khususnya Program Studi D4 Teknik Komputer dan Jaringan.
7. Teman-teman seperjuangan di Program Studi D-4 Teknik Komputer dan Jaringan angkatan 2018 yang mempunyai peranan besar dalam membantu menyusun skripsi ini dan mengajarkan banyak hal kepada penulis baik dari segi akademik maupun non akademik.
8. Semua pihak yang telah memberikan bantuan moril maupun materil yang tidak dapat disebutkan satu per satu.

Penulis menyadari bahwa skripsi ini masih kurang dari kata sempurna, sehingga penulis mengharapkan kritik dan saran yang sifatnya membangun untuk perbaikan di masa mendatang. Semoga tulisan ini bermanfaat.

Makassar, 27 April 2023

Penulis

DAFTAR ISI

POLITEKNIK NEGERI
UJUNG PANDANG

DAFTAR TABEL

POLITEKNIK NEGERI
UJUNG PANDANG

DAFTAR GAMBAR

DAFTAR LAMPIRAN

POLITEKNIK NEGERI
UJUNG PANDANG

SURAT PERNYATAAN

Saya yang bertanda tangan dibawah ini:

Nama : Arni Septiani

NIM : 42518005

Menyatakan dengan sebenar-benarnya bahwa segala pernyataan dalam skripsi ini yang berjudul **Implementasi Spark Mllib Pada Pengolahan Data Stream Untuk Prakiraan Tingkat Konsentrasi CO_2** merupakan gagasan dan hasil karya saya sendiri dengan arahan komisi pembimbing, dan belum pernah diajukan dalam bentuk apapun pada perguruan tinggi dan instansi manapun.

Semua data dan informasi yang digunakan telah dinyatakan secara jelas dan dapat diperiksa kebenarannya. Sumber informasi yang berasal atau dikutip dari karya yang diterbitkan dari penulis lain telah disebutkan dalam naskah dan dicantumkan dalam skripsi ini.

Jika pernyataan saya tersebut diatas tidak benar, saya siap menanggung resiko yang ditetapkan oleh Politeknik Negeri Ujung Pandang.

Makassar, 27 April 2023

Arni Septiani
NIM 42518005

Implementasi Spark Mllib Pada Pengolahan Data Stream Untuk Prakiraan Tingkat Konsentrasi Co2

RINGKASAN

Kualitas udara dalam ruangan atau *indoor air quality* (IAQ) merupakan salah satu dari lima risiko lingkungan paling mendesak bagi kesehatan masyarakat. Hal ini disebabkan karena sebagian besar masyarakat menghabiskan waktu di dalam ruangan, baik itu di rumah, kantor, sekolah, fasilitas kesehatan, atau tempat publik lainnya. Salah satu polutan utama yang paling mempengaruhi tingkat kualitas udara di dalam ruangan adalah karbondioksida (CO_2). Peningkatan kadar CO_2 lebih dari 450-1000 ppm di dalam ruangan akan memberikan dampak terhadap produktivitas penghuni ruangan, terutama gangguan pada kemampuan kognitif, pengambilan keputusan, dan pada tingkat konsentrasi yang sangat tinggi dapat menyebabkan hilangnya kesadaran. Oleh karena itu, kadar CO_2 di dalam ruangan harus selalu diukur dan dijaga agar tidak melebihi level yang ditentukan. Untuk mengukur dan menjaga kadar CO_2, dibutuhkan sebuah metode yang dapat melakukan prakiraan CO_2 secara *realtime* sehingga dibutuhkan metode pengolahan data secara secara *streaming*. Pengolahan *data stream* dapat dilakukan menggunakan beberapa *tools*, salah satunya adalah *Apache Spark. Apache Spark* bersifat *open source* dan memiliki *library Spark Streaming* yang dapat digunakan untuk memproses data secara *real time.* Selain itu, *Apache Spark* juga dapat digunakan untuk melakukan prakiraan data menggunakan algoritma *machine learning* yang disediakan oleh *library Spark MLlib*. *Spark MLlib* dapat bekerja sama dengan *Spark Streaming* serta memiliki kemampuan dan kecepatan yang sangat baik untuk mengolah data karena pemrosesan data pada *Spark MLlib* dilakukan di memori. Salah satu algoritma yang dapat digunakan untuk melakukan prakiraan pada *Spark MLlib* adalah algoritma *Random Forest*. Pada penelitian ini akan dilakukan prakiraan kadar CO_2 di dalam ruangan menggunakan algoritma *Random Forest* pada *library Spark MLlib* secara *stream* agar hasil prakiraan dapat diperoleh secara kontinu. Untuk mendapatkan nilai terbaik dari prakiraan yang dilakukan, digunakan parameter *windows size* karena dapat berpengaruh terhadap akurasi hasil suatu prakiraan. Hasil penelitian ini berhasil meramalkan CO_2 baik itu pada kadar normal maupun di atas kadar normal. Nilai terbaik didapat menggunakan *windows size* 13 yang memperoleh RMSE sebesar 12,0045 dan MAPE sebesar 1,616489171%.

Kata Kunci: CO_2, *Spark Streaming*, *Spark MLlib*, *Windows Size*

BAB I. PENDAHULUAN

1.1 Latar Belakang

Kualitas udara dalam ruangan atau *indoor air quality* (IAQ) merupakan salah satu hal yang dapat mempengaruhi kesehatan penghuni ruangan (Dewi et al., 2021). Badan Perlindungan Lingkungan A.S. mengidentifikasi IAQ sebagai salah satu dari lima risiko lingkungan paling mendesak bagi kesehatan masyarakat. Hal ini disebabkan karena sebagian besar masyarakat menghabiskan waktu di dalam ruangan, baik itu di rumah, kantor, sekolah, fasilitas kesehatan, atau tempat publik lainnya sehingga penting untuk menjamin kualitas udara dalam ruangan tetap aman dan nyaman bagi penghuninya (Dewi et al., 2021). Salah satu polutan utama yang paling mempengaruhi tingkat kualitas udara di dalam ruangan adalah karbondioksida (CO_2).

Peningkatan kadar CO_2 lebih dari 450-1000 ppm di dalam ruangan akan memberikan dampak terhadap produktivitas penghuni ruangan, terutama gangguan pada kemampuan kognitif, pengambilan keputusan, dan pada tingkat konsentrasi yang sangat tinggi dapat menyebabkan hilangnya kesadaran (Lazovic et al., 2016). Di sisi lain, pemerintah juga sudah mengeluarkan syarat kadar aman CO_2 di dalam yaitu 1000 ppm dalam 8 jam sesuai dengan Peraturan Menteri Kesehatan Republik Indonesia No 1077 Tentang Pedoman Penyehatan Udara Di Dalam Ruangan. Oleh karena itu, kadar CO_2 di dalam ruangan harus selalu diukur dan dijaga agar tidak melebihi level yang ditentukan. Untuk mengukur dan menjaga kadar CO_2, dibutuhkan sebuah metode yang dapat melakukan prakiraan CO_2 secara *realtime*.

Untuk melakukan prakiraan secara *realtime*, dibutuhkan pengolahan data secara *streaming*.

Pengolahan data secara *streaming* memungkinkan pembuatan keputusan dapat dilakukan saat *runtime*, sehingga proses analisis data untuk mendapatkan informasi tidak memakan waktu yang lama (Ebada et al., 2022). Pengolahan *data stream* dapat dilakukan menggunakan beberapa *tools*, salah satunya adalah *Apache Spark* (Hassan et al., 2020). *Apache Spark* bersifat *open source* dan memiliki *library Spark Streaming* yang dapat digunakan untuk memproses data secara *real time*. *Spark Streaming* akan berkomunikasi dengan sumber data, kemudian menerima data dari sumber tersebut dan memprosesnya secara terus menerus. Selain itu, *Apache Spark* juga dapat digunakan untuk melakukan prakiraan data menggunakan algoritma *machine learning* yang disediakan oleh *library Spark MLlib*.

Spark MLlib adalah *library machine learning* pada *Apache Spark* yang memiliki kemampuan dan kecepatan yang sangat baik untuk mengolah data karena pemrosesan data pada *Spark MLlib* dilakukan di memori (Ebada et al., 2022). Selain itu, *Library Spark MLlib* dapat bekerja sama dengan *Spark Streaming* sehingga memungkinkan untuk melakukan pengolahan *machine learning* secara *stream* tanpa membangun banyak arsitektur (Omran et al., 2021). *Spark MLlib* menyediakan berbagai fungsi yang dapat dipanggil untuk melakukan pembelajaran *Supervised* maupun *Un-supervised*, termasuk algoritma untuk melakukan prakiraan *time series* menggunakan metode *Regression*. Salah satu algoritma prakiraan *time series* yang dapat digunakan adalah algoritma *Random Forest*.

Oleh karena itu pada penelitian ini akan dilakukan prakiraan kadar CO_2 di dalam ruangan dengan menggunakan algoritma *Random Forest* pada *library Spark MLlib*. Prakiraan akan dilakukan secara *stream* sehingga hasil yang didapatkan bersifat kontinu agar kadar CO_2 di dalam ruangan dapat diukur setiap waktu dan dijaga sesuai level yang ditentukan. Untuk mendapatkan nilai terbaik dari prakiraan yang dilakukan, digunakan parameter *windows size* karena *windows size* dapat berpengaruh terhadap akurasi hasil suatu prakiraan (Wahyuni, 2021).

1.2 Rumusan Masalah

1. Bagaimana melakukan prakiraan CO_2 secara *streaming* menggunakan library *Spark MLlib* dengan parameter *windows size*?

1.3 Ruang Lingkup Penelitian

1. Data yang digunakan dalam penelitian bersumber dari lingkungan yang dibangun oleh peneliti.
2. Tidak membahas cara mengurangi kadar CO_2 jika prakiraan yang didapat melebihi level yang ditentukan.
3. Tidak membahas bagaimana memvisualisasi hasil prakiraan CO_2 yang didapat.

1.4 Tujuan Penelitian

1. Untuk meramalkan kadar CO_2 secara *streaming* menggunakan *library Spark MLlib* dengan parameter *windows size*

1.5 Manfaat Penelitian

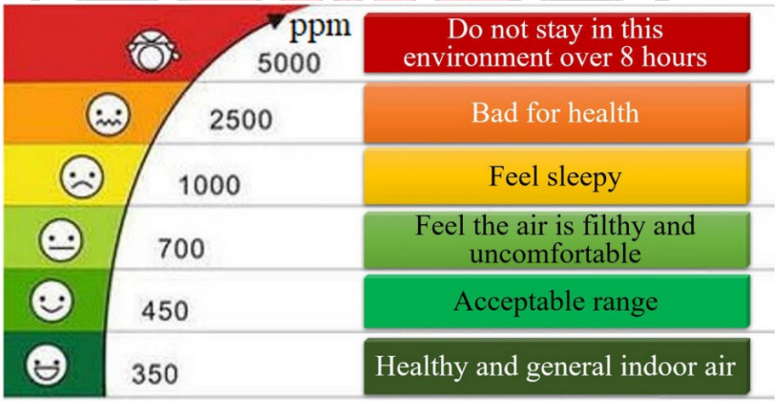
1. Dapat digunakan sebagai acuan untuk menjaga kadar CO_2 tetap pada level yang ditentukan berdasarkan hasil prakiraan yang diperoleh secara kontinu setiap waktu.
2. Dapat dijadikan referensi bagi para peneliti dalam mengembangkan *forecasting* secara *stream* di *Spark MLlib*.

POLITEKNIK NEGERI
UJUNG PANDANG

BAB II. TINJAUAN PUSTAKA

2.1 Karbondioksida (CO_2)

Gas Karbondioksida (CO_2) adalah gas yang tidak berwarna, tidak berbau, agak asam, dan tidak menyala pada suhu ruangan. Jika sering terhirup maka akan terjadi gejala-gejala seperti perubahan tekanan darah, telinga mendenging, mual, kesulitan bernafas, detak jantung tidak teratur, sakit kepala, mengantuk, pusing, tremor, lemah, gangguan penglihatan, konvulsi, hilang kesadaran, dan bahkan koma (Junaedy et al., 2022). Beberapa faktor yang dapat menyebabkan terjadinya peningkatan kadar CO_2 yaitu pernafasan manusia, asap kendaraan bermotor, kebakaran hutan, pembakaran bahan bakar seperti minyak tanah, bensin, dan solar. Paparan karbondioksida dapat menyebabkan berbagai efek kesehatan, seperti kesulitan bernafas, sakit kepala, berkeringat dan lain-lain. Tingkat karbon dioksida dan potensi masalah kesehatan ditunjukkan pada Gambar 2.1 dan Tabel 2.1 (Sung & Hsiao, 2021):



Gambar 2. 1 Efek Konsentrasi *CO_2* pada Tubuh Manusia

Sumber: (Sung & Hsiao, 2021)

1. 350-450 ppm: tingkat CO_2 di dalam ruangan yang baik
2. 450-700 ppm: tingkat CO_2 yang membuat orang mulai merasa udara kotor dan tidak nyaman
3. 1000-2500 ppm: tingkat udara buruk ringan yang dapat menimbulkan efek mengantuk.
4. 2500-5000 ppm: tingkat udara buruk yang buruk bagi kesehatan
5. > 5000 ppm: tingkat udara yang sangat buruk, menunjukkan kondisi udara yang tidak biasa, kekurangan oksigen parah yang menyebabkan kerusakan otak permanen, koma, bahkan kematian.

Tabel 2. 1 Efek konsentrasi CO_2 pada Tubuh Manusia

Kadar CO_2	Lama Paparan dan Gejala Fisik
0,01% (100 ppm)	Menimbulkan gejala seperti sakit kepala, lesu, mual, dan kelemahan otot dalam 6-8 jam.
0,02% (200 ppm)	Menyebabkan sakit kepala ringan dalam 2-3 jam
0,04% (400 ppm)	Menyebabkan sakit kepala parah dalam 2,5-3,5 jam
0,08% (800 ppm)	Menyebabkan pusing, mual, dan kram dalam 45 menit
0,16% (1600 ppm)	Menyebabkan sakit kepala dan pusing dalam 20 menit dan menyebabkan kematian dalam dua jam
0,32% (3200 ppm)	Menyebabkan sakit kepala, pusing, dan muntah dalam waktu 5-10 menit, dan menyebabkan kematian dalam waktu 30 menit
0,64% (6400 ppm)	Menyebabkan sakit kepla dan pusing dalam 1-2 menit, dan menyebabkan kematian dalam 10-15 menit

1,28% (12.800 ppm)	Menyebabkan kematian dalam 1-3 menit

Sumber: (Sung & Hsiao, 2021)

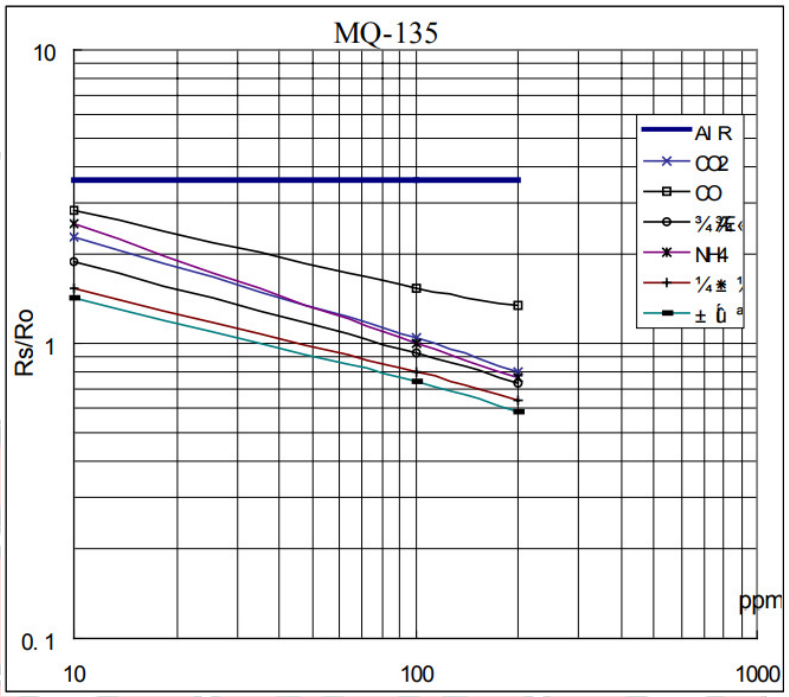
Sung & Hsiao pada tahun 2021 menyadari pentingnya mengetahui kualitas udara dalam ruangan terutama Karbondioksida (CO_2) sehingga melakukan penelitian untuk memonitoring kualitas udara termasuk CO_2 dalam ruangan. Penelitian yang dilakukan bertujuan untuk meningkatkan kualitas hidup di semua tempat tinggal dan menjaga kualitas udara dalam ruangan yang baik dengan segera merespon peningkatan konsenstrasi polusi dan kualitas udara yang buruk. Dalam penelitiannya dinyatakan bahwa nilai konsentrasi CO_2 terbaik harus antara 0 ppm < CO_2 (ppm) < 450 (ppm).

2.2 Sensor MQ-135

MQ-135 Air Quality Sensor adalah sensor yang memonitor kualitas udara untuk mendeteksi gas amonia (NH_3), natrium-(di)oksida (NOx), alkohol / ethanol (C_2H_5OH), benzena (C_6H_6), karbondioksida (CO_2), gas belerang /sulfurhidroksida (H_2S) dan asap / gas-gas lainnya di udara (Hardika & Nurfiana, 2019). Sensor ini mempunyai nilai resistansi Rs yang akan berubah bila terkena gas dan juga mempunyai sebuah pemanas (*heater*) yang digunakan untuk membersihkan ruangan sensor dari kontaminasi udara luar.

Untuk memperoleh ppm gas pada sensor MQ135, digunaka grafik sensitivitas gas yang tersedia pada datasheet sensor seperti yang terlihat pada Gambar 2.2.

Grafik tersebut menggambarkan karakteristik sensitivitas sensor gas MQ135 untuk beberapa gas, termasuk CO_2.



Gambar 2. 2 Grafik Sensitivitas Gas pada Sensor MQ135

Sumber: (Olimex, 2013)

Penggunaan sensor *MQ-135* untuk mendeteksi kadar CO_2 telah dilakukan dalam beberapa penelitian. Penelitian dari Sequeira et al. pada tahun 2015 membuat rancangan sistem kontrol otomatis untuk deteksi polusi udara dalam industri berbasis mikrokontroler PIC16F877A. Perancangan ini menggunakan komponen *MQ-135* sebagai kelengkapan dalam sistem kontrol kualitas udara dan cocok untuk mendeteksi gas polutan. Alasan kenapa *MQ-135* digunakan yaitu karena sensor ini mempunyai cakupan deteksi yang luas, respon cepat, sensitifitas tinggi, stabil dan tahan lama. Selanjutnya Abbas et al. pada tahun 2020 melakukan penelitian mengenai kemampuan sensor untuk memantau kualitas udara. Penelitian

ini berhasil mendeteksi kualitas udara menggunakan sensor *MQ-135*. Junaedy et al. pada tahun 2022 juga melakukan penelitian untuk membangun alat pengontrol kadar udara bersih dan gas berbahaya CO, CO_2 dalam ruangan berbasis mikrokontroller menggunakan sensor *MQ-135* untuk mendeteksi kadar CO_2. Penelitian ini berhasil mendeteksi CO_2 dimana jika kadar gas dalam ruangan tersebut >350 ppm maka *led* berwarna merah dan *blower* akan jalan secara otomatis.

