

**PREDIKSI CUACA BERDASARKAN KORELASI MULTI-
PARAMETER FENOMENA FISIS ALAM DENGAN
*MACHINE LEARNING***

SKRIPSI

Oleh :

**Ragil Bagus Agung Budiyono
175090307111003**



**JURUSAN FISIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN
ALAM
UNIVERSITAS BRAWIJAYA
MALANG
2021**

**PREDIKSI CUACA BERDASARKAN KORELASI MULTI-
PARAMETER FENOMENA FISIS ALAM DENGAN
*MACHINE LEARNING***

SKRIPSI

Sebagai Salah Satu Syarat Untuk Meraih Gelar
Sarjana Sains Bidang Fisika

Oleh :

Ragil Bagus Agung Budiyono

175090307111003



**JURUSAN FISIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN
ALAM
UNIVERSITAS BRAWIJAYA
MALANG
2021**

LEMBAR PENGESAHAN SKRIPSI

PREDIKSI CUACA BERDASARKAN KORELASI MULTI- PARAMETER FENOMENA FISIS ALAM DENGAN MACHINE LEARNING

Oleh:

Ragil Bagus Agung Budiyo

175090307111003

Setelah dipertahankan di depan Majelis Penguji
pada tanggal

Dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana Sains dalam bidang Fisika

Pembimbing I

Pembimbing II

Agus Naba, S.Si., MT., Ph.D

NIP. 197208061995121001

Dr.rer.nat. Abdurrouf, S.Si.,M.Si

NIP. 197209031994121001

Mengetahui,

Ketua Jurusan Fisika

FMIPA Universitas Brawijaya

Prof. Dr. rer. nat. Muhammad Nurhuda

NIP. 196400910199021001

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini:

Nama : Ragil Bagus Agung Budiyo

NIM : 175090307111003

Jurusan : Fisika

Penulis skripsi berjudul :

PREDIKSI CUACA BERDASARKAN KORELASI MULTI-PARAMETER FENOMENA FISIS ALAM DENGAN MACHINE LEARNING

Dengan ini menyatakan bahwa:

- 1. Isi dari skripsi yang saya tulis dan buat adalah benar karya saya sendiri dan tidak menjiplak karya orang lain.**
- 2. Apabila ditemukan ternyata skripsi yang saya tulis terbukti hasil menjiplak, maka saya akan bersedia menanggung segala risiko yang akan saya terima.**

Demikian pertanyaan ini dibuat dengan penuh kesadaran.

Malang, 1 April 2021

Yang menyatakan,

Ragil Bagus Agung Budiyo

175090307111003

ABSTRAK

Prediksi kondisi cuaca mampu memberikan informasi yang sangat besar untuk berbagai sektor seperti pertanian, perairan bahkan sampai pengendalian pandemi, menjadikan penelitian seputar prediksi kondisi cuaca semakin penting dilakukan. Pada penelitian ini, penulis menggunakan dua model *Machine Learning* yaitu regresi dengan *Recurrent Neural Network – Long Short-Term Memory* (RNN-LSTM) untuk menghasilkan prediksi tiap parameter dengan 2 hidden layer terdiri dari 32 dan 64 neuron, kemudian menggunakan hasil prediksi tiap parameter tersebut untuk memprediksi kondisi cuaca dengan *K – Nearest Neighbours Classification* sehingga prediksi kondisi cuaca berdasarkan korelasi tiap parameter didapatkan. Evaluasi dilakukan dengan melihat *Loss Function* yaitu *Root Mean Square Error* (RMSE) pada model RNN-LSTM dan perbandingan hasil dengan data asli (akurasi) pada *K – Nearest Neighbours*, *Optimizer* yang digunakan adalah ‘adam’ dengan opsi *learning rate* ‘default’. *Activation function* yang digunakan adalah relu. *Batch* disesuaikan dengan jumlah data yang diolah. Hasil yang didapatkan adalah regresi parameter dengan beberapa hasil yang tidak baik ditandai tingginya RMSE, dan prediksi kondisi cuaca dengan melibatkan seluruh parameter atau parameter terbaik saja yang meningkatkan akurasi prediksi.