2.3 Modul NodeMCU ESP8266

NodeMCU adalah sebuah board elektronik yang berbasis *chip ESP8266* dengan kemampuan menjalankan fungsi mikrokontroler dan juga koneksi internet (*WiFi*). Terdapat beberapa pin *I/O* sehingga dapat dikembangkan menjadi sebuah aplikasi *monitoring* maupun *controlling* pada proyek *IoT*. *NodeMCU ESP8266* dapat diprogram dengan c*ompiler* Arduino, menggunakan Arduino IDE. Berdasarkan protokol *HTTP, ESP8266 NodeMCU* terhubung ke *Wi-Fi* dan berkomunikasi dengan *platform IoT*, *Thingspeak* dan *Bylnk* (Wan et al., 2019).

ESP8266 berfungsi sebagai media komunikasi sensor. Sensor mengukur kadar CO_2 dan mengirimkan data ke Modul *Wi-Fi*. Setelah Modul *Wi-Fi* menerima aliran data, *ESP8266 NodeMCU* mengirimkan data ke platform *IoT*. Karena sifatnya yang berbiaya rendah, *ESP8266 NodeMCU* banyak dipilih untuk berinteraksi dengan sensor untuk mengunggah data lingkungan, misalnya untuk mengunggah data CO_2 ke *platform IoT* melalui *Wi-Fi* (Wan et al., 2019).

NodeMCU ESP8266 memiliki *port USB* (*miniUSB*) sehingga akan memudahkan dalam pemrogramannya. Secara fungsi modul ini hampir menyerupai *platform* modul arduino, tetapi yang membedakan yaitu dikhususkan untuk "*Connected to Internet*" (Damanik, 2019).

Wan et al. dalam penelitiannya tentang pemantauan lingkungan dan remot kontrol rumah kaca pada tahun 2019 menggunakan *ESP8266* sebagai media komunikasi sensor. Data lingkungan berupa suhu udara, kelembaban udara, dan CO_2 dikumpulkan dari sensor untuk diproses dan dikirim ke *IoT* melalui *Wi-Fi*. Penelitian ini berhasil mengumpulkan data dari sensor dan mengirimkannya ke *platform cloud* dan *IoT* melalui *Wi-Fi.*

2.4 MQTT

MQTT adalah protokol ringan yang mendukung *Internet of Things (IoT)* dan merupakan protokol komunikasi *publish/subscribe topic-based* yang didesain untuk komunikasi dengan menggunakan *bandwidth* yang rendah maupun *latecy* tinggi. *MQTT* didesain untuk meminimalkan penggunaan *bandwidth* jaringan dan kebutuhan sumber daya pada perangkat, pada waktu yang sama juga berusaha untuk memastikan keandalan dan kepastian dari pengiriman data. Prinsip yang ada ini juga memunculkan beberapa ide protokol mengenai "*machine to machine*" (M2M) atau *IoT* yang menginginkan perangkat di dunia untuk saling terhubung (MQTT.org, 2022).

2.5 Rabbit MQ

RabbitMQ adalah perangkat lunak antrian pesan yang juga dikenal sebagai *broker* pesan atau manajer antrian. Sederhananya, *RabbitMQ* adalah perangkat lunak di mana antrian ditentukan, yang terhubung dengan aplikasi untuk mentransfer pesan. *RabbitMQ* bersifat *opensource* yang dapat dijalankan di hampir semua sistem operasi dan mendukung beberapa protokol API seperti AMQP, STOMP, MQTT dan HTTP. *RabbitMQ* juga mendukung banyak bahasa pemrograman umum seperti C++, Python, dan .NET dan dapat berjalan di berbagai lingkungan *cloud* dan sistem operasi (Johansson, 2022).

2.6 Data Stream

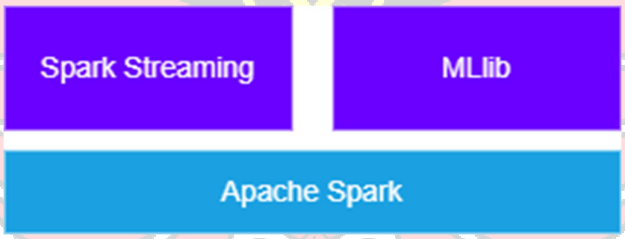
Data stream adalah data yang dihasilkan secara kontinu oleh berbagai sumber data seperti *log file*, media sosial, dan sensor. *Data stream* diproses secara berurutan dan bertahap berdasarkan data yang tiba (Amazon Web services, 2017). Teknologi pemrosesan data secara *stream* memungkinkan aliran data dapat diproses, disimpan, dianalisis, dan ditindaklanjuti secara *realtime* sehingga memungkinkan untuk segera menanggapi suatu situasi.

Data *stream* memiliki beberapa karakteristik umum, yaitu sebagai berikut (Tibco, 2021):

1. *Time Sensitive*. Setiap elemen dalam *data stream* memiliki waktu tertentu. *Data stream* sensitif terhadap waktu, dan akan kehilangan signifikansi setelah waktu tertentu.

2. *Continuous*. *Data stream* tidak memiliki awal dan akhir. *Data stream* terjadi secara *realtime*.
3. *Heterogeneous*. *Data stream* berasal dari berbagai sumber yang berbeda, sehingga terdiri dari berbagai format yang berbeda.
4. *Imperfect*. *Data stream* berasal dari berbagai sumber dan mekanisme transmisi data yang berbeda, sehingga memungkinkan terjadinya kehilangan atau kerusakan data.
5. *Volatile and Unrepeatable*. *Data stream* terjadi secara *realtime* sehingga sulit untuk transmisi *stream* berulang. Meskipun transmisi ulang dapat dilakukan, namun data yang didapat tidak akan sama dengan data sebelumnya.

2.7 Apache Spark



Gambar 2. 3 Komponen Utama Apache Spark

Sumber: (Alotaibi et al., 2020)

Apache spark adalah perangkat lunak super cepat untuk memproses data dalam skala besar (Spark, 2020). *Spark* dirancang untuk pemrosesan *big data* dengan cepat. Hal ini karena *Spark* mampu melakukan eksekusi proses secara *in memory*. Hal ini juga yang meyebabkan A*pache Spark* lebih cepat dari *Hadoop* untuk pemrosesan *big data* (Omran et al., 2021). *Apache Spark* mampu membagi beban kerja secara *parallel* dalam beberapa mesin yang terhubung dalam *cluster*.

Misalnya kemampuan mesin berada pada kapasitas 64GB RAM. Dengan *Apache Spark*, sistem dapat mengambil data dari *disk* dan menyelesaikan sebagian besar tugas pemrosesan data dalam memori. Karena semua hasil disimpan dalam memori, ini membuat pemrosesan data jauh lebih cepat daripada *Hadoop MapReduce*. Apabila terdapat 100 mesin dengan masing-masing memiliki RAM 64GB dan 16 *Core*, maka *cluster* yang dibangun memiliki kekuatan pemrosesan gabungan RAM 6400GB dan 1600 core.

Pemrosesan data pada beberapa mesin di *Apache Spark* dapat dimisalkan dalam pencarian bilangan prima antara 1 dan 100. Jika komputer yang dimiliki hanya satu dengan *core* satu, maka proses pemerikasaan bilangan prima akan dilakukan seluruhnya pada satu komputer tersebut. Jika terdapat dua komputer, maka beban kerja dapat dibagi antara dua komputer tersebut. Komputer 1 akan memeriksa bilangan prima dari 1-500 dan komputer 2 akan memeriksa bilangan prima antara 501-1000. Apabila pemrosesan tugas ditingkatkan menggunakan lima komputer, maka pekerjaan akan diselesaikan lebih cepat daripada menggunakan dua komputer.

Disisi lain, *Apache Spark* juga menawarkan *shell* interaktif yang ditulis dalam bahasa pemrograman yang berbeda seperti Python, Scala, atau R serta memiliki dua *library* utama yang dapat dimanfaatkan untuk melakukan analisis atau pemrorsesan suatu data yaitu S*park Streaming* dan *Spark MLlib* (Alotaibi et al., 2020).

2.7.1 Spark Streaming

Spark streaming merupakan salah satu komponen pada *Apache Spark* yang dapat menangai data *stream* dari berbagai sumber secara *realtime*. *Spark streaming* menerima aliran data langsung dari berbagai sumber seperti Kafka, Flume, Kinesis, TCP Socket, atau sosial media dan membagi data menjadi *batch* dan kemudian memprosesnya menggunakan mesin S*park* (Ebada et al., 2022). *Spark Streaming* menyediakan abstraksi tingkat tinggi yang disebut *DStreams* (aliran diskrit) yang merupakan kumpulan urutan RDD untuk mengalirkan aliran data berkelanjutan dari berbagai sumber. Selain itu, *Spark Streaming* memiliki skalabilitas yang baik, *throughput* yang tinggi, serta mekanisme toleransi kesalahan (Arafat et al., n.d.).

2.7.2 MLlib

MLlib adalah *library* untuk melakukan pengolahan *machine learning* di *Apache Spark* yang mendukung berbagai algoritma pembelajaran popular seperti *regression*, *classification*, *clustering*, dan *collaborative filtering*, termasuk implementasi *Random Forest* untuk prakiraan *time series* (Ebada et al., 2022). *MLlib* memiliki kecepatan seratus kali lebih cepat dari *MapReduce* karena S*park* mendukung komputasi berulang sehingga memungkinkan *MLlib* berjalan lebih cepat (Apache Spark, 2022). Selain mendukung berbagai algoritma pembelajaran popular, *MLlib* juga menyediakan berbagai alat pendukung untuk *machine learning* sebagai berikut:

1. *Featurization*, yaitu fitur-fitur yang disediakan oleh *MLlib* seperti ekstraksi fitur, transformasi, pengurangan dimensi, serta seleksi.

2. *Pipelines*, yaitu alat untuk membuat, mengevaluasi, dan menyetel *Pipeline Machine Learning.*
3. *Persistence*, yaitu kemampuan untuk menyimpan dan memuat algorima, model, dan *pipeline.*

Penggunaan *Spark Streaming* dan *Spark MLlib* untuk pengolahan data telah dilakukan dalam beberapa penelitian sebelumnya. Misalnya pada penelitian yang dilakukan oleh Nair et al. pada tahun 2018 yang melakukan prediksi status kesehatan secara *realtime* menggunakan *Spark Streaming* dan *Spark MLlib* menggunakan data Twitter. Penelitian ini melakukan prediksi status kesehatan dengan menerapkan algoritma *Decision Tree* pada data, kemudian mengirimkan pesan langsung kepada pengguna mengenai status kesehatannya.

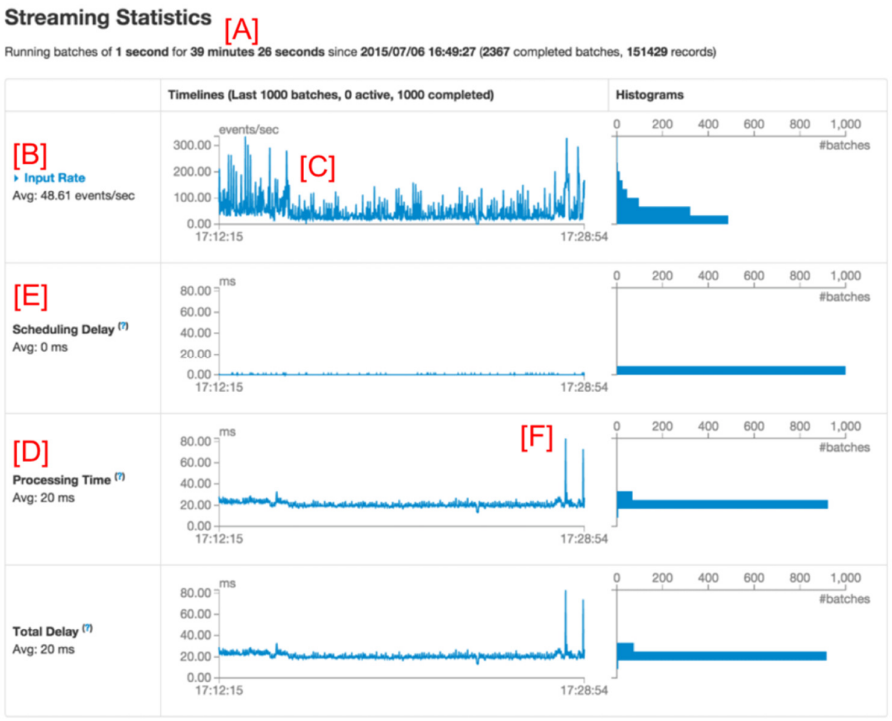
Penelitian lainnya dilakukan oleh Ed-Daoudy & Maalmi pada tahun 2019 untuk memprediksi penyakit jantung secara *realtime* menggunakan *Apache Spark* dengan menerapkan *machine learning* pada *streaming data*. Sistem dikembangkan dengan dua tahapan utama, yang pertama adalah *streaming processing* menggunakan *Spark Streaming* dan *Spark MLlib* dengan menerapkan algoritma klasifikasi R*andom Forest* pada data untuk prediksi penyakit jantung. Tahap kedua adalah penyimpanan dan visualisasi data yang menggunakan *Apache Cassandra* untuk menyimpan sejumlah besar data yang dihasilkan.

Selanjutnya pada tahun 2020, Hassan et al. membuat sistem prakiraan status Kesehatan secara *realtime* menggunakan *Kafka*, *Spark Streaming*, dan *Spark MLlib* dengan menggunakan algoritma *Streaming Linear Regression With SGD* untuk melatih data dan meramalkan status kesehatan pasien. Prakiraan dilakukan

dengan menggunakan dataset catatan medis untuk penyakit diabetes, jantung, dan kanker payudara dengan ukuran yang berbeda untuk tiap data. Setiap kumpulan data dibaca dari file CSV kemudian dikirimkan ke *Spark Streaming*. *Spark* kemudian membagi masing data menurut ukuran *window* kemudian diproses pada *Spark MLlib* menggunakan algoritma *Linear Regression with SGD* untuk mendapatkan hasil prakiraan. Hasil penelitian yang diapat adalah ramalan status kesehatan berhasil didapatkan berdasarkan dataset kesehatan yang digunakan. Akurasi terbaik didapat pada dataset yang memiliki jumlah data terbanyak serta ukuran *windows* yang lebih besar.

2.8 Streaming Statistic UI Spark

Apache Spark menyediakan visualisasi pada *Tab Streaming UI Spark* untuk melihat aktifitas *streaming* yang terjadi ketika *Spark Streaming* dijalankan (Microsoft, 2023). Aktifitas yang dilakukan di *Spark Streaming* dapat dilihat dalam grafik yang disajikan seperti yang terlihat pada Gambar 2.4 dan Gambar 2.5.



Gambar 2. 4 Tab Streaming UI Spark

Sumber : (Das et al., 2015)

Gambar 2. 5 Detail Input Rate

Sumber : (Das et al., 2015)

Baris pertama (bagian A) menunjukkan status aplikasi *streaming* saat ini, misalnya berdasarkan gambar berarti bahwa aplikasi telah berjalan selama hampir 40 menit dengan *interval batch* satu detik. Berikutnya pada bagian *input rate* (bagian B) menunjukkan bahwa aplikasi *streaming* telah menerima data dengan laju sekitar 49 kejadian/detik di semua sumbernya. Misalnya pada gambar menunjukkan bahwa garis waktu menunjukkan sedikit penurunan pada tingkat rata-rata di tengah (bagian C). Detail lebih lanjut dari informasi *input rate* dapat dilihat pada gambar 2.5. Gambar 2.5 menunjukkan garis waktu dari masing-masing sumber data, dimana dari gambar tersebut terlihat bahwa aplikasi memiliki dua sumber data yaitu *SocketReceiver-0* dan *SocketReceiver-1* yang salah satunya menyebabkan tingkat penerimaan keseluruhan data turun karena berhenti menerima data untuk durasi yang singkat (Das et al., 2015).

Pada bagian *processing time* (bagian D), garis waktu menunjukkan bahwa kumpulan data telah diproses rata-rata dalam waktu 20 ms. Hal tersebut menunjukkan bahwa waktu pemrosesan lebih pendek dibandingkan dengan *interval batch* sebesar satu detik (bagian A) (Microsoft, 2023). Nilai yang lebih pendek tersebut menunjukkan bahwa *Scheduling Delay* (bagian E) atau waktu sebuah *batch* menunggu *batch* sebelumnya selesai sebagian besar bernilai nol karena *batch* diproses secepat *batch* tersebut dibuat. *Scheduling Delay* ini menjadi indikator utama apakah aplikasi *Streaming* stabil atau tidak (Das et al., 2015).

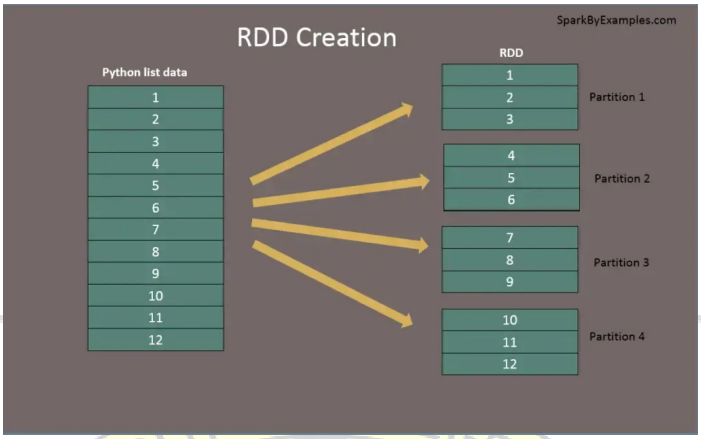
2.9 Micro-Batch

Micro-batch atau yang biasa disebut *micro-batch processing* adalah praktik pengumpulan data dalam kelompok kecil (*batch*) untuk diproses. Data akan diproses dalam waktu yang sangat singkat. *Micro-batch* digunakan ketika kita akan memproses data yang segar (Levy, 2021).

Pemrosesan *micro-batch* sangat mirip dengan pemrosesan *batch* tradisional, namun yang membedakannya adalah *micro-batch* diproses lebih cepat atau dalam waktu yang lebih kecil. *Micro-batch* dapat memproses data bedasarkan beberapa frekuensi. *Micro-batch processing* digunakan pada *Apache Spark Streaming* untuk memproses *data stream* (Hazelcast, 2021).

2.10 RDD

RDD secara sederhana dapat diartikan sebagai kumpulan objek terdistribusi yang bersifat kekal (*immutable*). RDD tidak dapat dubah, namun dapat diakali dengan membuat RDD baru dengan melakukan transformasi pada RDD yang sudah ada (Ryanto, 2017). Setiap data di RDD dapat dibagi kedalam beberapa partisi yang dapat dihitung pada node cluster yang berbeda, berbeda dengan *Python* yang hanya dapat diproses dalam satu proses (SparkByExamples, 2022).



Gambar 2. 6 Analogi Kerja RDD

Sumber: (SparkByExamples, 2022)

2.11 Prakiraan Time Series

Time series data adalah data yang berurutan (*Sequence of data points*) yang dikumpulkan selama interval waktu serta digunakan untuk melacak perubahan dari waktu ke waktu, sehingga dapat dikatakan bahwa prakiraan *Time series* atau *time series forecasting* adalah sebuah area pada *machine learning* yang berfokus pada atribut waktu, atau analisis rentetan data yang sekuensial terhadap waktu, lalu meramalkan data-data yang akan datang berdasarkan data sebelumnya. Pada prakiraan *time series*, dimensi waktu memiliki peran yang sangat penting. Dimensi waktu berfungsi sebagai *feature* yang dapat memberikan banyak kegunaan dalam pemrosesan dan analisis data untuk mendapat *insight* atau kesimpulan terhadap suatu observasi (Firdaus, 2019). Ada tiga aspek penting dalam *time series prediction*, yaitu (influxdata, 2022):

1. Sample data

   Sample data merupakan data yang dikumpulkan selama beberapa waktu yang diperlukan untuk membuat model sehingga dapat digunakan untuk meramalkan nilai di masa depan. Sampel data harus terdiri dari pasangan *input* dan *output* (x, y).

2. Learn a Model

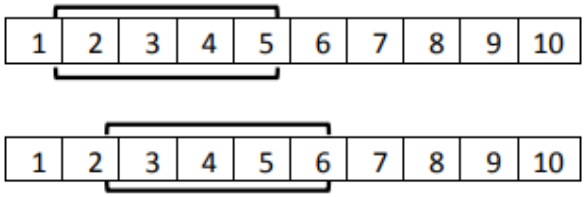
   Model merupakan algoritma yang digunakan dalam *time series prediction*, dalam hal ini yaitu model regresi. *Sample* data akan dipelajari hingga didapatkan sebuah model yang akan digunakan untuk mendapatkan hasil prakiraan.

3. Making Prediction

   Setelah model regresi mempelajari sample data, model tersebut kemudian digunakan untuk meramalkan nilai di masa depan.

2.12 Window Operations

Window operations merupakan komputasi *windows* yang disediakan pada *Apache Spark*. *Windows operation* memungkinkan untuk menerapkan transformasi pada *sliding window data* (Apache Spark Documentation, 2018). *Sliding Window* atau *windowing* adalah pembentukan struktur dari data *time series* yang tersedia. Ukuran dari *window* dan segmen dilakukan perubahan untuk memperoleh *error* yang terkecil (Wahyuni, 2021).



Gambar 2. 7 Ilustrasi *Sliding Windows*

Sumber: (Wahyuni, 2021)

Dari Gambar 2.7 dapat diilustrasikan contoh dari *sliding window size* 5. Angka 1 sampai 10 adalah unit observasi dari data *time series*. Misalkan angka 1 menggambarkan data pada menit ke-1, angka 2 menggambarkan data pada menit ke-2 dan seterusnya. *Window* pertama dimulai dari data ke-1 sampai dengan ke-5 yang digunakan untuk memprediksi data ke-6. Segmen kedua dimulai dari data ke-2 sampai dengan data ke-6 yang digunakan untuk memprediksi data ke-7. Proses ini terus dilanjutkan sampai dengan seluruh data hasil observasi habis tersegmentasi.

Beberapa istilah yang dipakai dalam *window operations* adalah sebagai berikut (Apache Spark Documentation, 2018)

1. *Batch interval*, merupakan berapa lama data akan dikumpulkan sebelum diteruskan ke pemrosesan berikutnya. Misalnya diatur *interval batch* 1 detik, *Spark* akan mengumpulkan data selama 1 detik kemudian melakukan perhitungan dengan data itu.
2. *Windows size*, merupakan berapa banyak data historis yang dikumpulkan dalam RDD sebelum diproses. Misalnya diatur *windows size* 5 detik, maka data historis akan dikumpul sebanyak 5 detik untuk diproses. Lama pemrosesan data

menggunakan *batch interval* sebagai acuan. Misalnya diatur *interval batch* 1 detik dan *windows size* 5 detik, maka perhitungan akan dilakukan setiap detik untuk 5 data historis yang dikumpulkan.

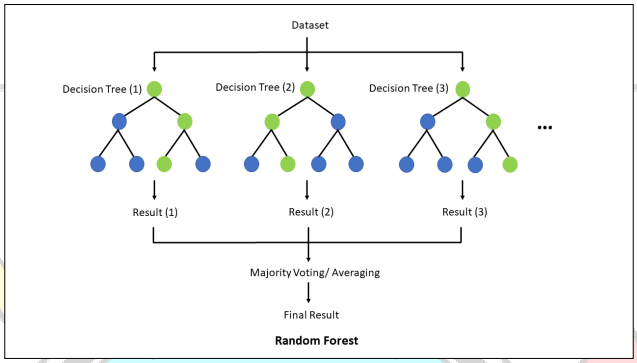
3. *Sliding interval*, merupakan jumlah waktu dalam detik untuk seberapa banyak jendela yang akan bergeser.

Penggunaan *windows size* sebagai teknik untuk melakukan prediksi *time series* telah diteliti sebelumnya oleh Wahyuni pada tahun 2021. Penelitian ini melakukan prediksi inflasi menggunakan metode *neural network* dengan menggunakan teknik *sliding window* yang disebut juga metode *windowing*. Penelitian dilakukan dengan tiga percobaan *window size* yaitu 6, 12, dan 18 untuk melihat adakah perbedaan akurasi hasil dari beberapa *window size* tersebut. Hasil percobaan menyimpulkan bahwa ukuran dari *windowing* akan mempengaruhi akurasi dari hasil prediksi dimana dari penelitian yang dilakukan didapat bahwa *window size* 6 memiliki akurasi paling baik untuk memprediksi inflasi dengan RMSE 0,435.

2.13 Random Forest

Random forest (RF) merupakan salah satu metode yang paling banyak digunakan dalam *data mining*, termasuk untuk *time series forecasting*. *Random Forest* dapat digunakan untuk klasifikasi maupun regresi. *Random Forest* termasuk ke dalam *decision tree*. *Decision Tree* atau pohon pengambil keputusan adalah sebuah diagram alir yang berbentuk seperti pohon yang memiliki sebuah *root node* yang digunakan untuk mengumpulkan data. Sebuah *inner node* yang berada pada *root node* yang berisi tentang pertanyaan tentang data dan sebuah *leaf node* yang

digunakan untuk memecahkan masalah serta membuat keputusan. *Decision tree* mengklasifikasikan suatu sampel data yang belum diketahui kelasnya kedalam kelas – kelas yang ada (UGM, 2018).



Gambar 2. 8 Ilustrasi Random Forest

Sumber: (UGM, 2018)

Random forest adalah kombinasi dari masing – masing *tree* yang baik kemudian dikombinasikan ke dalam satu model. *Random Forest* bergantung pada sebuah nilai *vector random* dengan distribusi yang sama pada semua pohon *decision tree*. Ilustrasi *Random Forest* dapat dilihat pada Gambar 2.8. Cara kerja R*andom Forest* berawal dari memecah data sampel yang ada ke dalam *decision tree* secara acak kemudian disatukan hingga membentuk sekumpulan *tree (forest)*. Selanjutnya, akan dilakukan *voting* pada setiap kelas kemudian diambil *vote* yang paling banyak.

Altinçöp & Oktay pada tahun 2019 melakukan penelitian untuk meramal tingkat polusi udara menggunakan algoritma Jaringan Syaraf Tiruan dan *Random Forest*. Penelitian ini menggunakan dua indikator penting pencemaran udara, yaitu partikulat 10 (PM10) dan karbon monoksida (CO) yang diramalkan dengan

analisis deret waktu *Random Forest* dan metode jaringan syaraf tiruan menggunakan data meteorologi seperti suhu udara, kelembaban, kecepatan angin dan udara. Prakiraan polusi udara dibuat untuk hari berikutnya dengan data yang dikumpulkan dari Kementerian Lingkungan Hidup Republik Turki dan Layanan Meteorologi Negara Turki setelah pra-pemrosesan. Ketika hasil prakiraan diperiksa, terlihat jelas bahwa metode R*andom Forest* menghasilkan hasil yang sangat akurat dan berkinerja lebih baik daripada jaringan syaraf tiruan.

Pada tahun 2022, Lei et al. juga melakukan penelitian untuk memprediksi kualitas udara di Macau, China. Penelitian ini menggunakan data dari Biro Meteorologi dan Geofisika Macau dari tahun 2013 hingga tahun 2022 dengan mengambil dua indikator polusi udara yaitu partikulat 10 (PM10) dan partikulat 2.5 (PM2.5). Pembuatan model menggunakan data dari tahun 2013 hingga 2018, sendangkan data tahun 2019 hingga tahun 2022 digunakan untuk validasi. Penelitian ini menggunakan beberapa metode pembelajaran mesin yaitu *Random Forest (RF)*, *Gradient Boosting (GB*), Support Vector Regression (SVR), serta *Multiple linear Regression (MLR)* dan berhasil menunjukkan bahwa kinerja *Random Forest* secara sifnifikan lebih baik daripada metode lainnya.

Akurasi yang baik dari penggunaan algoritma *Random Forest* untuk melakukan prakiraan juga dibuktikan pada penelitian yang dilakukan oleh Juarez & Petersen pada tahun 2022. Penelitian ini melakukan prakiraan konsentrasi ozon di kota Delhi, India. Data yang digunakan adalah data tingkat polutan per jam di Delhi, India pada januari 2015 hingga juni 2020 dari Dewan Kontrol Polusi Pusat India. Data yang diperoleh mencakup variabel polutan dan variabel cuaca yaitu O3,

O3P24, PM10, NO2, NOX, NH3, CO, SO2, Toluene, Temperatur, dan Kelembaban. Beberapa algoritma *machine learning* digunakan dalam penelitian ini, yaitu *XGBoost, Random Forest, K-Nearest Neighbor Regression, Support Vector Regression, Decision Trees, AdaBoost, dan Lienar Regression*. Penelitian ini menunjukkan bahwa *machine learning* sangat menjanjikan untuk melakukan prakiraan ozon, dimana *XGBoost* dan *Random Forest* merupakan algoritma yang mendemonstrasikan keterampilan prakiraan terbaik secara keseluruhan. (Hassan et al., 2020)

2.14 Pengukuran Error

Pengukuran *error* sangat penting untuk dilakukan, terutama dalam melakukan *forecasting* untuk mengetahui tingkat kesalahan kinerja metode yang digunakan. Hasil prakiraan terbaik adalah metode terpilih dengan memiliki tingkat kesalahan yang paling minim. Pengukuran tingkat *error* suatu model dapat dilakukan dengan beberapa metrik seperti RMSE (*Root Mean Square Error*) dan MAPE (*Mean Absolute Percentage Error*) (Wiranda & Sadikin, 2019).

2.14.1 Root Mean Square Error (RMSE)

Root Mean Square Error (RMSE) merupakan besarnya tingkat kesalahan hasil prakiraan, dimana semakin kecil (mendekati 0) nilai RMSE maka hasil prakiraan akan semakin akurat (Hastomo et al., 2021). *Root Mean Squared Error* (RMSE) merupakan salah satu cara untuk mengevaluasi model dan telah sering digunakan dalam berbagai studi yang berkaitan dengan prakiraan (Wiranda & Sadikin, 2019). Nilai RMSE rendah menunjukkan bahwa variasi nilai yang

dihasilkan oleh suatu model prakiraan mendekati variasi nilai observasinya. RMSE menghitung seberapa berbedanya seperangkat nilai. Semakin kecil nilai RMSE, semakin dekat nilai yang di ramal dan diamati (Maulid, 2022).

Cara Menghitung *Root Mean Square Error* (RMSE) adalah dengan mengurangi nilai aktual dengan nilai prakiraan kemudian dikuadratkan dan dijumlahkan keseluruhan hasilnya kemudian dibagi dengan banyaknya data. Hasil perhitungan tersebut selanjutnya dihitung kembali untuk mencari nilai dari akar kuadrat. Rumus menghitung RMSE dapat dilihat pada gambar 2.9

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n}(A_t - F_t)^2}{n}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Gambar 2. 9 Rumus RMSE (*Root Mean Square Error*)

Dimana:

At = Nilai data aktual

Ft = Nilai hasil prakiraan

n = Banyaknya data

$\sum$ = Summation (Jumlahkan keseluruhan nilai)

2.14.2 Mean Absolute Percentage Error (MAPE)

Mean Absolute Percentage Error (MAPE) merupakan rata-rata dari keseluruhan persentase kesalahan (selisih) antara data aktual dengan data hasil prakiraan. MAPE lebih mudah dipahami dalam memprediksi akurasi prakiraan karena memberikan informasi seberapa besar kesalahan prakiraan dibandingkan dengan nilai sebenarnya. Semakin kecil nilai presentasi kesalahan *(percentage error)* pada MAPE maka semakin akurat hasil prakiraan tersebut. Kriteria keakuratan MAPE dapat dilihat pada tabel 2.2 (Aditya et al., 2019).

Tabel 2. 2 Tabel Kriteria Keakuratan MAPE

Range MAPE	Arti Nilai
< 10%	Kemampuan model prakiraan sangat baik
10-20%	Kemampuan model prakiraan baik
20-50%	Kemampuan model prakiraan layak
>50%	Kemampuan model prakiraan buruk

Sumber: (Aditya et al., 2019)

Cara menghitung *Mean Absolute Percentage Error* (MAPE) adalah dengan melakukan penjumlahan secara keseluruhan dengan terlebih dahulu melakukan pengurangan nilai data aktual dengan data prakiraan kemudian membaginya dengan data aktual (diharuskan nilainya absolut) dan dikalikan dengan 100 kemudian dibagi dengan banyaknya data yang ada. Yang dimaksud absolut disini adalah nilainya apabila negative tetap bernilai positif. Rumus *Mean Absolute Percentage Error* (MAPE) dapat dilihat pada gambar 2.10.

$$MAPE = \frac{\sum_{t=1}^{n} \left| \left(\frac{A_t - F_t}{A_t} \right) 100 \right|}{n} \quad \ldots\ldots\ldots\ldots\ldots\ldots (2)$$

Gambar 2. 10 Rumus MAPE (*Mean Absolute Percentage Error*)

Dimana:

At = Nilai data aktual

Ft = Nilai hasil prakiraan

n = Banyaknya data

∑ = Summation (Jumlahkan keseluruhan nilai)

BAB III. METODOLOGI PENELITIAN

3.1 Tempat dan Waktu Penelitian

Tempat penelitian dilaksanakan di Laboratorium Jaringan Komputer Kampus 1 Politeknik Negeri Ujung Pandang, Tamalanrea Indah, Tamalanrea, Tamanlanrea Indah, Kec. Tamalanrea, Kota Makassar, Sulawesi Selatan 90245 dimulai dari bulan Februari 2022 sampai dengan bulan April 2023.

3.2 Alat dan Bahan

Adapun alat dan bahan yang akan digunakan dalam proses penelitian sebagai berikut:

3.2.1 Alat

Alat-alat yang digunakan selama proses penelitian ini dibedakan menjadi dua yakni kebutuhan perangkat keras pada Tabel 3.1 dan kebutuhan perangkat lunak pada Tabel 3.2.

Tabel 3. 1 Kebutuhan Perangkat Keras

No.	Perangkat Keras	Spesifikasi	Keterangan
1.	1 Unit Server	Processor: Intel(R) Core(TM) i7-8700 RAM: 16 GB Hard drive: 1 TB	Sebagai perangkat tempat infrastruktur sistem dibangun
2.	1 Unit Laptop	Processor: Intel core i5-7200U CPU @ 2.50GHz RAM: 8 GB	Sebagai perangkat yang digunakan untuk me-remote server

		HDD: 1TB SSD: 240 GB	
3.	1 Unit Kabel LAN		Sebagai perangkat yang digunakan untuk menghubungkan switch dan server
4.	1 Unit ESP 8266		Sebagai perangkat yang menghubungkan ke jaringan
5.	1 Unit MQ 135		Sebagai perangkat untuk menangkap data CO_2
6.	3 Kabel Jumper		Sebagai perangkat yang menghubungkan sensor dengan ESP 8266
7.	1 Kabel Micro USB		Sebagai perangkat yang mengubungkan IoT ke sumber daya

Tabel 3. 2 Kebutuhan Perangkat Lunak

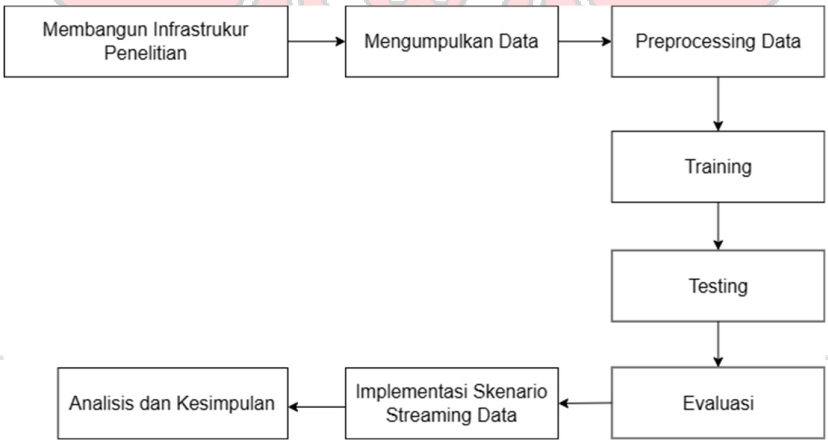
No.	Perangkat Lunak	Spesifikasi	Keterangan
1.	Hypervisor	Proxmox Virtualization Environtment versi 7.1	Sistem operasi yang digunakan
2.	Putty	Versi 0.73	Me-remote server
3.	Apache Spark	Versi 3.2.1	Framework yang digunakan
4.	Jupyter Notebook	Versi 8.4.0	Layanan untuk komputasi interaktif di berbagai bahasa pemrograman

5.	Java Development Kit	Versi 8	Mengembangkan aplikasi berbasis Java
6.	Python	Versi 3.7+	Bahasa pemrograman yang digunakan
7.	Spark MLlib		Library ML yang digunakan
8.	Arduino IDE	Versi 2.0.3	Framework code menulis dan mengupload ke data dari sensor
9.	Rabbit MQ		Message broker yang digunakan

3.2.2 Bahan

Bahan yang digunakan merupakan data kualitas udara CO_2 yang didapatkan dari sensor *MQ-135* secara *stream* setiap menit.

3.3 Prosedur Penelitian



Gambar 3. 1 Prosedur Penelitian

Agar penelitian yang dilakukan dapat berjalan dengan baik dan terstruktur diperlukan sebuah prosedur penelitian sehingga hasil yang diperoleh sesuai dengan

tujuan penelitian. Metode penelitian yang digunakan adalah metode simulasi dengan tahapan digambarkan pada Gambar 3.1.