Kata Kunci: Prediksi, Kondisi Cuaca, Parameter Fisis Cuaca, Recurrent Neural Network, Long-Short Term Memory, *K – Nearest Neighbours*,

ABSTRACT

Prediction for weather conditions is able to provide enormous information for various sectors such as agriculture, irrigation and even pandemic control, making research around predicting weather conditions increasingly important. In this study, authors used two Machine Learning models, using regression with Recurrent Neural Network - Long Short-Term Memory (RNN-LSTM) to produce predictions for each parameter with 2 hidden layers consisting of 32 and 64 neurons, then use the prediction results of each parameter to predict weather conditions with the K - Nearest Neighbors Classification so that the prediction of weather conditions based on the correlation of each parameter is obtained. The evaluation is done by looking at the Loss Function, namely the Root Mean Square Error (RMSE) in the RNN-LSTM model and the comparison of the results with the original data (accuracy) on K - Nearest Neighbors. The optimizer used is 'adam' with the learning rate option 'default'. The activation function used is relu. The batch is adjusted to the amount of data processed. Results obtained are parameter done with regression with some unfavorable results marked by high RMSE, and prediction of weather conditions involving all parameters or chosen parameters that increase the accuracy of the prediction.

Keywords: Prediction, Weather Conditions, Physical Weather Parameters, Recurrent Neural Networks, Long-Short Term Memory, K - Nearest Neighbors,

KATA PENGANTAR

Puji syukur kepada Allah SWT, karena atas segala limpahan rahmat, rezeki serta nikmat-Nya penulis dapat menyelesaikan skripsi dengan judul “**PREDIKSI CUACA BERDASARKAN KORELASI MULTI-PARAMETER FENOMENA FISIS ALAM DENGAN MACHINE LEARNING**”.

Dengan selesainya karya tulis ini, penulis ingin memberikan rasa terima kasih kepada pihak yang telah memberikan bantuan, dukungan, bimbingan serta saran sampai penulis dapat menyelesaikan proposal ini, khususnya kepada:

1. Allah SWT atas nikmat, rahmat, dan rezeki sehingga penulisan Proposal Tugas Akhir dapat diselesaikan dengan baik.
2. Kedua orang tua yang selalu memberikan dukungan baik moral maupun material.
3. Bapak Prof. Dr. rer.nat Muhammad Nurhuda. selaku Ketua Jurusan Fisika FMIPA Universitas Brawijaya Malang serta dosen pembimbing akademik.
4. Ibu Dr. Eng. Masruroh, M.Si. selaku ketua prodi Fisika.
5. Bapak Agus Naba, S.Si., MT., Ph.D selaku dosen Pembimbing I.
6. Bapak Dr.rer.nat. Abdurrouf, S.Si., M.Si selaku dosen Pembimbing I.
7. Teman-teman Jurusan Fisika FMIPA Universitas Brawijaya Malang, sahabat dekat serta pihak yang tidak dapat disebutkan yang telah memberikan semangat, dorongan serta motivasi.

Penulis mengetahui bahwa karya selalu ada cela, sehingga dari pembuatan karya tulis ini apabila terdapat kritik, saran serta masukan dapat diberikan ke penulis untuk mengembangkan dan memperbaiki dari karya tulis ini maupun diri penulis sendiri.

Malang, 1 April 2021

Penulis

DAFTAR ISI

LEMBAR PENGESAHAN SKRIPSI	i
LEMBAR PERNYATAAN.....	iii
ABSTRAK	v
ABSTRACT.....	vii
KATA PENGANTAR	ix
DAFTAR ISI.....	xi
DAFTAR GAMBAR	xv
DAFTAR TABEL.....	xix
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	3
1.4 Batasan Masalah.....	3
1.5 Manfaat	3
BAB II TINJAUAN PUSTAKA	5
2.1 Cuaca.....	5
2.1.1 Parameter Cuaca	6
2.1.2 Fenomena Fisis Cuaca	6
2.2 Artificial Intelligence.....	8
2.2.1 Machine Learning.....	9
2.2.2 Supervised Learning	9
2.2.3 Regresi dan Klasifikasi	10
2.2.4 Neural Network	11

2.2.5	Time-Series Neural Network	13
2.2.6	K-Nearest Neighbours	14
2.2.7	Loss Function	15
2.3	Python	16
2.3.1	TensorFlow.....	16
2.3.2	Keras.....	16
2.3.3	Scikit-Learn	17
BAB III METODOLOGI.....		19
3.1	Waktu dan Tempat Pelaksanaan	19
3.2	Alat dan Bahan	19
3.3	Tahapan Penelitian	19
3.3.1	Persiapan Komputasi	20
3.3.2	Pengolahan Data	20
3.3.3	Pembuatan Model Neural Network dan K-Nearest Neighbours	21
3.3.4	Pelatihan Neural Network dan K-Nearest Neighbours	22
3.3.5	Pengujian Neural Network dan K-Nearest Neighbours	23
3.3.6	Evaluasi dan Visualisasi ML	24
BAB IV HASIL DAN PEMBAHASAN		25
4.1	Perbandingan Hasil Regresi Neural Network Tiap Parameter	25
4.2	Prediksi Cuaca Dengan Seluruh Parameter.....	34
4.3	Prediksi Cuaca Dengan Parameter Pilihan.....	35
4.4	Korelasi Parameter Pada Model dan Fenomena Fisis	39
BAB V PENUTUP.....		45