3.3.1 Membangun Infrastruktur Penelitian

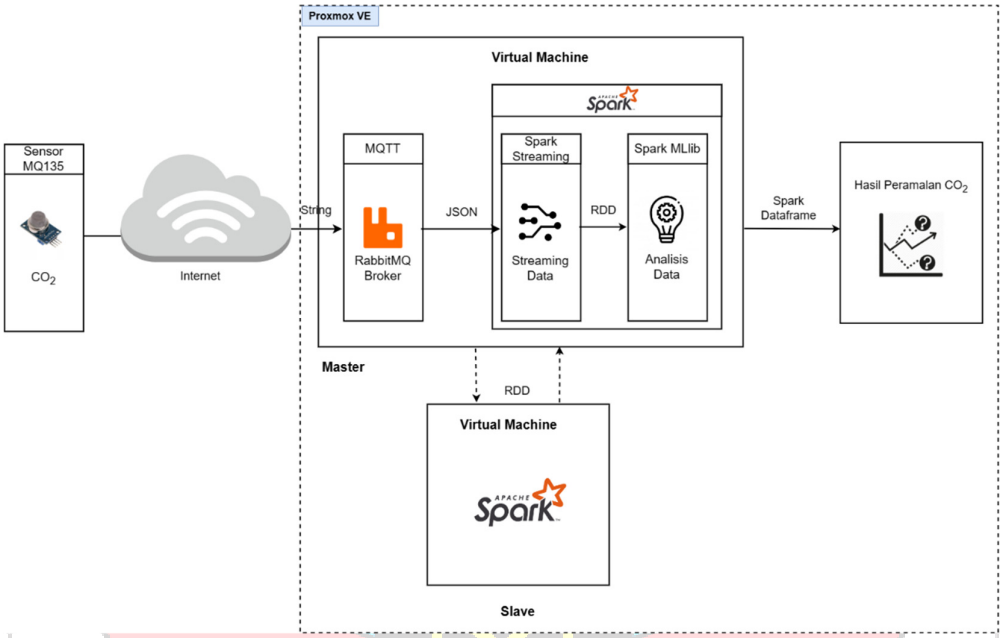
Pada tahap ini akan dibangun infrastruktur sistem yang terdiri dari *MQTT Broker*, S*park* dan sensor. Infrastruktur yang dibangun menggunakan sebuah perangkat *server* yang memiliki *OS Hypervisor Proxmox VE*. Di dalam *server* tersebut akan dibangun infrastruktur *virtualization environment multi node cluster* yang terdiri dari 2 buah VM, yakni 1 VM bertindak sebagai *master* dan 1 VM lainnya bertindak sebagai *slave (workers)*. Adapun spesifikasi pada *master* dan *workers* dapat dilihat pada Tabel 3.3. Setelah membangun infrastruktur *virtualization environment*, selanjutnya adalah melakukan instalasi Ubuntu pada setiap VM yang dibuat.

Tabel 3. 3 Spesifikasi Master dan Workers

No.	Spesifikasi	Master Node	Workers Node
1.	RAM	2	4
2.	Core	2	4
3.	Memory	32GB	32GB

Dalam tiap mesin yang telah terinstall sistem operasi Ubuntu, selanjutnya akan dilakukan instalasi *tools*. Mesin yang bertindak sebagai *master* akan diinstall *MQTT Broker* serta *Apache Spark* yang akan dihubungkan dengan sensor *MQ-135* untuk membaca data Karbondioksida (CO_2) diruangan, sedangkan mesin yang bertindak sebagai *slave* hanya akan diinstal *Apache Spark* untuk membagi beban

kerja dari mesin *master* seperti yang terlihat pada Gambar 3.2. (Langkah-langkah untuk konfigurasi sensor dapat dilihat pada lampiran 1).



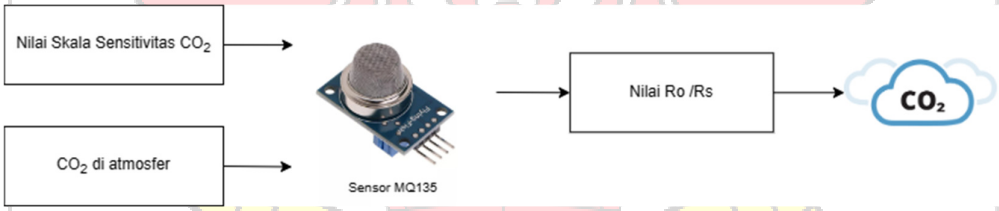
Gambar 3. 2 Infrastruktur Penelitian

Dari Gambar 3.2 dapat dijabarkan bahwa pertama, sensor akan membaca data kualitas udara berupa Karbondioksida (CO_2). Cara memilah data CO_2 pada sensor dijelaskan pada bagian 3.3.2 yang digambarkan pada Gambar 3.3. Data yang berasal dari sensor diatur untuk berformat *string* kemudian dikirimkan ke *broker RabbitMQ* menggunakan protokol *MQTT*. *RabbitMQ* berfungsi untuk mengatur data yang diterima dari sensor untuk dikirmkan ke *Spark Streaming*. Data yang dikirim ke *Spark Streaming* berformat JSON. Data tersebut kemudian diatur sehingga menghasilkan data dalam format RDD. Data yang telah melalui S*park Streaming* akan bersifat *realtime*, artinya data akan keluar satu per satu secara kontinu. Data berformat RDD dari *Spark Streaming* kemudian dikirim ke *Spark*

MLlib untuk dianalisis. Analisis yang dilakukan pada *Spark MLlib* akan memberikan hasil prakiraan kadar CO_2 secara *realtime* dalam format *spark dataframe*. Proses yang dilakukan menggunakan mesin *slave* untuk membagi beban kerja dari mesin *master*. *Slave* akan mengolah data yang berformat RDD kemudian mengembalikan hasilnya ke *master*.

3.3.2 Mengumpulkan Data

Data yang digunakan dalam penelitian ini adalah data CO_2 dalam satuan ppm yang didapatkan dari lingkungan yang dibangun oleh peneliti. Lingkungan yang dibangun diatur untuk memperoleh kadar CO_2 yang berada di kadar normal serta diatas kadar normal. Data ini diperoleh dari sensor *MQ-135* yang telah dikalibrasi sesuai dengan kadar CO_2 di atmosfer agar nilai CO_2 yang ditangkap sesuai dengan data CO_2 di udara. Tahapan memilah data CO_2 pada sensor digambarkan pada Gambar 3.3.

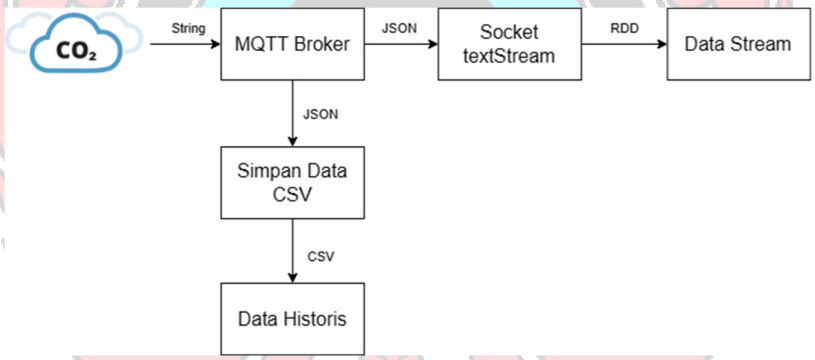


Gambar 3. 3 Tahapan Memilah Data CO_2 Pada Sensor

Berdasarkan Gambar 3.3 dapat dijabarkan bahwa untuk mendapatkan data CO_2 dibutuhkan nilai skala sensitivitas CO_2 dan nilai kadar CO_2 di atmosfer. Nilai skala sensitivitas diperoleh dari *datasheet* sensor MQ135. Nilai skala sensitifitas pada sensor MQ135 berbeda untuk tiap gas, sehingga untuk mendapatkan nilai kadar CO_2, dibutuhkan nilai sensitivitas untuk gas CO_2. Nilai kadar CO_2 di atmosfer

pakai untuk mengatur kadar CO_2 agar data yang ditangkap sesuai dengan kadar CO_2 diudara. Nilai sensitivitas dan kadar CO_2 di atmosfer selanjutnya diproses oleh sensor untuk memperoleh nilai Ro/Rs (resistansi sensor). Setelah nilai Ro/Rs diperoleh, selanjutnya nilai tersebut akan diproses lebih lanjut hingga diperoleh kadar CO_2 di lingkungan yang diuji.

Pengambilan data CO_2 dibagi menjadi dua metode, yaitu pengambilan data historis (*data batch*) yang akan digunakan untuk *training* dan *testing* dan pengambilan data secara *stream* yang akan digunakan pada untuk prakiraan secara *stream*. Tahap pengumpulan data dapat dilihat pada Gambar 3.4.



Gambar 3. 4 Tahapan Pengumpulan Data

Untuk pengambilan data historis, pertama sensor akan membaca data CO_2 setiap satu menit kemudian mengirimkannya ke *MQTT Broker*. Data tersebut dikumpulkan kemudian disimpan dalam bentuk CSV. Pengambilan data historis dilakukan selama delapan hari dan diperoleh data sebanyak 11843.

Pengambilan data dilakukan selama delapan hari karena hasil pengambilan data selama delapan hari tersebut sudah dapat mewakili kondisi lingkungan yang dibangun. Pengambilan data ini dilakukan selama 24 jam untuk mengambil data

tiap menit. Selama pengambilan data, kondisi lingkungan diatur setiap hari secara konsisten untuk memperoleh data CO_2 yang berada di kadar normal dan di atas kadar normal. Data historis yang diperoleh selanjutnya akan digunakan untuk proses *training* dan *testing* guna memperoleh nilai terbaik dari ukuran *windows size* yang akan digunakan pada tahap *streaming data*.

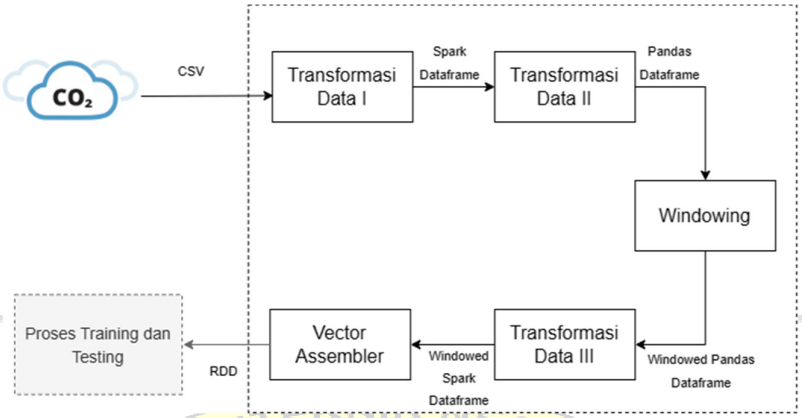
Selanjutnya untuk pengambilan data *stream*, pertama sensor akan membaca data CO_2 setiap satu menit kemudian mengirimkannnya ke *MQTT Broker*. *MQTT broker* kemudian akan mengirimkan tiap data yang diterima ke *Socket textStream* satu persatu sehingga didapatkan data dalam bentuk *streaming*.

3.3.3 Preprocessing Data

Preprocessing adalah tahapan yang berfungsi untuk mengatur data hingga siap untuk digunakan pada tahap *training*, *testing,* dan tahap *streaming data*. Alur tahapan *preprocessing* adalah sebagai berikut

3.3.3.1 Preprocessing Data Training dan Testing

Tahapan ini dilakukan untuk mengatur data hingga siap untuk digunakan untuk tahapan *training* dan *testing*. Alur tahapan *preprocessing* yang dilakukan dapat dilihat pada gambar 3.5.

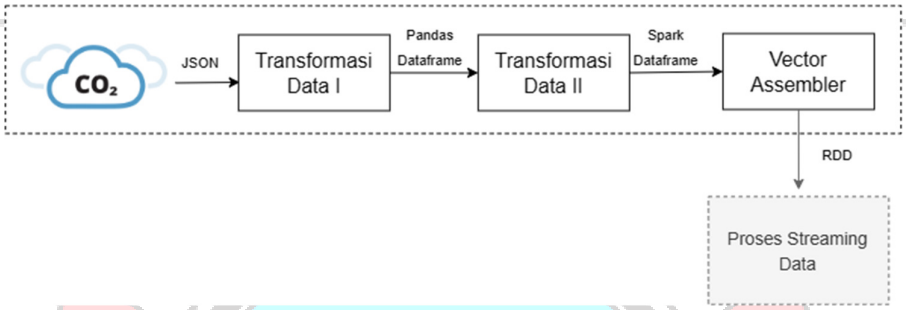


Gambar 3. 5 Tahapan *Preprocessing* Data *Training* dan *Testing*

1. Tahap pertama adalah melakukan transformasi data untuk mengubah data format CSV ke dalam *dataframe* agar dapat diolah untuk tahap selanjutnya. Proses pengubahan format CSV menjadi *dataframe* dilakukan dalam beberapa tahap agar data dapat diolah dengan baik. Pertama, data dalam format CSV diubah kedalam *spark dataframe*, lalu data dalam format *spark dataframe* tersebut diubah kedalam *pandas dataframe* untuk keperluan *windowing*. Selanjutnya, data yang masih dalam format *pandas dataframe* kemudian diubah kembali kedalam bentuk *spark dataframe* untuk dapat dilanjutkan ke tahap berikutnya.
2. Tahap kedua adalah melakukan *vector assembler* untuk mengubah *spark dataframe* menjadi *vector* agar dapat digunakan sebagai *variable input* pada tahap *training* dan *testing*. Mengubah format *spark dataframe* menjadi *vector* perlu dilakukan karena *spark MLlib* hanya dapat mengolah data dalam bentuk vector. Banyaknya data *vector* diatur menurut ukuran *windows* yang dimasukkan. Hasil dari *vector assembler* berformat RDD.

3.3.3.2 Preprocessing Data Stream

Tahapan ini dilakukan untuk mengatur data hingga siap untuk digunakan untuk tahapan *streaming data.* Alur tahapan *preprocessing* yang dilakukan dapat dilihat pada gambar 3.6.

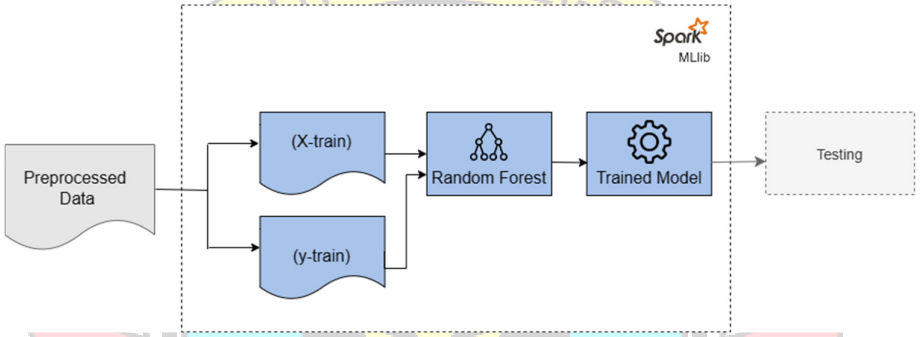


Gambar 3. 6 Tahapan *Preprocessing* Data *Stream*

1. Tahap pertama adalah melakukan transformasi data untuk mengubah data format JSON ke dalam *spark dataframe* agar dapat diolah untuk tahap selanjutnya. Pertama, data dalam format JSON diubah kedalam *pandas dataframe*, lalu data dalam format *pandas dataframe* tersebut diubah kedalam *spark dataframe* agar dapat dilanjutkan ke tahap berikutnya.
2. Sama seperti *preprocessing* untuk *training* dan *testing*, *preprocessing* untuk *stream data* juga perlu untuk mengubah *spark dataframe* menjadi *vector* melalui tahapan *vector assembler* agar dapat digunakan sebagai *variable input* pada tahap *streaming data*. Mengubah format *dataframe* menjadi *vector* perlu dilakukan karena *spark MLlib* hanya dapat mengolah data dalam bentuk *vector*. Banyaknya data *vector* diatur menurut ukuran *windows* yang dimasukkan. Hasil dari *vector assembler* berformat RDD.

3.3.4 Training

Pada tahap ini dilakukan proses *training* data untuk mendapat *training model*. Model yang didapatkan pada tahap ini akan digunakan untuk mendapatkan hasil prakiraan pada tahap testing. Data yang digunakan untuk proses *training* adalah data historis yang telah di-*preprocessing*. Adapun tahapan proses *training* dapat dilihat pada Gambar 3.7.



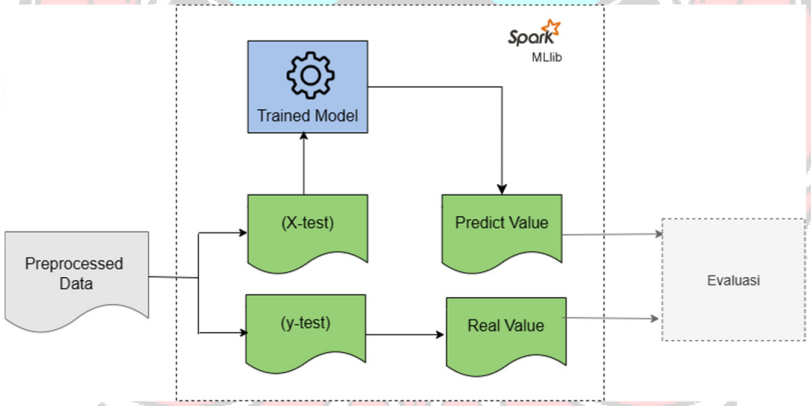
Gambar 3. 7 Proses *Training*

Tahap *training* dimulai ketika sistem membaca data pada folder *train*. Selanjutnya data akan dipisah menjadi dua yaitu *X-train* dan *y-train*. *X-train* adalah data yang akan dipelajari untuk prakiraan, dan *y-train* merupakan label atau data berikutnya yang akan menjadi hasil prakiraan. *X-train* dan *y-train* selanjutnya akan diolah menggunakan algoritma *Random Forest* hingga didapatkan model prakiraan yang kemudian akan digunakan terhadap data *testing*.

Proses *training* akan dilakukan untuk setiap ukuran *windows size* yang digunakan agar didapatkan model *training* yang sesuai untuk setiap ukuran *windows size* yang akan digunakan pada tahap *testing*

3.3.5 Testing

Tahap ini dimulai ketika model sudah berhasil didapatkan pada tahap *training*. Tahap *testing* dilakukan untuk mendapatkan hasil prakiraan berdasarkan model yang telah didapat sebelumnya. Data yang digunakan untuk proses *testing* adalah data historis berupa data batch yang telah di-*preprocessing*. Hasil dari tahap testing adalah hasil prakiraan pada tiap ukuran *windows size* yang selanjutnya akan dievaluasi untuk mengetahui nilai terbaik dari tiap ukuran *windows size* yang diuji. Nilai terbaik yang diperoleh selanjutnya akan digunakan untuk prakiraan CO_2 secara *stream*. Adapun tahapan proses *testing* dapat dilihat pada Gambar 3.8.



Gambar 3. 8 Proses Testing

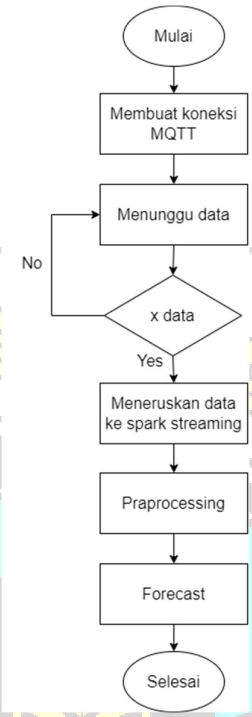
Tahap *testing* dimulai apabila model telah berhasil didapatkan. Tahap untuk *testing* sama seperti tahap *training*, yaitu sistem akan membaca data pada folder *testing*, kemudian data akan dipisah menjadi dua yaitu *X-test* dan *y-test*. *X-test* akan digunakan untuk mendapatkan hasil prakiraan berdasarkan model yang telah didapat sebelumnya. *y-test* merupakan label atau nilai sesungguhnya dari hasil prakiraan.

Proses *testing* dilakukan untuk setiap ukuran *windows size* yang diuji. Hasil prakiraan yang didapat kemudian dibandingkan dengan nilai pada tahap evaluasi untuk mendapat nilai *windows size* yang akan digunakan pada tahap *streaming data.*

3.3.6 Evaluasi

Proses evaluasi dilakukan dengan melihat nilai *error* menggunakan metode *Root Mean Square Error* (RMSE) untuk melihat seberapa baik hasil prakiraan yang didapat pada tahap *testing* berdasarkan ukuran *windows size* yang digunakan. Pengujian RMSE ini akan dilakukan beberapa kali percobaan ukuran *windows size* hingga didapat nilai yang paling optimal. Setelah didapat nilai optimal menggunakan RMSE, selanjutnya akan dilakukan perhitungan persentase nilai *error* untuk data optimal yang didapat menggunakan rumus *Mean Absolute Percentage Error* (MAPE). Ukuran *windows size* dengan nilai RMSE terkecil selanjutnya akan digunakan pada tahap implementasi skenario untuk prakiraan secara *stream*.

3.3.7 Implementasi Skenario



Gambar 3. 9 Proses Prakiraan Secara *Stream*

Pada tahap ini dilakukan prakiraan kadar CO_2 secara *stream* menggunakan *Spark Streaming* dan *Spark MLlib*. Tahapan proses prakiraan secara *stream* dapat dilihat pada Gambar 3.9.

Data CO_2 dari sensor ditangkap menggunakan *broker RabbitMQ*. Data tersebut akan dikumpulkan hingga mencapai x data (x disesuaikan dengan ukuran windows size yang digunakan) kemudian diteruskan melalui soket ke S*park Streaming*. *Spark Streaming* akan menerima data dari *socket* dan memproses data tersebut secara *stream*.

Pada tahap ini, diterapkan metode *sliding window*, dimana sistem akan melakukan pembacaan data secara bergeser apabila ada data baru yang masuk. Misalnya diatur *windows size* sebanyak 5 data.

Pertama, 5 data pertama masuk, kemudian *Spark Streaming* akan langsung membaca data 1 sampai 5 untuk selanjutnya dilakukan proses *preprocessing* dan prakiraan menggunakan *Spark MLlib*. Selanjutnya, *socket* akan mengirimkan satu data baru atau data ke 6 ke *Spark Streaming*. *Spark streaming* kemudian membaca data 2 sampai 6 untuk selanjutnya dilakukan proses *preprocessing* dan prakiraan menggunakan *Spark MLlib*. Proses pergeseran data akan berlangsung terus menerus setiap *Spark Streaming* menerima data baru dari *socket*.

3.3.7 Analisis dan Kesimpulan

Pada tahap ini dilakukan analisis terhadap parameter yang diuji serta hasil prakiraan yang didapat untuk selanjutnya dilakukan pengambilan kesimpulan terhadap hasil yang telah didapat.

BAB IV HASIL DAN PEMBAHASAN

Pada bab ini akan dibahas mengenai hasil penelitian sesuai dengan alur yang telah dipaparkan pada bab sebelumnya. Hasil dari penelitian ini berupa sistem untuk dapat melakukan prakiraan CO_2 secara *stream* baik itu pada kadar normal, maupun di atas kadar normal berdasarkan ukuran parameter *windows size* terbaik yang didapatkan. Berdasarkan alur penelitian yang telah dipaparkan pada bab sebelumnya, penelitian ini mengikuti tahapan utama yaitu, mengumpulkan data, *preprocessing* data, *training*, *testing*, evaluasi, dan implementasi scenario streaming data.

4.1 Hasil Pegumpulan Data

Data yang digunakan dalam penelitian ini adalah data CO_2 dalam satuan ppm yang didapatkan dari lingkungan yang dibangun oleh peneliti. Lingkungan yang dibangun diatur untuk memperoleh kadar CO_2 yang berada di kadar normal serta diatas kadar normal. Pengambilan data diatas kadar normal diperoleh dengan memberikan faktor pemicu meningkatnya kadar CO_2 yaitu emisi karbon berupa pembakaran solar. Data diperoleh dari sensor *MQ-135* yang telah dikalibrasi sesuai dengan kadar CO_2 di atmosfer agar nilai CO_2 yang ditangkap sesuai dengan data CO_2 di udara.

Tahapan ini akan menghasilan dua jenis data yaitu data *batch* yang akan disimpan dalam bentuk csv dan data *stream*. *Data batch* akan digunakan untuk keperluan *training* dan *testing*, sedangkan data *stream* akan digunakan untuk

keperluan implementasi scenario data stream. Data yang diambil adalah data waktu dalam format *datetime* dan data CO_2 dalam format *float*.

Proses pengambilan data pada penelitian ini dimulai dengan membuat koneksi ke *MQTT Broker*. Untuk melakukan koneksi ke *MQTT Broker*, perlu untuk memasukkan *username, password*, serta alamat IP *dari master* seperti yang terlihat pada Gambar 4.1. Dari perintah tersebut dapat dilihat bahwa *master* dengan IP 10.2.3.74 memiliki username "arni" dan password "12345". Script yang digunakan dapat dilihat pada Gambar 4.1

```
def run(self):
    global conn
    self.username_pw_set("arni", "12345")
    self.connect("10.2.3.74", 1883, 60)
    self.subscribe("sensor", 0)
```

Gambar 4. 1 *Script* Koneksi MQTT

4.1.1 Mengambil Data Batch

Proses pengambilan data *batch* pada penelitian ini dilakukan menggunakan Bahasa *Python*. *Output* dari proses ini adalah data waktu dan CO_2 yang disimpan dalam format CSV. Pengambilan data ini dilakukan selama delapan hari dan diperoleh data sebanyak 11843 baris dengan format data disesuaikan dengan kebutuhan. Hasil dari pengambilan data *batch* ini akan digunakan pada tahap *training* dan *testing* untuk memperoleh nilai terbaik dari ukuran *windows size* yang digunakan. S*cript* yang digunakan dapat dilihat pada Gambar 4.2:

```
def on_message(client, userdata, msg):
    dt      = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    dtfile  = datetime.now().strftime("%Y%m%d%H%M%S")
    co2     = json.loads(msg.payload.decode())
    data    = [dt, co2]
```

```
    rows.append(data)

fname =
'/home/arni/stream/test/{}_{}.csv'.format(str(dtfile),str(u
uid.uuid1())) with open(fname, 'w') as f:
write = csv.writer(f)
```

Gambar 4. 2 *Script* Pengambilan Data Batch

Variable ‘dt’ dan ‘dtfile’ dalam perintah pada Gambar 4.2 merupakan variabel yang didefenisikan untuk mengambil waktu dalam format *datetime*. Berdasarkan *script* tersebut, format penulisan waktu yang akan didapatkan adalah “tahun-bulan-hari jam:menit:detik”. Variabel ‘co2’ dalam perintah diatas merupakan variabel yang didefenisikan untuk mengambil data CO_2 dalam format json. Variabel ‘data’ merupakan variabel yang didefenisikan untuk menuliskan data waktu dan CO_2. Berdasarkan *script* tersebut, data akan dituliskan secara berurut yaitu “waktu, co2”.

Output dari *script* Gambar 4.2 terlihat pada Gambar 4.3

```
1   datetime,co2
2   2022-09-01 09:54:00,156.2
3   2022-09-01 09:55:00,155.3559322
4   2022-09-01 09:56:00,155.0
5   2022-09-01 09:57:00,155.0833333
6   2022-09-01 09:58:00,154.7833333
7   2022-09-01 09:59:00,154.8833333
8   2022-09-01 10:00:00,154.6610169
9   2022-09-01 10:01:00,154.8333333
10  2022-09-01 10:02:00,154.6666667
11  2022-09-01 10:03:00,154.8333333
12  2022-09-01 10:04:00,155.0
13  2022-09-01 10:05:00,154.6333333
14  2022-09-01 10:06:00,154.8983051
15  2022-09-01 10:07:00,155.8333333
16  2022-09-01 10:08:00,157.55
17  2022-09-01 10:09:00,157.9666667
18  2022-09-01 10:10:00,158.1666667
19  2022-09-01 10:11:00,159.0
20  2022-09-01 10:12:00,158.8305085
21  2022-09-01 10:13:00,159.0
22  2022-09-01 10:14:00,158.8333333
23  2022-09-01 10:15:00,159.4666667
24  2022-09-01 10:16:00,159.9333333
25  2022-09-01 10:17:00,161.3666667
26  2022-09-01 10:18:00,162.1355932
27  2022-09-01 10:19:00,163.0
28  2022-09-01 10:20:00,162.85
```

Gambar 4. 3 *Output* Pengambilan *Data Batch*

Dari Gambar 4.3 terlihat bahwa data CO_2 berhasil diambil setiap setiap menit dengan format *datetime* "yyyy-mm-dd hh-mm-ss" dan format CO_2 berupa *float*. Tiap data juga telah tersusun dalam "waktu, co2". Format data yang terambil telah sesuai dengan kebutuhan, yaitu *datetime* untuk waktu dan *float* untuk CO_2. Persentase pengambilan data adalah 100%, artinya tidak ditemukan adanya data yang hilang selama pengambilan data.

4.1.2 Mengambil Data Stream

Data yang masuk pada *MQTT Broker* akan diteruskan melalui *socket stream*. Banyaknya data yang dikirim adalah sebanyak x data, tergantung ukuran *windows size* yang diatur. Pengambilan data dilakukan secara bergeser, artinya bahwa setiap ada data baru yang masuk, data pertama akan dihapus dan sistem akan membaca data kedua dan seterusnya. Misalnya windows size diatur sebanyak lima menit. Pengambilan data pertama akan dimulai dari menit ke satu hingga menit ke lima. Apabila data menit ke enam masuk, pengambilan data kedua akan dimulai dari menit kedua hingga menit ke enam. *Script* yang digunakan dapat dilihat pada Gambar 4.4

```
def on_message(self, mqttc, obj, msg):
dt=datetime.now().strftime("%Y-%m-%d %H:%M:%S")

y={"datetime":dt,"co2":float(msg.payload.decode())}z["data"].append(y)
  if len(z["data"]) == 5:
  print(z)
  conn, address = client_socket.accept()
  conn.send(json.dumps(z).encode())
  conn.close()
  z["data"].pop(0)
```

Gambar 4. 4 *Script* Pengambilan *Data Stream*

Format penulisan data yang akan terbaca adalah “datetime, co2” dimana format *datetime* adalah “tahun-bulan-hari jam:menit:detik” yang dijabarkan dalam variabel ‘dt’. Susunan data yang akan diterima juga akan diurutkan yaitu “waktu, co2” dimana format waktu dalam *datetime* dan co2 dalam *float* yang didefenisikan dalam variabel ‘y’.

Percobaan pengambilan data akan dilakukan tiap menit dan akan dikirim setelah mendapatkan data selama lima menit. Pendefenisian pengambilan data selama lima menit dijabarkan dalam perintah `if len(z["data"]) == 5:` yang berarti bahwa jika panjang data sama dengan lima, maka data akan dikirimkan ke *Spark Streaming*.

Output dari *script* Gambar 4.4 terlihat pada Gambar 4.5

```
Received PUBLISH (d0, q0, r0, m0), 'sensor', ...  (6 bytes)
Sending PINGREQ
Received PINGRESP
Received PUBLISH (d0, q0, r0, m0), 'sensor', ...  (6 bytes)
{'data': [{'datetime': '2022-11-30 15:30:54', 'co2': 615.5}, {'datetime': '2022
-11-30 15:31:54', 'co2': 589.37}, {'datetime': '2022-11-30 15:32:54', 'co2': 56
3.98}, {'datetime': '2022-11-30 15:33:54', 'co2': 589.37}, {'datetime': '2022-1
1-30 15:34:54', 'co2': 563.98}]}
Sending PINGREQ
Received PINGRESP
Received PUBLISH (d0, q0, r0, m0), 'sensor', ...  (6 bytes)
{'data': [{'datetime': '2022-11-30 15:31:54', 'co2': 589.37}, {'datetime': '202
2-11-30 15:32:54', 'co2': 563.98}, {'datetime': '2022-11-30 15:33:54', 'co2': 5
89.37}, {'datetime': '2022-11-30 15:34:54', 'co2': 563.98}, {'datetime': '2022-
11-30 15:35:54', 'co2': 563.98}]}
Sending PINGREQ
Received PINGRESP
Received PUBLISH (d0, q0, r0, m0), 'sensor', ...  (6 bytes)
{'data': [{'datetime': '2022-11-30 15:32:54', 'co2': 563.98}, {'datetime': '202
2-11-30 15:33:54', 'co2': 589.37}, {'datetime': '2022-11-30 15:34:54', 'co2': 5
63.98}, {'datetime': '2022-11-30 15:35:54', 'co2': 563.98}, {'datetime': '2022-
11-30 15:36:54', 'co2': 563.98}]}
```

Gambar 4. 5 Hasil Pengambilan Data Secara *Stream*

Dari Gambar 4.5 terlihat bahwa sistem berhasil membaca data CO_2 secara *stream* tiap menit dengan ukuran data sebesar 6 bytes dan dikumpulkan hingga terdapat lima data dengan format penulisan data yaitu “waktu, co2”. Banyaknya

data ini tergantung dari ukuran *windows size* yang dimasukkan. Dari hasil di atas juga terlihat bahwa setiap sistem membaca satu data baru, data pertama akan dihapus dan membaca lima data berikutnya. Misalnya untuk pengambilan data lima pertama sesuai hasil di atas, data pertama diambil pada waktu 15:30:54, data kedua diambil pada waktu 15:31:54, data ketiga diambil pada waktu 15:32:54, data keempat diambil pada waktu 15:33:54, dan data kelima diambil pada waktu 15:34:54. Setelah mencapai data kelima, data dikirimkan ke *Spark Streaming* dan dilakukan pengambilan data berikutnya.

Untuk pengambilan lima data berikutnya, data pertama diambil pada waktu 15:31:54, data kedua diambil pada waktu 15:32:54, data ketiga diambil pada waktu 15:33:54, data keempat diambil pada waktu 15:34:54, dan data kelima diambil pada waktu 15:35:54.

Dari kedua pengambilan data di atas, pengambilan data pertama dimulai pada waktu 15:30:54 dan berakhir pada waktu 15:34:54. Untuk pengambilan data kedua dimulai pada waktu 15:31:54 dan berakhir pada waktu 15:35:54. Hal ini membuktikan bahwa pengambilan data dilakukan secara bergeser, yaitu data pertama akan dihapus apabila terdapat satu data baru yang masuk.

4.2 Preprocessing Data

Tahap ini menghasilkan data yang siap digunakan untuk proses *training*, *testing*, dan implementasi *data stream*. Data yang siap digunakan merupakan data yang telah berbentuk *vector*. Ada beberapa tahapan dalam *preprocessing* data yaitu:

4.2.1 Preprocessing Data Training dan Testing

1. Transformasi Data I

Tahapan ini berfungsi untuk mengubah data CSV ke dalam *spark dataframe*. *Script* yang digunakan dapat dilihat pada Gambar 4.6.

```
df = spark.read.csv(lp, inferSchema=True,
header=True)
```

Gambar 4. 6 *Script* Mengubah CSV ke *Spark Dataframe*

2. Transformasi Data II

Tahapan ini berfungsi untuk mengubah *spark dataframe* ke dalam *pandas dataframe* agar dapat digunakan untuk keperluan *windowing*. *Script* yang digunakan dapat dilihat pada Gambar 4.7.

```
df = df.toPandas()
df = df.set_index('datetime')
```

Gambar 4. 7 *Script* Mengubah *Spark Dataframe* ke *Pandas Dataframe*

3. Windowing

Tahap ini merupakan tahap untuk mengatur banyaknya data yang akan digunakan untuk meramalkan data satu waktu berikutnya. *Script* yang digunakan dapat dilihat pada Gambar 4.8.

```
df_sliding = dataset['co2'].astype(float).values
X,y=sliding_window(df_sliding, window_size, ph)
window_size = 5
ph=1
```

Gambar 4. 8 *Script Windowing*

Berdasarkan *script* pada Gambar 4.8, ukuran *windows* yang digunakan adalah 5 dengan *prediction horizon* (ph) 1, artinya bahwa data yang akan dijadikan fitur adalah data sebanyak 5 baris atau 5 menit untuk meramalkan data 1 menit berikutnya. Atur kapasitas *windows size* pada langkah ini untuk pengujian ukuran *windows size* satu sampai sepuluh.

4. Transformasi Data III

Tahap ini merupakan tahap untuk mengubah hasil *windowing* yang masih berbentuk *pandas dataframe* ke *spark dataframe* agar dapat digunakan untuk tahap *preprocessing* berikutnya. *Script* yang digunakan dapat dilihat pada Gambar 4.9.

```
test_data = pd.DataFrame(X)
test_data['y'] = pd.DataFrame(y)

test_spark = spark.createDataFrame(test_data)
test_spark.show(5)
```

Gambar 4. 9 *Script* Mengubah *Pandas Dataframe* ke *Spark Datarfame*

5. Vector Assembler

Tahap ini merupakan tahap untuk mengubah data dalam s*park dataframe* ke dalam bentuk *vector*. Mengubah format *dataframe* menjadi *vector* perlu dilakukan karena S*park MLlib* hanya dapat mengolah data dalam bentuk *vector*. Hasil dari *vector assembler* berbentuk RDD. *Script* yang digunakan dapat dilihat pada Gambar 4.10.

```
stage_1 = VectorAssembler(inputCols=[ '0', '1',
'2', '3', '4' ], outputCol="features")


prediction = model.transform(test_spark)
prediction.select('features','prediction','y').show()
```

Gambar 4. 10 *Script* Mengubah *Spark Dataframe* ke Vektor

4.2.2 Preprocessing Data Stream

1. Transformasi Data 1

Tahapan ini berfungsi untuk mengubah data JSON ke dalam *pandas dataframe*. *Script* yang digunakan dapat dilihat pada Gambar 4.11.

```
data = test_spark.collect()
        data = json.loads(data[0])
```

```
print(data["data"])
df = pd.json_normalize(data["data"])
print(df)
```

Gambar 4. 11 Script Mengubah JSON ke *Pandas Dataframe*

2. Transformasi Data II

Tahapan ini berfungsi untuk mengubah data dalam *pandas dataframe* ke *spark dataframe* agar dapat digunakan untuk tahap *praprocessing* berikutnya. *Script* yang digunakan dapat dilihat pada Gambar 4.12.

```
df_slide = pd.DataFrame([df["co2"].to_list()])
test_spark = spark.createDataFrame(df_slide)
```

Gambar 4. 12 Script Mengubah *Pandas Dataframe* ke *Spark Dataframe*

3. Vector Assembler

Tahap ini merupakan tahap untuk mengubah data dalam s*park dataframe* ke dalam bentuk *vector*. Mengubah format *dataframe* menjadi *vector* perlu dilakukan karena S*park MLlib* hanya dapat mengolah data dalam bentuk *vector*. Hasil dari *vector assembler* berbentuk RDD. *Script* yang digunakan dapat dilihat pada Gambar 4.13.