5.1	Kesimpulan.....	45
5.2	Saran.....	46
DAFTAR PUSTAKA		47
LAMPIRAN A DATA HASIL PENELITIAN		50
LAMPIRAN B KODE PROGRAM.....		59

DAFTAR GAMBAR

Gambar 2. 1. Gambar yang merujuk deep learning masuk cakupan machine learning, dan machine learning masuk cakupan AI.....	8
Gambar 2. 2. Alur pengerjaan <i>supervised learning</i>	10
Gambar 2. 3. Plot kiri merupakan algoritma klasifikasi dan kanan adalah regresi.	11
Gambar 2. 4. Gambaran Neural Network.	12
Gambar 2. 5. Skema LSTM dan unit memorinya.....	14
Gambar 2. 6. K-Nearest Neighbours untuk data banyaknya gaji bulanan tiap pekerja.	15
Gambar 3. 1. Arsitektur LSTM	22
Gambar 4. 1. Grafik RMSE tiap parameter dengan jangkauan waktu berbeda	28
Gambar 4. 2. Perbandingan hasil uji dan latih pada parameter presipitasi (hasil regresi Desember 2010).....	29
Gambar 4. 3. Perbandingan hasil uji dan latih pada parameter temperatur (hasil regresi Desember 2010).....	30
Gambar 4. 4. Perbandingan hasil uji dan latih pada parameter indeks panas (hasil regresi Desember 2010).....	30
Gambar 4. 5. Perbandingan hasil uji dan latih pada parameter kecepatan angin (hasil regresi Desember 2010)	31
Gambar 4. 6. Perbandingan hasil uji dan latih pada parameter visibilitas (hasil regresi Desember 2010).....	31
Gambar 4. 7. Perbandingan hasil uji dan latih pada parameter kelembapan relatif (hasil regresi Desember 2010)	32
Gambar 4. 8. Perbandingan hasil uji dan latih pada parameter tutupan awan (hasil regresi Desember 2010).....	33
Gambar 4. 9. Perbandingan hasil uji dan latih pada parameter arah angin.....	33
Gambar 4. 10. Heat Map Correlation dari data set	40
Gambar 4. 11. Visualisasi data untuk korelasi parameter temperatur, tutupan awan dan presipitasi	41

Gambar 4. 12. Visualisasi data untuk korelasi parameter presipitasi, tutupan awan dan kelembapan relatif.	42
Gambar 4. 13. Visualisasi data untuk korelasi parameter temperatur, indeks panas dan kelembapan relatif.	42
Gambar 4. 14. Visualisasi data untuk korelasi parameter temperatur, arah angin dan kecepatan angin.	43
Gambar A. 1. Perbandingan hasil uji dan latih pada parameter tutupan awan (hasil regresi Juli - Desember 2012).....	50
Gambar A. 2. Perbandingan hasil uji dan latih pada parameter indeks panas (hasil regresi Juli - Desember 2012).....	50
Gambar A. 3. Perbandingan hasil uji dan latih pada parameter presipitasi (hasil regresi Juli - Desember 2012).....	51
Gambar A. 4. Perbandingan hasil uji dan latih pada parameter kelembapan relatif (hasil regresi Juli - Desember 2012).....	51
Gambar A. 5. Perbandingan hasil uji dan latih pada parameter temperatur (hasil regresi Juli - Desember 2012).....	52
Gambar A. 6. Perbandingan hasil uji dan latih pada parameter visibilitas (hasil regresi Juli - Desember 2012).....	52
Gambar A. 7. Perbandingan hasil uji dan latih pada parameter arah angin (hasil regresi Juli - Desember 2012)	53
Gambar A. 8. Perbandingan hasil uji dan latih pada parameter kecepatan angin (hasil regresi Juli - Desember 2012).....	53
Gambar A. 9. Perbandingan hasil uji dan latih pada parameter tutupan awan (hasil regresi Januari - Desember 2015).....	54
Gambar A. 10. Perbandingan hasil uji dan latih pada parameter indeks panas (hasil regresi Januari - Desember 2015)	54
Gambar A. 11. Perbandingan hasil uji dan latih pada parameter presipitasi (hasil regresi Januari - Desember 2015)	55
Gambar A. 12. Perbandingan hasil uji dan latih pada parameter kelembapan relatif (hasil regresi Januari - Desember 2015).....	55
Gambar A. 13. Perbandingan hasil uji dan latih pada parameter temperatur (hasil regresi Januari - Desember 2015)	56
Gambar A. 14. Perbandingan hasil uji dan latih pada parameter visibilitas (hasil regresi Januari - Desember 2015).....	56