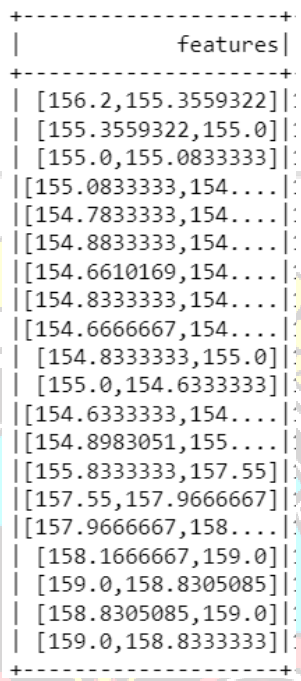
```
prediction = model.transform(test_spark)
prediction.select('features','prediction').show()

prediction = model.transform(test_spark)
prediction.select('features','prediction').show()
```

Gambar 4. 13 Script Mengubah *Spark Dataframe* ke Vektor

Tahap *preprocessing* akan menghasilkan data yang siap untuk diolah pada *Spark MLlib*, yaitu data dalam bentuk *vector*. Berdasarkan *script* pada Gambar 4.10, *input* yang akan digunakan sesuai yang dijabarkan dalam variable `inputCols` adalah sebanyak lima index yaitu index ke nol hingga index ke empat sesuai dengan ukuran *windows size* yang digunakan yaitu lima. Hasil dari *vector*

assembler akan disimpan dalam kolom *features* sesuai yang dijabarkan dalam variabel `outputCol` seperti yang terlihat pada Gambar 4.14



Gambar 4. 14 Hasil Tahapan *Preprocessing*

Dari Gambar 4.14 terlihat bahwa data telah berhasil diubah ke dalam bentuk *vector* dan disimpan ke dalam kolom *features*. Banyaknya index *vector* pada tiap baris tergantung dari besarnya ukuran *windows size* yang digunakan.

4.3 Training

Tahap *training* akan menghasilkan model prakiraan yang akan digunakan pada tahap *testing*. Proses *training* akan dilakukan untuk tiap ukuran *windows size* agar diperoleh model yang sesuai untuk setiap ukuran *windows size* yang akan diuji.

Tahapan *training* dimulai ketika sistem membaca data pada folder *train*. Data tersebut kemudian akan dipisah menjadi dua yaitu *X-train* dan *y-train*. *X-train* berisi

data yang akan dipelajari untuk prakiraan, dan *y-train* merupakan label atau data berikutnya yang akan menjadi hasil prakiraan. *Script* yang digunakan dapat dilihat pada Gambar 4.15.

```
X,y=sliding_window(df_sliding, window_size, ph)


train_data = pd.DataFrame(X)
train_data['y'] = pd.DataFrame(y)
```

Gambar 4. 15 *Script* Membagi Data Train

Output dari *script* Gambar 4.15 terlihat pada Gambar 4.16

```
+-----------+-----------+-----------+-----------+-----------+-----------+
|          0|          1|          2|          3|          4|          y|
+-----------+-----------+-----------+-----------+-----------+-----------+
|      156.2|155.3559322|      155.0|155.0833333|154.7833333|154.8833333|
|155.3559322|      155.0|155.0833333|154.7833333|154.8833333|154.6610169|
|      155.0|155.0833333|154.7833333|154.8833333|154.6610169|154.8333333|
|155.0833333|154.7833333|154.8833333|154.6610169|154.8333333|154.6666667|
|154.7833333|154.8833333|154.6610169|154.8333333|154.6666667|154.8333333|
+-----------+-----------+-----------+-----------+-----------+-----------+
only showing top 5 rows
```

Gambar 4. 16 Hasil Tahapan *Training*

Dari Gambar 4.16, terlihat bahwa data berhasil dikelompokkan menjadi *X-train* dan *y-train*. Data pada kolom '0', '1', '2', '3', dan '4' merupakan *X-train* dan data pada kolom 'y' merupakan *y-train*. Banyaknya *X-train* tergantung dari besarnya ukuran *windows size* yang digunakan.

X-train dan *y-train* selanjutnya akan diolah menggunakan algoritma *Random Forest* hingga didapatkan model prakiraan. *Script* yang digunakan dapat dilihat pada Gambar 4.17.

```
model = RandomForestRegressor(featuresCol='features',
labelCol='y')
```

Gambar 4. 17 *Script* Algoritma *Random Forest*

Apabila model telah berhasil didapatkan, selanjutnya akan dilanjutkan proses *testing* menggunakan model prakiraan yang didapat.

4.4 Testing

Tahap *testing* akan menghasilkan nilai prakiraan untuk tiap ukuran *windows size* yang selanjutnya akan dievaluasi untuk mengetahui nilai terbaik yang diperoleh untuk tiap ukuran *windows size* yang digunakan.

Sama seperti *training*, pada tahap *testing* data juga akan dipisah menjadi dua yaitu *X-test* dan *y-test*. *X-test* menggunakan data yang telah diubah kedalam *vector* dan akan digunakan untuk mendapatkan hasil prakiraan berdasarkan model yang telah didapat sebelumnya. *Y-test* merupakan label atau nilai sesungguhnya dari hasil prakiraan. Hasil prakiraan yang didapat kemudian dibandingkan dengan nilai sesungguhnya pada tahap evaluasi. *Script* yang digunakan dapat dilihat pada Gambar 4.18.

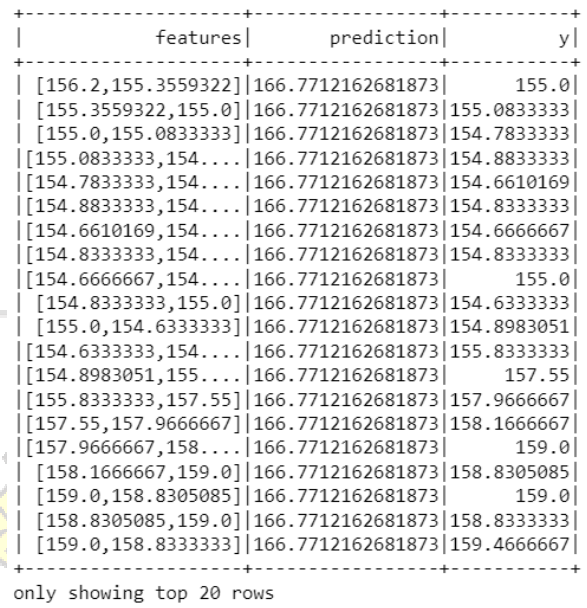
```
X,y=sliding_window(df_sliding, window_size, ph)

test_data = pd.DataFrame(X)
test_data['y'] = pd.DataFrame(y)

prediction = model.transform(test_spark)
prediction.select('features','prediction','y').show()
```

Gambar 4. 18 *Script* Pembagian *Data Test*

Output dari *script* Gambar 4.18 terlihat pada Gambar 4.19



```
+--------------------+-----------------+-----------+
|            features|       prediction|          y|
+--------------------+-----------------+-----------+
| [156.2,155.3559322]|166.7712162681873|      155.0|
| [155.3559322,155.0]|166.7712162681873|155.0833333|
| [155.0,155.0833333]|166.7712162681873|154.7833333|
|[155.0833333,154....|166.7712162681873|154.8833333|
|[154.7833333,154....|166.7712162681873|154.6610169|
|[154.8833333,154....|166.7712162681873|154.8333333|
|[154.6610169,154....|166.7712162681873|154.6666667|
|[154.8333333,154....|166.7712162681873|154.8333333|
|[154.6666667,154....|166.7712162681873|      155.0|
| [154.8333333,155.0]|166.7712162681873|154.6333333|
| [155.0,154.6333333]|166.7712162681873|154.8983051|
|[154.6333333,154....|166.7712162681873|155.8333333|
|[154.8983051,155....|166.7712162681873|     157.55|
|[155.8333333,157.55]|166.7712162681873|157.9666667|
|[157.55,157.9666667]|166.7712162681873|158.1666667|
|[157.9666667,158....|166.7712162681873|      159.0|
| [158.1666667,159.0]|166.7712162681873|158.8305085|
| [159.0,158.8305085]|166.7712162681873|      159.0|
| [158.8305085,159.0]|166.7712162681873|158.8333333|
| [159.0,158.8333333]|166.7712162681873|159.4666667|
+--------------------+-----------------+-----------+
only showing top 20 rows
```

Gambar 4. 19 Hasil Tahapan *Testing*

Dari Gambar 4.19, terlihat bahwa sistem berhasil melakukan prakiraan data CO_2 menggunakan model yang didapat pada tahap *training*. Kolom *features* berisi data *vector* sebagai *input*, kolom *prediction* berisi data hasil prakiraan, dan kolom y merupakan data sesungguhnya dari hasil prakiraan.

4.5 Evaluasi

Evaluasi dilakukan menggunakan metode *Root Mean Square Error* (RMSE). Tahap evaluasi menghasilkan nilai RMSE terbaik dari ukuran *windows size* yang digunakan. Nilai RMSE terbaik akan digunakan pada tahap implementasi. Pengujian nilai RMSE akan dilakukan hingga didapat nilai paling optimal. Tiap ukuran *windows size* akan diuji sebanyak sepuluh kali dan akan diambil nilai rata-rata dari sepuluh kali percobaan tersebut. Adapun prinsip kerja dari RMSE adalah semakin rendah nilai RMSE maka hasil prakiraan yang didapat semakin mendekati nilai yang sebenarnya. *Script* yang digunakan dapat dilihat pada Gambar 4.20.

```
prediction = model.transform(test_spark)
prediction.select('features','prediction','y').show()
evaluator = RegressionEvaluator(labelCol="y",
predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(prediction)
```

Gambar 4. 20 *Script* Pengecekan Nilai RMSE

Output dari *script* Gambar 4.20 terlihat pada Gambar 4.21

```
Root Mean Squared Error (RMSE) on test data = 12.0406
```

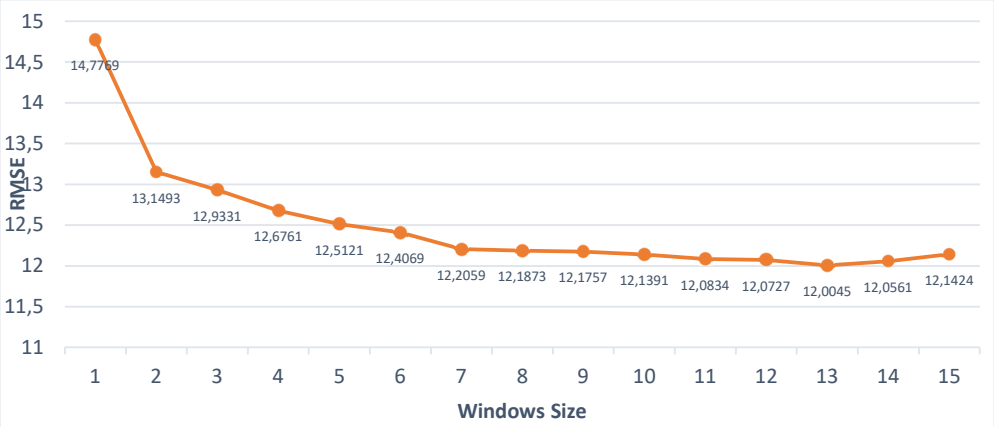
Gambar 4. 21 Hasil Pengujian RMSE

Gambar 4.21 merupakan sampel dari hasil pengujian RMSE. Hasil diatas merupakah hasil pengujian RMSE untuk ukuran windows size 13 yang telah dilakukan. Pengujian RMSE akan dilakukan untuk beberapa ukuran *windows size* hingga didapatkan nilai optimal. Untuk menghindari nilai yang didapat adalah nilai kebetulan, tiap ukuran *windows size* akan diuji sebanyak sepuluh kali. Untuk hasil yang lebih jelas, akan disajikan ke dalam Tabel 4.1 dan diagram pada Gambar 4.22

Tabel 4. 1 Tabel RMSE

WS	RMSE										Mean
	1	2	3	4	5	6	7	8	9	10	
1	14,7062	14,5977	14,7551	14,5806	14,8015	15,0408	14,7607	14,8970	14,8414	14,7878	14,7769
2	13,1133	13,0735	12,9336	13,1039	13,1309	13,3750	13,0379	13,2134	13,1523	13,3588	13,1493
3	12,6630	12,8911	12,9576	13,0147	13,0905	12,9787	13,0087	13,0275	12,8998	12,7996	12,9331
4	12,5909	12,6188	12,8168	12,7578	12,5353	12,6684	12,6703	12,7234	12,7778	12,6019	12,6761
5	12,4875	12,5926	12,4978	12,5848	12,4616	12,4580	12,4930	12,5678	12,4564	12,5212	12,5121
6	12,4486	12,2058	12,4835	12,4829	12,4686	12,3148	12,4517	12,4573	12,3316	12,4244	12,4069
7	12,2812	12,1637	12,2807	12,3996	12,1622	12,3493	12,0613	12,0408	12,2376	12,0830	12,2059
8	12,0492	12,1909	12,2910	12,2318	12,2702	12,2858	12,2017	12,1958	12,1692	11,9873	12,1873
9	12,1405	12,2336	11,9117	12,1305	12,2493	12,2390	12,2542	12,2493	12,1669	12,1823	12,1757
10	12,1615	11,9698	12,3197	12,0840	12,0876	12,2410	12,2420	12,1288	12,1050	12,0520	12,1391
11	12,0523	11,8691	12,1895	12,0761	12,0487	11,9190	12,1930	12,1725	12,0392	12,2749	12,0834
12	12,0462	12,1770	12,1930	11,9787	12,0426	12,1209	12,1411	12,0227	12,0465	11,9586	12,0727
13	12,0958	12,0406	12,1096	11,9935	12,0054	11,8003	12,1695	11,9897	11,8679	11,9729	12,0045
14	11,9785	11,7920	12,1725	11,9642	12,1227	12,0256	12,1852	12,1559	12,2944	11,8704	12,0561
15	12,2866	12,1304	11,9661	12,2009	12,0899	12,0515	12,2267	12,3491	12,0808	12,0416	12,1424

UJUNG PANDANG



Gambar 4. 22 Visualisasi Grafik RMSE

Dari Tabel 4.1 dan diagram pada Gambar 4.22 terlihat bahwa hasil RMSE terbaik didapat pada *windows size* ke-13 yaitu 12,0045 yang berarti bahwa kesalahan terkecil dari prakiraan didapatkan dengan menggunakan data selama 13 menit untuk memprediksi menit ke-14. Dikatakan bahwa nilai tersebut adalah nilai yang paling optimal karena nilai RMSE pada *windows size* ke-13 merupakan nilai terkecil dari semua percobaan yang dilakukan. Hal ini sesuai dengan prinsip dari RMSE bahwa semakin kecil nilai RMSE, maka semakin bagus hasil yang didapatkan. Dari hasil diatas juga terlihat bahwa nilai RMSE mengalami penurunan nilai optimal apabila nilai *windows size* berada lebih dari 13. Untuk selanjutnya, ukuran *windows size* yang akan digunakan pada tahap implementasi adalah *windows size* yang memiliki nilai error paling kecil, yaitu *windows size* ke-13.

Untuk membuktikan hasil perhitungan RMSE yang di dapat, dilakukan perhitungan manual dengan RMSE *windows size* ke-13 sebagai sampel. Hasil RMSE yang didapat adalah 12,0406. Berikut persamaan yang digunakan:

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n}(At - Ft)^2}{n}}$$

Dengan:

At = Nilai data aktual

Ft = Nilai hasil prakiraan

N= Banyaknya data

∑ = Summation (Jumlah keseluruhan nilai)

$$RMSE = \sqrt{\frac{1714914{,}27262617}{11829}}$$

$$RMSE = \sqrt{144{,}97542248932}$$

$$RMSE = 12{,}0405740$$

Berdasarkan hasil perhitungan di atas, didapatkan nilai RMSE sebesar 12,0405740. Hasil tersebut sesuai dengan hasil perhitungan RMSE yang didapat pada Gambar 4. 21 menggunakan *script* pada Gambar 4.20.

Nilai optimal yang didapat dapat dievaluasi lebih lanjut menggunakan rumus *Mean Absolute Percentage Error* (MAPE) untuk melihat seberapa akurat hasil yang didapat dalam bentuk persen. Berikut persamaan yang digunakan

$$\text{MAPE} = \frac{\sum_{t=1}^{n} |(\frac{At-Ft}{At})100|}{n}$$

Dengan:

At = Nilai data aktual

Ft = Nilai hasil prakiraan

N= banyaknya data

$\sum$ = Summation (Jumlah keseluruhan nilai)

$$\text{MAPE} = \frac{19121{,}450399009}{11829}$$

$$\text{MAPE} = 1{,}616489171$$

Dari hasil perhitungan diatas, didapat nilai MAPE sebesar 1,616489171% dimana hasil ini menunjukkan bahwa persentase nilai error lebih kecil dari 10%. Ini membuktikan bahwa ketepatan prakiraan yang didapat bernilai sangat baik.

4.5 Implementasi Skenario

Prakiraan kadar CO_2 akan dilakukan secara *stream*. Prakiraan data secara *stream* akan menggunakan *MQTT Broker* dan *Spark Streaming*. Pertama, diperlukan koneksi ke *MQTT Broker* untuk memperoleh data waktu dan CO_2 dari sensor kemudian meneruskan data waktu dan CO_2 melalui *socket stream*. *Script* yang digunakan dapat dilihat pada Gambar 4.23.