Gambar A. 15. Perbandingan hasil uji dan latih pada parameter arah angin (hasil regresi Januari - Desember 2015)	57
Gambar A. 16. Perbandingan hasil uji dan latih pada parameter kecepatan angin (hasil regresi Januari - Desember 2015).....	57

DAFTAR TABEL

Tabel 3. 1. Tabel jangkauan waktu latih dan uji	23
Tabel 4. 1. Hasil tiap parameter dengan rasio bulan latih dan uji 11:1	25
Tabel 4. 2. Hasil tiap parameter dengan rasio bulan latih dan uji 30:6	26
Tabel 4. 3. Hasil tiap parameter dengan rasio bulan latih dan uji 60:12	26
Tabel 4. 4. Tabel hasil akurasi dengan seluruh parameter	34
Tabel 4. 5. Tabel akurasi dengan berbagai penggunaan parameter pada rasio latih dan uji bulanan 11:1	36
Tabel 4. 6. Tabel akurasi dengan berbagai penggunaan parameter pada rasio latih dan uji bulanan 30:6	36
Tabel 4. 7. Tabel akurasi dengan berbagai penggunaan parameter pada rasio latih dan uji bulanan 60:12.....	37
Tabel 4. 8. Tabel akurasi dengan perbandingan penggunaan parameter terbaik pada rasio latih uji 11:1	38
Tabel 4. 9. Tabel akurasi dengan perbandingan penggunaan parameter terbaik pada rasio latih uji 30:6	38
Tabel 4. 10. Tabel akurasi dengan perbandingan penggunaan parameter terbaik pada rasio latih uji 60:12.....	39

BAB I

PENDAHULUAN

1.1 Latar Belakang

Cuaca sangat mempengaruhi bagaimana manusia menjalankan kegiatannya, menjadikan cuaca sebagai faktor yang harus dipertimbangkan dalam berbagai sektor. Pada pertanian, cuaca mampu mempengaruhi bagaimana tiap tanaman dapat bertumbuh dengan optimal, Parameter dari cuaca sangat berpengaruh untuk pertumbuhan tanaman seperti kelembapan, temperatur dan lainnya (Cogato et al., 2019). Cuaca juga dapat mempengaruhi bagaimana sektor perairan, bukan hanya pertanian, cuaca mempengaruhi pasokan air (Kirono et al., 2016). Bahkan cuaca juga mempengaruhi bagaimana sebuah pandemi dikendalikan, dengan melihat parameter cuaca maka transmisi dari pandemi dapat diprediksi dan memberikan tindak lanjut yang tepat (Tosepu et al., 2020).

Pentingnya cuaca dalam kehidupan manusia yang melibatkan berbagai sektor mendorong manusia untuk melakukan peramalan dan prediksi cuaca, sehingga dampak dari cuaca dapat diantisipasi bahkan sebelum cuaca tersebut terjadi. Prediksi cuaca umumnya melibatkan satu parameter yang nantinya akan dikorelasikan dengan dampak dari parameter tersebut. Seperti yang ditulis oleh Khosravi dan kawan-kawan, prediksi kecepatan angin dilakukan untuk melihat dampak kecepatan angin ke *wind farm* atau pembangkit listrik tenaga angin (Khosravi et al., 2018). Prediksi dengan satu parameter juga telah dilakukan oleh banyak peneliti, seperti prediksi curah hujan dan temperatur dengan fungsi bergantung pada tujuan (Kirono et al., 2016; Volokitin et al., 2016).

Peramalan dengan satu parameter dapat dilakukan dengan tujuan spesifik, karena cuaca secara umum merupakan masalah multi parameter sehingga banyak hal yang dapat mempengaruhinya. Tanpa disertai dengan peramalan, terdapat penelitian yang membahas tentang klasifikasi cuaca berdasarkan

data yang ada. Terdapat penelitian yang memberikan klasifikasi cuaca berdasarkan foto yang diambil pada saat itu dan algoritma yang dijalankan akan mengidentifikasi tergolong apakah cuaca pada hari itu (Zhang et al., 2016). Penggolongan cuaca memang bermanfaat dalam beberapa bidang, seperti penggunaannya untuk mengidentifikasi cuaca dari kamera sehingga tidak diperlukannya manusia untuk mengetahui cuaca secara manual, tetapi hanya menggolongkan cuaca tidak dapat memberikan prediksi cuaca kedepannya.

Berdasarkan latar belakang tersebut, penulis akan membangun sebuah algoritma *time-series machine learning* sehingga prediksi tiap parameter cuaca akan dilakukan dan hasil prediksi tersebut digolongkan menggunakan algoritma *classification machine learning*, sehingga didapatkan ramalan cuaca multi parameter. Dalam penelitian kedepannya juga akan diteliti terkait hubungan antar parameter fisis yang paling mempengaruhi sehingga tercipta kategori cuaca pada waktu tersebut.