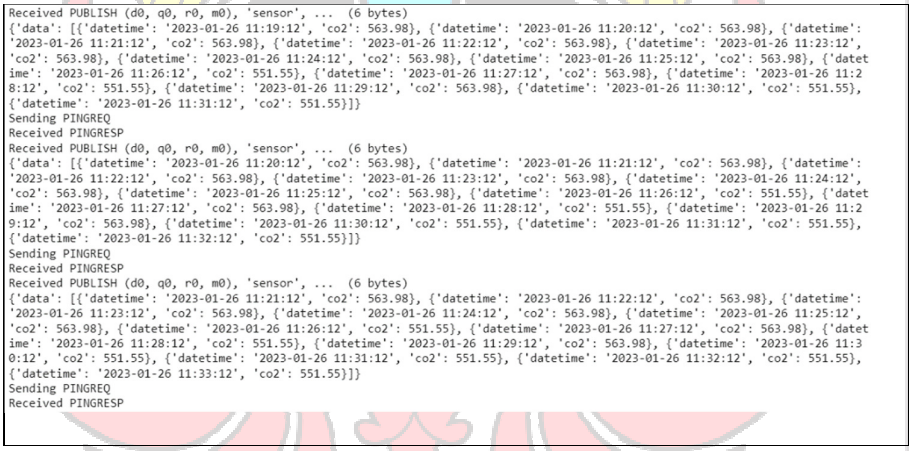
```
def on_message(self, mqttc, obj, msg):
 dt   = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
 y = {"datetime":dt, "co2": float(msg.payload.decode())}
 z["data"].append(y)
 if len(z["data"]) == 13:
    print(z)
    conn, address = client_socket.accept()
    conn.send(json.dumps(z).encode())
    conn.close()
```

Gambar 4. 23 *Script* Koneksi MQTT Broker

Berdasarkan perintah pada Gambar 4.23, *socket* akan mengirimkan data ke *Spark Streaming* apabila sudah terdapat 13 data. Data yang akan dikirimkan adalah data waktu dalam bentuk *datetime* dan data CO_2 dalam bentuk *float*.

Untuk data pertama, *MQTT Broker* akan menunggu hingga mencapai 13 data, selanjutnya apabila terdapat satu data baru yang masuk, sistem diatur untuk menghapus data pertama dan mengirimkan data 13 berikutnya.

Output dari script Gambar 4.23 terlihat pada Gambar 4.24



Gambar 4. 24 Pengambilan *Data Stream* Pada *MQTT Broker*

Gambar 4.24 menunjukkan bahwa data waktu dan CO_2 berhasil diambil tiap menit. Tiap data dikumpulkan selama 13 menit untuk selanjutnya dikirmkkan ke *Spark*.

Untuk mengolah data pada S*park*, perlu dijalankan *Spark Streaming* untuk menerima data secara *stream*. *Script* yang digunakan dapat dilihat pada Gambar 4.25.

```
PerediksiData = ssc.socketTextStream('10.2.3.74', 65432)
```

Gambar 4.25 *Script* Menjalankan *Spark Streaming*

Output dari *script* Gambar 4.25 terlihat pada Gambar 4.26


```
[{'datetime': '2023-01-26 11:16:12', 'co2': 563.98}, {'datetime': '2023-01-26 11:17:12', 'co2': 563.98}, {'datetime': '2023-01-
26 11:18:12', 'co2': 563.98}, {'datetime': '2023-01-26 11:19:12', 'co2': 563.98}, {'datetime': '2023-01-26 11:20:12', 'co2': 56
3.98}, {'datetime': '2023-01-26 11:21:12', 'co2': 563.98}, {'datetime': '2023-01-26 11:22:12', 'co2': 563.98}, {'datetime': '20
23-01-26 11:23:12', 'co2': 563.98}, {'datetime': '2023-01-26 11:24:12', 'co2': 563.98}, {'datetime': '2023-01-26 11:25:12', 'co
2': 563.98}, {'datetime': '2023-01-26 11:26:12', 'co2': 551.55}, {'datetime': '2023-01-26 11:27:12', 'co2': 563.98}, {'datetim
e': '2023-01-26 11:28:12', 'co2': 551.55}]
             datetime     co2
0   2023-01-26 11:16:12  563.98
1   2023-01-26 11:17:12  563.98
2   2023-01-26 11:18:12  563.98
3   2023-01-26 11:19:12  563.98
4   2023-01-26 11:20:12  563.98
5   2023-01-26 11:21:12  563.98
6   2023-01-26 11:22:12  563.98
7   2023-01-26 11:23:12  563.98
8   2023-01-26 11:24:12  563.98
9   2023-01-26 11:25:12  563.98
10  2023-01-26 11:26:12  551.55
11  2023-01-26 11:27:12  563.98
12  2023-01-26 11:28:12  551.55
```


Gambar 4. 26 Hasil Pembacaan Data Secara Stream pada Spark

Gambar 4.26, menunjukkan bahwa data yang masuk di S*park* adalah 13 data sesuai dengan ukuran *windows size* yang digunakan. Data yang terbaca adalah data waktu dalam bentuk *datetime* dan data CO_2 dalam bentuk *float* yang diambil setiap menit.

Selanjutnya, data yang telah dibaca oleh *Spark Streaming* akan diolah untuk mendapatkan hasil prakiraan secara *stream*. Sebelum dijalankan perintah prakiraan, data terlebih dahulu akan di-*praprocessing* seperti yang telah dijelaskan pada bagian 4.2.

Prakiraan data akan berlangsung secara kontinu berdasarkan waktu kedatangan data pada *Spark MLlib*. Setiap *Spark Streaming* menerima data dari *MQTT Broker*, *Spark Streaming* akan mengirimkan data tersebut ke *Spark MLlib*, selanjutnya *Spark MLlib* akan melakukan prakiraan terhadap data tersebut. Hasil prakiraan CO_2 yang didapat ditunjukkan pada Gambar 4.27 dan Gambar 4.28

```
[{'datetime': '2023-01-26 13:26:54', 'co2': 406.06}, {'datetime': '2023-01-26 13:27:54', 'co2': 406.06}, {'datetime': '2023-01-
26 13:28:54', 'co2': 406.06}, {'datetime': '2023-01-26 13:29:54', 'co2': 406.06}, {'datetime': '2023-01-26 13:30:54', 'co2': 39
6.06}, {'datetime': '2023-01-26 13:31:54', 'co2': 396.06}, {'datetime': '2023-01-26 13:32:54', 'co2': 396.06}, {'datetime': '20
23-01-26 13:33:54', 'co2': 396.06}, {'datetime': '2023-01-26 13:34:54', 'co2': 406.06}, {'datetime': '2023-01-26 13:35:54', 'co
2': 406.06}, {'datetime': '2023-01-26 13:36:54', 'co2': 396.06}, {'datetime': '2023-01-26 13:37:54', 'co2': 396.06}, {'datetim
e': '2023-01-26 13:38:54', 'co2': 396.06}]
              datetime     co2
0   2023-01-26 13:26:54  406.06
1   2023-01-26 13:27:54  406.06
2   2023-01-26 13:28:54  406.06
3   2023-01-26 13:29:54  406.06
4   2023-01-26 13:30:54  396.06
5   2023-01-26 13:31:54  396.06
6   2023-01-26 13:32:54  396.06
7   2023-01-26 13:33:54  396.06
8   2023-01-26 13:34:54  406.06
9   2023-01-26 13:35:54  406.06
10  2023-01-26 13:36:54  396.06
11  2023-01-26 13:37:54  396.06
12  2023-01-26 13:38:54  396.06
+--------------------+-----------------+
|            features|       prediction|
+--------------------+-----------------+
|[406.06,406.06,40...|438.242500603805|
+--------------------+-----------------+
```

Gambar 4. 27 Prakiraan CO_2 pada Kadar Normal

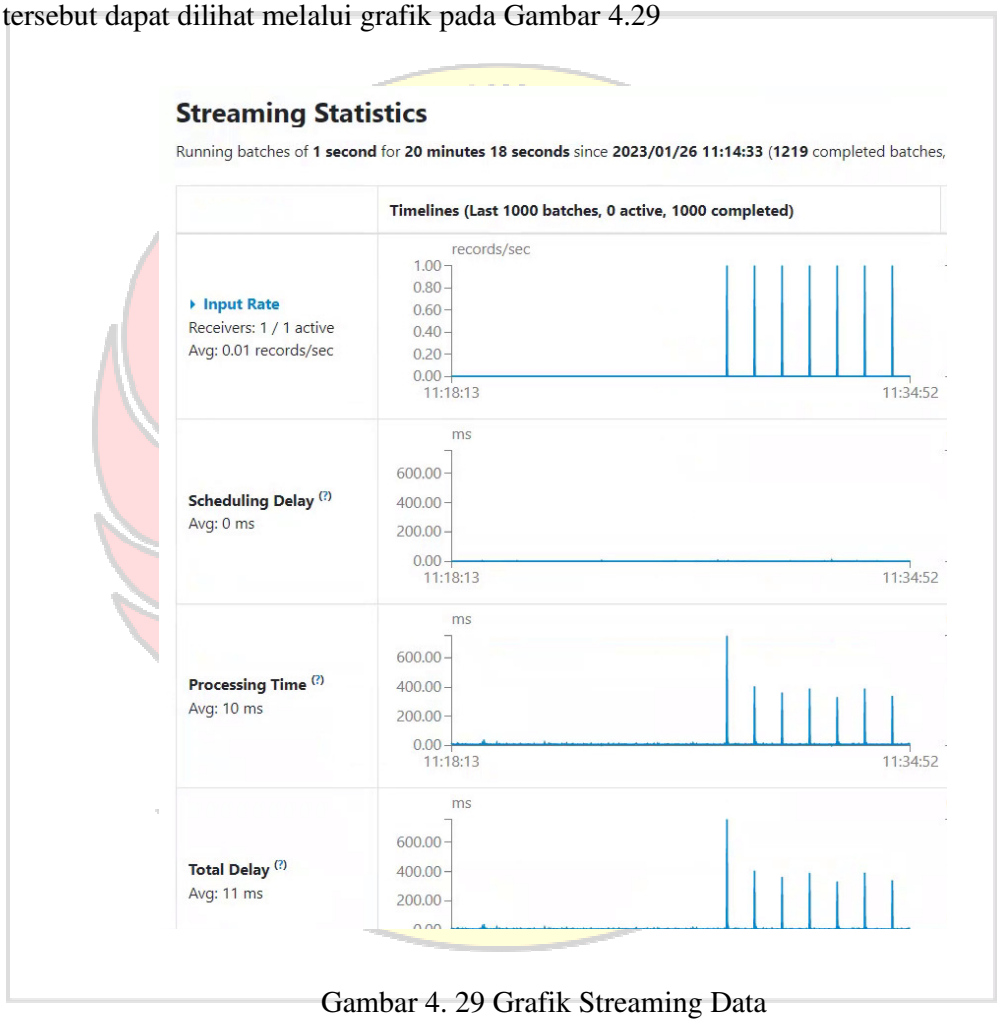
```
[{'datetime': '2023-01-26 11:28:12', 'co2': 551.55}, {'datetime': '2023-01-26 11:29:12', 'co2': 563.98}, {'datetime': '2023-01-
26 11:30:12', 'co2': 551.55}, {'datetime': '2023-01-26 11:31:12', 'co2': 551.55}, {'datetime': '2023-01-26 11:32:12', 'co2': 55
1.55}, {'datetime': '2023-01-26 11:33:12', 'co2': 551.55}, {'datetime': '2023-01-26 11:34:12', 'co2': 551.55}, {'datetime': '20
23-01-26 11:35:12', 'co2': 551.55}, {'datetime': '2023-01-26 11:36:12', 'co2': 551.55}, {'datetime': '2023-01-26 11:37:12', 'co
2': 551.55}, {'datetime': '2023-01-26 11:38:12', 'co2': 628.85}, {'datetime': '2023-01-26 11:39:12', 'co2': 628.85}, {'datetim
e': '2023-01-26 11:40:12', 'co2': 628.85}]
              datetime     co2
0   2023-01-26 11:28:12  551.55
1   2023-01-26 11:29:12  563.98
2   2023-01-26 11:30:12  551.55
3   2023-01-26 11:31:12  551.55
4   2023-01-26 11:32:12  551.55
5   2023-01-26 11:33:12  551.55
6   2023-01-26 11:34:12  551.55
7   2023-01-26 11:35:12  551.55
8   2023-01-26 11:36:12  551.55
9   2023-01-26 11:37:12  551.55
10  2023-01-26 11:38:12  628.85
11  2023-01-26 11:39:12  628.85
12  2023-01-26 11:40:12  628.85
+--------------------+-----------------+
|            features|       prediction|
+--------------------+-----------------+
|[551.55,563.98,55...|616.3088444478482|
+--------------------+-----------------+
```

Gambar 4. 28 Prakiraan CO_2 di Atas Kadar Normal

Gambar 4.27 dan 4.28 menunjukkan bahwa sistem yang dibangun berhasil meramalkan kadar CO_2 baik itu pada kadar normal maupun di atas kadar normal. Gambar 4.27 merupakan hasil prakiraan CO_2 yang diatur pada kadar normal dengan hasil prakiraan 438.2425, dan Gambar 4.28 merupakah hasil prakiraan CO_2 yang diatur di atas kadar normal, dengan hasil prakiraan yang didapat adalah 616.3088. Hasil prakiraan yang didapat sesuai dengan data yang dipelajari sebanyak *windows size* yang digunakan yaitu 13 menit data. Hal tersebut membuktikan bahwa nilai

optimal RMSE yang didapat pada *windows size* 13 mampu memberikan hasil prakiraan yang sesuai.

Streaming data berhasil dilakukan tanpa ada data yang hilang, artinya bahwa sistem berhasil memproses tiap data yang masuk tanpa adanya antrian data. Hal tersebut dapat dilihat melalui grafik pada Gambar 4.29



Gambar 4. 29 Grafik Streaming Data

Gambar 4.29 merupakan grafik dari pengambilan data yang dilakukan pada *Spark Streaming*. Pada bagian "*Input Rate*" terlihat bahwa tiap menit grafik secara konsisten berada pada nilai 1.00, hal ini menunjukkan bahwa spark konsisten menerima satu input dari sumber data tiap menit. Pada bagian "*Processing Time*"

menunjukkan bahwa kumpulan data telah diproses dalam waktu 10 ms. Waktu rata-rata yang dibutuhkan bernilai lebih kecil dari *interval batch* (1 detik) berarti tidak ada data yang mengantri selama proses *streaming* data berlangsung, atau dapat dikatakan bahwa tiap *batch data* diproses secepat data tersebut diterima. Hal ini dapat dilihat pada bagian "*Scheduling Delay*" yang menunjukkan nilai rata-rata sebesar 0 ms.

Hasil pengujian yang telah dilakukan menunjukkan bahwa prakiraan kadar CO_2 dimasa yang akan datang berhasil dilakukan pada *Spark MLlib*. Sistem prakiraan berhasil meramalkan kadar CO_2 baik itu pada kadar normal maupun di atas kadar normal berdasarkan data yang dipelajari sebanyak ukuran *windows size* yang diberikan. Ukuran *windows size* yang digunakan adalah *windows size* 13 karena nilai tersebut adalah nilai yang paling optimal berdasarkan proses *training* dan *testing* yang dilakukan. Hal ini dibuktikan dengan hasil perhitungan RMSE yang didapat memperoleh hasil paling kecil yaitu 12,0045 dengan persentase error sebesar 1,616489171%.

BAB IV PENUTUP

5.1 Kesimpulan

Berdasarkan hasil pengujian, penelitian ini menghasilkan beberapa kesimpulan bahwa:

1. Penelitian ini berhasil melakukan prakiraan CO_2 di masa yang akan datang secara *streaming* menggunakan *Spark MLlib,* baik itu prakiraan kadar CO_2 pada kadar normal maupun diatas kadar normal. Hasil prakiraan yang didapat pada kadar normal adalah 438.2425, dan hasil prakiraan yang didapat diatas kadar normal adalah 616.3088.
2. Akurasi maksimal berhasil didapatkan pada *windows size* 13. Hal ini dibuktikan dari hasil perhitungan RMSE yang mendapat nilai paling kecil pada *windows size* 13 yaitu sebesar 12,0045 dan hasil perhitungan persentase error berada dibawah 10% yaitu sebesar 1,616489171%.

5.2 Saran

Berdasarkan penelitian ini, maka saran untuk penelitian selanjutnya ialah sebagai berikut:

1. Penelitian selanjutnya diharapkan untuk menambah data yang digunakan untuk membangun training model dan testing agar akurasi yang didapat lebih maksimal.
2. Penelitian selanjutnya diharapkan menggunakan *database* sebagai penyimpanan datanya.

3. Penelitian selanjutnya diharapkan membuat visualisasi akhir hasil prakiraan yang didapat.

DAFTAR PUSTAKA

Abbas, F. N., Saadoon, I. M., Abdalrdha, Z. K., & Abud, E. N. (2020). Capable of gas sensor MQ-135 to monitor the air quality with arduino uno. *International Journal of Engineering Research and Technology*, *13*(10), 2955–2959. https://doi.org/10.37624/IJERT/13.10.2020.2955-2959

Aditya, F., Devianto, D., & Maiyastri, M. (2019). Peramalan Harga Emas Indonesia Menggunakan Metode Fuzzy Time Series Klasik. *Jurnal Matematika UNAND*, *8*(2), 45. https://doi.org/10.25077/jmu.8.2.45-52.2019

Alotaibi, S., Mehmood, R., Katib, I., Rana, O., & Albeshri, A. (2020). Sehaa: A big data analytics tool for healthcare symptoms and diseases detection using twitter, apache spark, and machine learning. *Applied Sciences (Switzerland)*, *10*(4). https://doi.org/10.3390/app10041398

Altinçöp, H., & Oktay, A. B. (2019). Air Pollution Forecasting with Random Forest Time Series Analysis. *2018 International Conference on Artificial Intelligence and Data Processing, IDAP 2018*, 8–12. https://doi.org/10.1109/IDAP.2018.8620768

Amazon Web services. (2017). What is Streaming Data? *Amazon Web Services (AWS)*. https://aws.amazon.com/streaming-data/?nc1=h_ls

Apache Spark. (2022). *Spark MLlib*. https://spark.apache.org/mllib/

Apache Spark Documentation. (2018). *Window Operations*. https://spark.apache.org/docs/2.1.2/streaming-programming-guide.html

Arafat, M., Khan, A., Bhushan, C., & Ravi, V. (n.d.). *Nowcasting the Financial Time Series with Streaming Data Analytics under Apache Spark*. 1–26.