1.2 Rumusan Masalah

Dari latar belakang tersebut, maka diperoleh beberapa rumusan masalah sebagai berikut:

1. Bagaimana korelasi antara parameter fisis (Suhu minimum-maksimum, kecepatan dan arah angin, presipitasi dan kelembapan) untuk meramalkan cuaca?
2. Bagaimana cara memprediksi cuaca pada rentang jam dengan durasi bulanan, enam bulanan dan tahunan menggunakan multi- parameter?
3. Bagaimana keakuratan model yang dibuat dan korelasinya terhadap masing-masing parameter fisis apabila parameter ditambahkan atau dikurangi?

4. Bagaimana kemampuan model untuk memprediksi cuaca dengan berbagai rentang waktu dan parameter yang digunakan?

1.3 Tujuan

Dari latar belakang dan rumusan masalah tersebut, maka diperoleh tujuan penelitian sebagai berikut:

1. Mengetahui korelasi antara parameter fenomena fisis terhadap cuaca yang terjadi.
2. Mengetahui cara untuk memprediksi cuaca terhadap multi-parameter fenomena fisis dengan metode regresi dan klasifikasi.
3. Mengetahui kemampuan model terhadap korelasi tiap parameter dan perubahan hasil apabila parameter ditambahkan atau dikurangi.
4. Mengetahui kemampuan model untuk memprediksi pada rentang waktu dan parameter terlibat yang ditentukan.

1.4 Batasan Masalah

Batasan masalah yang diberikan pada penelitian yang akan dilakukan adalah sebagai berikut:

1. Cuaca yang diprediksi memiliki kriteria cerah, Sebagian mendung, mendung, hujan, cerah disertai hujan, sebagian mendung disertai hujan
2. Cuaca yang diprediksi berdasarkan data curah hujan tiap jam di kota Malang pada Januari 2010 – Januari 2021
3. Parameter faktor penentu hujan secara fisis yang digunakan adalah temperatur, indeks panas, kecepatan dan arah angin, presipitasi, visibilitas, tutupan awan dan kelembapan.

1.5 Manfaat

Penelitian ini memiliki beberapa manfaat yaitu:

1. Bagi Instansi terkait: Hasil model penelitian dapat digunakan sebagai alat prediksi cuaca pertama sebelum menggunakan alat dengan akurasi tinggi
2. Bagi Peneliti dibidang Iklim dan Cuaca: Hasil model penelitian dapat digunakan untuk membuat dan menganalisis prediksi cuaca pada skala bulanan, enam bulanan dan tahunan dengan ketelitian per jam

BAB II

TINJAUAN PUSTAKA

2.1 Cuaca

Cuaca memiliki makna yang bercampur dengan iklim, sehingga perbedaan arti dari istilah tersebut menjadi kabur. Padahal kedua istilah tersebut memiliki makna yang cukup berbeda apabila ditinjau dari segi waktu. Cuaca sendiri merupakan keadaan atmosfer pada tiap satuan waktu, sehingga terdapat rujukan waktu yang dituju. Penggunaan cuaca dengan konotasi waktu seperti cuaca hari ini, cuaca besok, cuaca jam 1 siang dan sebagainya pasti mengarah pada waktu tertentu untuk mengutarakan keadaan atmosfer.

Cuaca memiliki berbagai parameter yang diungkapkan seperti suhu, tekanan, angin, kelembapan yang diukur dengan alat khusus. Parameter tersebut akan mampu memberikan probabilitas keadaan yang terjadi, seperti akan turun hujan, mendung, cerah dan sebagainya. Fenomena fisis sangat berpengaruh pada keadaan cuaca, karena cuaca dibentuk dari berbagai faktor yang terjadi di atmosfer seperti rotasi bumi, kelembapan, suhu dikarenakan intensitas sinar matahari dan lain sebagainya (Wirjohamidjojo & Swarinoto, 2010).

Pentingnya cuaca juga selaras dengan pentingnya prediksi cuaca. Peramalan cuaca sulit dilakukan karena cuaca memiliki banyak faktor, bukan hanya parameter cuaca saja yang diperhatikan tapi bagaimana posisi bumi terhadap matahari dan bulan, juga pengaruh alam yang lain. Banyak metode prediksi yang dilakukan untuk meningkatkan upaya keberhasilan dalam prediksi itu sendiri, mulai dengan perhitungan secara manual dengan memperhitungkan parameter fisis, menggunakan regresi maupun menggunakan *machine learning* (Watts, 2014).