Damanik, S. R. (2019). *Sistem Monitoring Kualitas Udara Pada Kamar Rumah Sakit Menggunakan Sensor Dht11, Mq135 Dan Arduino Uno Berbasis Android*. https://library.usu.ac.id

Das, T., Zhu, S., & Or, A. (2015). *New Visualizations for Understanding Apache Spark Streaming Applications*. https://www.databricks.com/blog/2015/07/08/new-visualizations-for-understanding-apache-spark-streaming-applications.html

Dewi, W. C., Raharjo, M., & Wahyuningsih, N. E. (2021). Literatur Review : Hubungan Antara Kualitas Udara Ruang Dengan Gangguan Kesehatan Pada Pekerja. *An-Nadaa: Jurnal Kesehatan Masyarakat*, *8*(1), 88. https://doi.org/10.31602/ann.v8i1.4815

Ebada, A. I., Elhenawy, I., Jeong, C. W., Nam, Y., Elbakry, H., & Abdelrazek, S. (2022). Applying apache spark on streaming big data for health status prediction. *Computers, Materials and Continua*, *70*(2), 3511–3527. https://doi.org/10.32604/cmc.2022.019458

Ed-Daoudy, A., & Maalmi, K. (2019). Real-time machine learning for early detection of heart disease using big data approach. *2019 International Conference on Wireless Technologies, Embedded and Intelligent Systems, WITS 2019*, 1–5. https://doi.org/10.1109/WITS.2019.8723839

Firdaus, I. (2019). *Mengenal Time Series Forecasting*. Fasilkom UI. https://medium.com/

Hardika, D., & Nurfiana, N. (2019). Sistem Monitoring Asap Rokok Menggunakan Smartphone Berbasis Internet of Things (Iot). *Explore: Jurnal Sistem*

Informasi Dan Telematika, *10*(1). https://doi.org/10.36448/jsit.v10i1.1221

Hassan, F., Shaheen, M. E., & Sahal, R. (2020). Real-time healthcare monitoring system using online machine learning and spark streaming. *International Journal of Advanced Computer Science and Applications*, *11*(9), 650–658. https://doi.org/10.14569/IJACSA.2020.0110977

Hastomo, W., Karno, A. S. B., Kalbuana, N., Nisfiani, E., & ETP, L. (2021). Optimasi Deep Learning untuk Prediksi Saham di Masa Pandemi Covid-19. *Jurnal Edukasi Dan Penelitian Informatika (JEPIN)*, *7*(2), 133. https://doi.org/10.26418/jp.v7i2.47411

Hazelcast. (2021). *Micro Batch Processing*. Hazelcast. https://hazelcast.com/glossary/micro-batch-processing/

influxdata. (2022). *Time Series Forecasting Methods | InfluxData*. InfluxData. https://www.influxdata.com/time-series-forecasting-methods/

Johansson, L. (2022). *Part 1: RabbitMQ for beginners - What is RabbitMQ?* CloudAMQP. https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html

Juarez, E. K., & Petersen, M. R. (2022). A Comparison of Machine Learning Methods to Forecast Tropospheric Ozone Levels in Delhi. *Atmosphere*, *13*(1), 1–26. https://doi.org/10.3390/atmos13010046

Junaedy, Sajiah, Azzahrah, Z., & Idaryani. (2022). Rancang Bangun Alat Kontroling Kadar Udara Bersih Dan Gas Berbahaya Co, Co2 Dalam Ruangan Berbasis Mikrokontroler. *Jurnal Teknologi Dan Komputer (JTEK)*, *2*(02), 216–222. https://doi.org/10.56923/jtek.v2i02.104

Lazovic, I., Stevanovic, Z., Jovasevic-Stojanovic, M., Zivkovic, M., & Banjac, M. (2016). Impact of CO2 concentration on indoor air quality and correlation with relative humidity and indoor air temperature in school buildings in Serbia. *Thermal Science*, *20*(suppl. 1), 297–307. https://doi.org/10.2298/tsci1508311731

Lei, T. M. T., Siu, S. W. I., Monjardino, J., Mendes, L., & Ferreira, F. (2022). Using Machine Learning Methods to Forecast Air Quality: A Case Study in Macao. *Atmosphere*, *13*(9), 1–14. https://doi.org/10.3390/atmos13091412

Levy, E. (2021). *Batch processing*. Upsolver. https://www.upsolver.com/blog/batch-stream-a-cheat-sheet

Maulid, R. (2022). Kriteria Jenis Teknik Analisis Data dalam Forecasting. *DQLab*. https://www.dqlab.id/kriteria-jenis-teknik-analisis-data-dalam-forecasting

Microsoft. (2023). *Troubleshooting with the Apache Spark user interface*. https://learn.microsoft.com/id-id/azure/databricks/clusters/debugging-spark-ui

MQTT.org. (2022). *FAQ MQTT*. MQTT. https://mqtt.org/faq/

Nair, L. R., Shetty, S. D., & Shetty, S. D. (2018). Applying spark based machine learning model on streaming big data for health status prediction. *Computers and Electrical Engineering*, *65*, 393–399. https://doi.org/10.1016/j.compeleceng.2017.03.009

Olimex. (2013). *Tecnical Data Mq-135 Gas Sensor*. Hanwei Electron. https://www.olimex.com/Products/Components/Sensors/Gas/SNS-MQ135/resources/SNS-MQ135.pdf

Omran, N. F., Abd-El Ghany, S. F., Saleh, H., & Nabil, A. (2021). Breast Cancer Identification from Patients' Tweet Streaming Using Machine Learning Solution on Spark. *Complexity*, *2021*. https://doi.org/10.1155/2021/6653508

Peraturan Menteri Kesehatan Republik Indonesia No 1077 tentang Pedoman Penyehatan Udara di Dalam Ruang, (2011).

Ryanto, A. M. (2017). *Analisis Kinerja Framework Big Data Pada Cluster Tervirtualisasi: Hadoop Mapreduce dan Apache Spark*. 13–14. http://digilib.unhas.ac.id/uploaded_files/temporary/DigitalCollection/MjA1M2JhOTdkNWMzNGE5NmU5YzRhZDZkYTA2ZGE0OGJmMTBlOWJjMA==.pdf

Sequeira, R., Sebastian, R., Bunde, Y., & Suryawanshi, U. (2015). Automated Control System for Air Pollution Detection in Industries. *International Journal of Students' Research in Technology & Management*, *3*(5), 355–357. https://doi.org/10.18510/ijsrtm.2015.353

Spark, A. (2020). *Apache SparkTM - Unified Analytics Engine for Big Data*. https://doi.org/10.1016/b978-0-12-819816-2.00024-1

SparkByExamples. (2022). *PySpark RDD Tutorial | Learn with Examples What is RDD (Resilient Distributed Dataset)? PySpark RDD Benefits In-Memory Processing*. SparkByExamples.Com. https://sparkbyexamples.com/pyspark-rdd/

Sung, W. T., & Hsiao, S. J. (2021). Building an indoor air quality monitoring system based on the architecture of the Internet of Things. *Eurasip Journal on Wireless Communications and Networking*, *2021*(1), 1–41.

https://doi.org/10.1186/s13638-021-02030-1

Tibco. (2021). *What is Data Streaming*. Tibco Software. https://www.tibco.com/reference-center/what-is-data-streaming

UGM. (2018). *Random Forest*. UGM. https://machinelearning.mipa.ugm.ac.id/

Wahyuni, R. E. (2021). Optimasi Prediksi Inflasi Dengan Neural Network Pada Tahap Windowing Adakah Pengaruh Perbedaan Window Size. *Technologia: Jurnal Ilmiah*, *12*(3), 176. https://doi.org/10.31602/tji.v12i3.5181

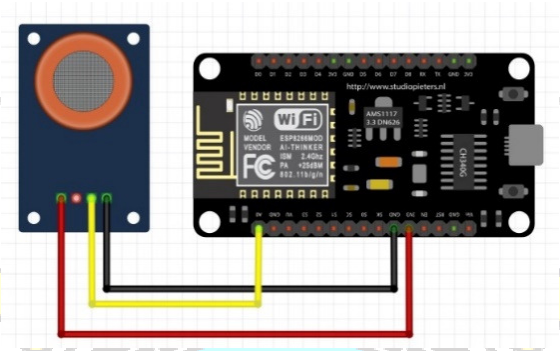
Wan, Z., Song, Y., & Cao, Z. (2019). Environment dynamic monitoring and remote control of greenhouse with ESP8266 NodeMCU. *Proceedings of 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference, ITNEC 2019*, *Itnec*, 377–382. https://doi.org/10.1109/ITNEC.2019.8729519

Wiranda, L., & Sadikin, M. (2019). Penerapan Long Short Term Memory Pada Data Time Series Untuk Memprediksi Penjualan Produk Pt. Metiska Farma. *Jurnal Nasional Pendidikan Teknik Informatika (JANAPATI)*, *8*(3), 184–196.

LAMPIRAN

Lampiran 1 Konfigurasi IoT

1. Rakit perangkat seperti gambar berikut



2. Kalibrasi sensor menggunakan script berikut

```
#include "MQ135.h"
const int ANALOGPIN=0;
MQ135 gasSensor = MQ135(ANALOGPIN);
void setup(){
Serial.begin(9600);
}
void loop(){
float rzero = gasSensor.getRZero();
Serial.println(rzero);
delay(1000);
}
```

3. Hitung kadar CO_2 menggunakan script berikut

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "MQ135.h"

const int ANALOGPIN=0;
MQ135 gasSensor = MQ135(A0);

// Update these with values suitable for your
network.

const char* ssid = "WLC_CNAP";
const char* password = "s4y4b1s4";
const char* mqtt_server = "10.2.3.74";

WiFiClient espClient;
```

```
PubSubClient client(espClient);
unsigned long lastMsg = 0;
String ppm;

void setup_wifi() {

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned
int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  // Switch on the LED if an 1 was received as first
character
  if ((char)payload[0] == '1') {
    digitalWrite(BUILTIN_LED, LOW);   // Turn the
LED on (Note that LOW is the voltage level
    // but actually the LED is on; this is because
    // it is active low on the ESP-01)
  } else {
    digitalWrite(BUILTIN_LED, HIGH);  // Turn the
LED off by making the voltage HIGH
  }

}

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
```

```
            if (client.connect("ESP", "arni", "12345")) {
              Serial.println("connected");
            } else {
              Serial.print("failed, rc=");
              Serial.print(client.state());
              Serial.println(" try again in 5 seconds");
              // Wait 5 seconds before retrying
              delay(5000);
            }
          }
        }

        void setup() {
          pinMode(BUILTIN_LED, OUTPUT);     // Initialize
        the BUILTIN_LED pin as an output
          Serial.begin(115200);
          setup_wifi();
          client.setServer(mqtt_server, 1883);
          client.setCallback(callback);
        }

        void loop() {

          if (!client.connected()) {
            reconnect();
          }
          client.loop();

          unsigned long now = millis();
          if (now - lastMsg > 60000) {
            lastMsg = now;
            ppm = gasSensor.getPPM();
            Serial.println(ppm);
            client.publish("sensor", ppm.c_str());
          }
}
```

Lampiran 2 Script Pengambilan Data Historis CO_2

```
import csv
import json
from datetime import datetime
import paho.mqtt.client as mqtt
import uuid

# The callback for when the client receives a CONNACK
response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    # Subscribing in on_connect() means that if we lose the
connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe("sensor")

# The callback for when a PUBLISH message is received from
the server.
def on_message(client, userdata, msg):
    dt      = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    dtfile  = datetime.now().strftime("%Y%m%d%H%M%S")
    co2     = json.loads(msg.payload.decode())
    data    = [dt, co2]
    rows.append(data)
    fname =
'/home/arni/stream/test/{}_{}.csv'.format(str(dtfile),str(u
uid.uuid1()))
    with open(fname, 'w') as f:
        # using csv.writer method from CSV package
        write = csv.writer(f)

        write.writerow(fields)
        write.writerows(rows)
        print("OK")

rows = []
fields = ['datetime', 'co2']
client = mqtt.Client()
client.username_pw_set("arni", "12345")
client.on_connect = on_connect
client.on_message = on_message

client.connect("10.2.3.74", 1883, 60)

# Blocking call that processes network traffic, dispatches
callbacks and
# handles reconnecting.
# Other loop*() functions are available that give a
threaded interface and a
# manual interface.
client.loop_forever()
```

Lampiran 3 Script Pengambilan Data CO_2 Secara Stream pada MQTT Broker

```
#untuk menerima data dari sensor

import paho.mqtt.client as mqtt
import socket
import json
from datetime import datetime

#membuat koneksi MQTT
class MyMQTTClass(mqtt.Client):

    def on_connect(self, mqttc, obj, flags, rc):
        print("rc: "+str(rc))

    def on_connect_fail(self, mqttc, obj):
        print("Connect failed")

#mengatur data masuk untuk dioper melalui socket.
Tiap ada data yang masuk, akan dikirim melalui
socket
    def on_message(self, mqttc, obj, msg):
        dt   = datetime.now().strftime("%Y-%m-%d
%H:%M:%S")
        y = {"datetime":dt, "co2":
float(msg.payload.decode())}
        z["data"].append(y)
        if len(z["data"]) == 13:
            print(z)
            conn, address = client_socket.accept()
            conn.send(json.dumps(z).encode())
            conn.close()
            z["data"].pop(0)

    def on_publish(self, mqttc, obj, mid):
        print("mid: "+str(mid))

    def on_subscribe(self, mqttc, obj, mid,
granted_qos):
        print("Subscribed: "+str(mid)+"
"+str(granted_qos))

    def on_log(self, mqttc, obj, level, string):
        print(string)

    def run(self):
        global conn
        self.username_pw_set("arni", "12345")
        self.connect("10.2.3.74", 1883)
        self.subscribe("sensor", 0)

        rc = 0
        while rc == 0:
            rc = self.loop()
        return rc
```

```
rows = '{"data": []}'
z = json.loads(rows)

HOST = "10.2.3.74"  # Standard loopback interface
address (localhost)
PORT = 65432  # Port to listen on (non-privileged
ports are > 1023)

client_socket = socket.socket()
client_socket.bind((HOST, PORT))
client_socket.listen()

mqttc = MyMQTTClass()
rc = mqttc.run()

print("rc: "+str(rc))
```

POLITEKNIK NEGERI UJUNG PANDANG

Lampiran 4 Script Prakiraan CO_2

```
#import library
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.streaming import StreamingContext
import sys
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.regression import LabeledPoint
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline
from pyspark.ml.regression import
RandomForestRegressor
from pyspark.ml.evaluation import
RegressionEvaluator
import numpy as np
import pandas as pd
import json

#membuat koneksi ke spark
sc = SparkContext("spark://10.2.3.74:7077","Stream")
spark = SparkSession(sc)
ssc = StreamingContext(sc, 1)

#membuat fungsi untuk training
def train(train_spark):
    print('\n\nDefining the pipeline
stages.................\n')

    #mengubah dataframe menjadi vektor
  stage_1 = VectorAssembler(inputCols=['0', '1',
'2','3','4', '5', '6','7', '8', '9', '10', '11',
'12'], outputCol="features")

    #mendefenisikan model
    model =
RandomForestRegressor(featuresCol='features',
labelCol='y')

    #membuat pipeline
    print('\n\nStages
Defined.........................\n')
    pipeline = Pipeline(stages= [stage_1, model])

    #mulai training model
    print('\n\nFit the pipeline with the training
data.......\n')
    pipelineFit = pipeline.fit(train_spark)

    #mengembalikan hasil training yaitu model
    print('\n\nModel Trained....Waiting for the
Data!!!!!!!!\n')
    return pipelineFit

#membuat fungsi untuk prediksi(testing)
def predict(test_spark):
```

```
        #mengubah data RDD  menjadi dataframe
        if not test_spark.isEmpty():
            df = spark.read.csv(test_spark,
    inferSchema=True, header=True)
            df.show()
            df = df.toPandas()
            df = df.set_index('datetime')
            dataset = df

            # get the specific IoT data
            df_sliding =
    dataset['co2'].astype(float).values

            #preparing the data, split it into X and y
    using sliding window method
            #the total number of feature of X is depend
    on the n_steps.
            #the y is depend on the ph or prediction
    horizon.
            X,y=sliding_window(df_sliding, window_size,
    ph)

            #mengubah X dan y menjadi dataframe (pandas)
            test_data = pd.DataFrame(X)
            test_data['y'] = pd.DataFrame(y)

            #mengubah train data menjadi dataframe
    (spark)
            test_spark =
    spark.createDataFrame(test_data)
            test_spark.show(5)

            #print(model)
            prediction = model.transform(test_spark)
            prediksi = prediction.toPandas()
            prediksi.to_csv("prediksi.csv", index=False)

    prediction.select('features','prediction','y').show(
    )
        evaluator = RegressionEvaluator(labelCol="y",
    predictionCol="prediction", metricName="rmse")
            rmse = evaluator.evaluate(prediction)
            print("Root Mean Squared Error (RMSE) on
    test data = %g" % rmse)

    #predict data dari sensor
    def val(test_spark):
        #mengubah data RDD json menjadi dataframe
        if not test_spark.isEmpty():
            data = test_spark.collect()
            data = json.loads(data[0])
            print(data["data"])
            df = pd.json_normalize(data["data"])
            print(df)
            df_slide =
    pd.DataFrame([df["co2"].to_list()])
```

```
            test_spark = spark.createDataFrame(df_slide)

            prediction = model.transform(test_spark)

    prediction.select('features','prediction').show()

    def sliding_window(data, window_size, ph):
    #create the matrix X and vector y. X will be used as
    input features while y is the output
        X, y = list(), list()
        for i in range(len(data)):
            # find the end of this pattern
            end_ix = i + window_size
            # check if we are beyond the sequence
            if end_ix > len(data) - ph:
                break
            seq_x, seq_y = data[i:end_ix], data[end_ix +
    ph - 1]
            #append it by creating matrix X and vector y
            X.append(seq_x)
            y.append(seq_y)
        return np.array(X), np.array(y)

    #membuat fungsi untuk mengubah data
    def parse(lp):
        #mengubah rdd menjadi dataframe
        if not lp.isEmpty():
            global model
            df = spark.read.csv(lp, inferSchema=True,
    header=True)
            df.show()
            #mengubah dataframe menjadi pandas
            df = df.toPandas()
            df = df.set_index('datetime')
            dataset = df

            df_sliding =
    dataset['co2'].astype(float).values

            #preparing the data, split it into X and y
    using sliding window method
            #the total number of feature of X is depend
    on the n_steps.
            #the y is depend on the ph or prediction
    horizon.
            X,y=sliding_window(df_sliding, window_size,
    ph)

            #mengubah X dan y menjadi dataframe (pandas)
            train_data = pd.DataFrame(X)
            train_data['y'] = pd.DataFrame(y)

            #mengubah train data menjadi dataframe
    (spark)
            train_spark =
    spark.createDataFrame(train_data)
```

```
        train_spark.show(5)

    #memanggil fungsi train dan menyimpan model
yang telah ditraining ke dalam variabel model
        model = train(train_spark)

#n_steps = total number of previous data that will
be used for learning. Or number of feature
# menggunakan data 1 hari (24 jam) sebelumnya
sebagai features
window_size = 13

#ph or prediction horizon = how many step ahead that
we want to predict?
# untuk memprediksi 6 jam ke depan
ph=1

#stream file
trainingData =
ssc.textFileStream("/home/arni/stream/train/")
prediksiData = ssc.socketTextStream('10.2.3.74',
65432)
testData =
ssc.textFileStream("/home/arni/stream/test/")

#hasil bacaan stream file dalam bentuk teks, diubah
kedalam RDD
trainingData.foreachRDD( lambda rdd: parse(rdd) )
testData.foreachRDD( lambda rdd: predict(rdd) )
prediksiData.foreachRDD( lambda rdd: val(rdd) )

# testData.pprint()

ssc.start()
ssc.awaitTermination()
```

Lampiran 5 Dokumentasi Pengambilan Data

Pengambilan Data CO_2 Di Kadar Normal

Pengambilan Data CO_2 Di Atas Kadar Normal