2.1.1 Parameter Cuaca

Cuaca memiliki parameter yang akan menggambarkan kondisi dari waktu tertentu, sehingga tiap parameter akan menggambarkan keadaan cuaca berdasarkan nilai dari parameter dan korelasi dari keseluruhan parameter. Parameter dari cuaca sendiri diukur dengan berbagai macam alat dan kondisi tertentu (Wirjohamidjojo & Swarinoto, 2010).

Beberapa parameter cuaca yang dapat dilihat adalah suhu minimum, suhu maksimum, keadaan angin, keadaan awan dan presipitasi, kondensasi air, dan evaporasi. Suhu minimum didapatkan dari pengukuran suhu pada malam hari dengan memperhitungkan keadaan berawan. Suhu minimum penting untuk menjaga tanaman tidak rusak karena keluar dari zona tumbuh. Suhu maksimum didapatkan dari keadaan cerah di siang hari, makna suhu maksimum sebenarnya adalah menandakan suhu maksimum dari udara. Keadaan angin didapatkan dari pengukuran kecepatan angin dan arah angin. Angin timbul akibat perubahan tekanan dan suhu pada suatu wilayah. Keadaan awan dan presipitasi dapat mendeskripsikan keadaan berawan atau hujan, dengan jumlah air di udara maupun di atmosfer diperhitungkan. Evaporasi merupakan parameter yang timbul akibat parameter lain, yaitu suhu juga luas permukaan maupun lokasi pengamatan, evaporasi dapat memberikan pandangan terkait siklus air yang nantinya akan membentuk awan dan sebagainya (Potter & Coleman, 2003).

2.1.2 Fenomena Fisis Cuaca

Bagaimana cuaca dan parameternya timbul tentu tidak akan jauh dari fisika karena cuaca merupakan produk dari fenomena fisika yang berada di alam. Mengerti bagaimana fenomena fisis terjadi di alam terkait dengan timbulnya nilai di parameter cuaca dapat menjelaskan mengenai parameter itu sendiri dan bisa didapatkannya metode untuk prediksi secara manual. Walau ternyata prediksi menggunakan cara manual ternyata

menghasilkan hasil yang kurang baik, memahami korelasi tiap parameter dapat meningkatkan hasil prediksi pada metode yang lain (Lions et al., 1992).

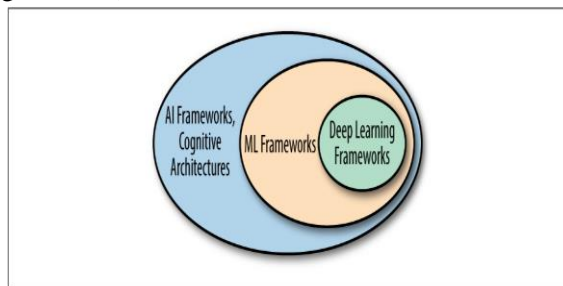
Fenomena fisis yang terjadi dari tiap parameter timbul dari intensitas radiasi matahari, kedekatan jarak bumi-matahari, perubahan kecepatan dan arah angin akibat perbedaan suhu dan tekanan, timbulnya hujan deras dengan berbagai jenis hujan dikarenakan bertambahnya uap air di awan yang turun disertai gesekan udara pada saat itu, bagaimana transfer energi yang terjadi dari sinar matahari ke bumi yang kemudian akan meningkatkan evaporasi, massa jenis uap air dengan masa jenis atmosfer saat itu dan lain sebagainya. Apabila semua faktor fisis dimasukkan kedalam prediksi maka akan timbul tingkat kekompleksan yang tinggi dan juga akan memakan sumber daya komputasi yang tinggi, sehingga faktor fisis biasanya disederhanakan menjadi beberapa bagian saja (Wirjohamidjojo & Swarinoto, 2010).

Variabel utama yang diambil untuk fenomena fisis adalah volume udara atau air di atmosfer, massa jenis, temperatur, dan tekanan. Dari variabel tersebut umumnya para peneliti cuaca (meteorologis) menggunakan beberapa model matematis untuk menjelaskan fenomena fisis tersebut, diantaranya persamaan konservasi momentum Navier-Stokes, hukum pertama termodinamika terkait konservasi energi, persamaan konservasi massa udara, persamaan kontinuitas massa uap air dan persamaan gas ideal. Persamaan tersebut dapat menjadi acuan korelasi antar parameter, tetapi korelasi antar parameter pastinya beragam bergantung pada faktor lain, dan hal tersebut menyebabkan prediksi menggunakan persamaan matematis memiliki hasil yang tidak optimal (Lions et al., 1992; Potter & Coleman, 2003).

2.2 Artificial Intelligence

Artificial Intelligence (AI) secara langsung berarti kecerdasan buatan, merujuk pada kemampuan komputer dalam menganalisis sebuah permasalahan dengan cerdas. AI sendiri tidak bermakna sempit merujuk pada sebuah robot cerdas, tapi sebuah algoritma yang mampu menemukan pola tertentu yang terdapat pada data. AI dapat diaplikasikan pada *e-commerce*, *facial recognition*, dan lainnya. AI memiliki beberapa cabang seperti *Machine Learning*, *Natural Language*, *Fuzzy Logic*, dan lain sebagainya (Lu et al., 2017).

Cakupan AI sangat luas, mencakup penyelesaian regresi, klasifikasi, *clustering* hingga ke penalaran, perencanaan dan navigasi. Ide besar dari AI adalah mampu menyelesaikan kebutuhan manusia yang mendasar dengan baik atau bahkan *lebih* baik. Walaupun usaha untuk mencapai hal tersebut sangat sulit karena otak manusia terdiri dari jutaan syaraf yang terhubung dan memiliki fungsionalitas masing-masing, AI merupakan terobosan maju di bidang teknologi. Cakupan AI dan *machine learning* maupun *deep learning* dapat dilihat pada Gambar 2. 1. Gambar yang merujuk deep learning masuk cakupan machine learning, dan machine learning masuk cakupan AI (Morgan, 2018).



Gambar 2. 1. Gambar yang merujuk deep learning masuk cakupan machine learning, dan machine learning masuk cakupan AI

2.2.1 Machine Learning

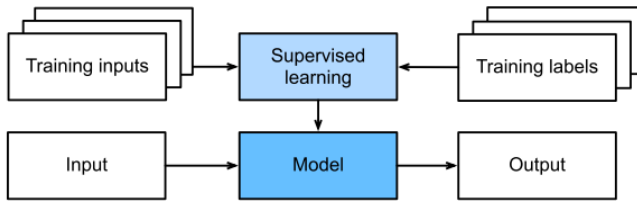
Machine Learning (ML) merupakan algoritma naungan AI. Fungsionalitas awalnya adalah mengubah data mentah menjadi sesuatu yang bernilai, hal tersebut berlaku pada ML secara konvensional. Dengan penelitian yang matang, ML pada terobosan terbarunya mampu benar-benar mengekstrak sebuah pola pada data mentah dalam berbagai jenis data. Pola yang diambil dari data mentah dapat berupa konstanta yang mempengaruhi tiap parameter, sehingga dari tiap data dapat ditemukan pola regresinya dan melengkapi atau meramalkan data yang belum ada, bahkan mampu mengklasifikasikan data pada pola tertentu bergantung dengan hasil yang didapatkan setelah proses pelatihan dilakukan.

ML merupakan pembahasan yang hangat karena kemampuan menemukan pola dari sebuah data mentah, sehingga proses *engineering* kode program menjadi lebih efisien, karena dengan memberikan algoritma tertentu kode program dapat beradaptasi dengan baik tanpa harus memberikan perlakuan khusus secara manual oleh pengguna. Terlebih pada era informasi saat ini, data merupakan hal yang mudah didapatkan dari berbagai sumber dan data adalah bahan utama agar ML bisa bekerja dengan baik (Lecun et al., 2015).

2.2.2 Supervised Learning

Supervised learning adalah sistem pembelajaran di ML dengan memberikan kategori label di tiap keluarannya. Pemberian kategori label yang dimaksud adalah hasil keluaran yang diharapkan sudah ditentukan, misal regresi dengan label curah hujan atau klasifikasi dengan label membedakan motor dan mobil. *Supervised learning* sangat umum digunakan karena penggunaannya yang sangat luas, berbeda dengan sistem lain seperti *unsupervised learning* yang tidak memberikan label pada

keluaran sehingga ML harus menentukan labelnya sendiri berdasarkan data (Lecun et al., 2015).



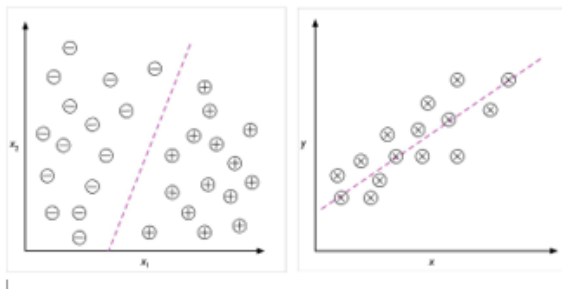
Gambar 2. 2. Alur pengerjaan *supervised learning*

Alur pengerjaan *supervised learning* dapat dilihat seperti pada **Error! Reference source not found.** Data berupa *training input* disiapkan sebagai bahan pelatih dari ML untuk menghasilkan model. *Training label* juga ditentukan untuk memberikan informasi ke ML keluaran yang seharusnya. Apabila keduanya telah disiapkan, maka data akan dilatih kemudian menghasilkan model yang tepat berdasarkan data masukan. Model yang dihasilkan dapat diuji maupun digunakan dengan *input* uji dan akan menghasilkan keluaran berdasarkan model yang telah dilatih sebelumnya (Czum, 2020).

2.2.3 Regresi dan Klasifikasi

Supervised learning dapat melakukan berbagai macam hal, pada umumnya pengguna akan menggunakan *supervised learning* sebagai sistem pembelajaran untuk menyelesaikan regresi dan klasifikasi. Sederhananya regresi merupakan prediksi yang dilakukan dengan angka real berdasarkan data yang ada. Regresi dapat menambahkan data yang kosong maupun meneruskan data yang sudah ada. Contoh regresi adalah prediksi harga rumah tahunan, prediksi harga saham, prediksi curah hujan. Kunci dari regresi adalah keluaran yang berupa nilai kontinu, sehingga sangat cocok untuk menyelesaikan permasalahan prediksi.

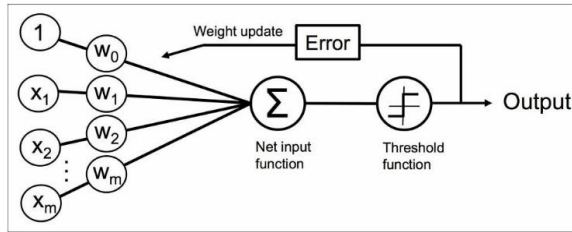
Lain halnya klasifikasi, ML akan mengelompokkan data sesuai dengan kategori label yang telah ditentukan (pada kasus *supervised learning*). Keluaran dari klasifikasi adalah nilai diskrit yang menentukan label dari data. Contohnya adalah bagaimana klasifikasi kucing dan anjing, kanker ganas dan tidak, serta klasifikasi keadaan cuaca (Czum, 2020). Perbedaan klasifikasi dan regresi dapat dibandingkan pada Gambar 2. 3. Plot kiri merupakan algoritma klasifikasi dan kanan adalah regresi.. Klasifikasi digunakan untuk mengelompokkan dan regresi untuk menemukan pola data (Raschka & Mirjalili, 2017).



Gambar 2. 3. Plot kiri merupakan algoritma klasifikasi dan kanan adalah regresi.

2.2.4 Neural Network

Algoritma ML yang cukup terkenal sebagai *powerhouse* untuk menyelesaikan banyak permasalahan adalah *Neural Network* (NN). NN dibangun berdasarkan penggambaran sistem syaraf manusia, ketika beragam *input* masuk lalu diproses untuk menghasilkan keluaran biner. NN akan mempelajari data yang diberikan lalu memberikan *update* bobot sehingga model akan semakin baik, maka dari itu NN merupakan algoritma yang dilakukan secara berulang (*epoch*) untuk menentukan bobot yang sesuai dengan label keluaran.



Gambar 2. 4. Gambaran Neural Network.

Skema NN seperti pada Gambar 2. 4. Gambaran Neural Network. menjelaskan terkait bagaimana *input* (x) yang diberikan akan diproses dengan bobot awal untuk membuat model berupa fungsi. Data kemudian diproses dengan *activation function* yang menjadikan hasil menjadi keluaran diskrit (sesuai dengan jenis *activation function*) lalu dievaluasi dengan fungsi *error* (atau *loss function*). Apabila sudah sesuai dengan batas maka training dihentikan, apabila belum maka hasil dari *error* tersebut digunakan untuk memberikan *update* ke bobot sehingga *epoch* selanjutnya akan menghasilkan *error* yang lebih kecil.

Model matematis sederhana untuk menggambarkan *update* bobot adalah sebagai berikut:

$$w_j := w_j + \Delta w_j \quad (2.1)$$

$$\Delta w_j = \eta (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)} \quad (2.2)$$

w_j menggambarkan bobot pada unit j , akan di-*update* setiap kali ada perubahan w_j yang ditandai dengan Δw_j . Δw_j adalah kalkulasi yang memperhitungkan η sebagai *learning rate* (ditentukan sendiri oleh pengguna) untuk menunjukkan besarnya langkah yang perlu diambil tiap bobot di-*update* (semakin kecil semakin presisi tetapi memakan sumber daya komputasi dan waktu), $y^{(i)}$ sebagai label keluaran atau target *output*, $\hat{y}^{(i)}$ adalah keluaran *training* dengan bobot sebelumnya, dan $x_j^{(i)}$ adalah input. Persamaan matematis tersebut dapat diartikan bahwa delta bobot (perubahan bobot) didapatkan dari selisih hasil keluaran

training dengan data sesungguhnya per data latih $x_j^{(i)}$, dengan kontrol step dikendalikan oleh *learning rate* (Raschka & Mirjalili, 2017).

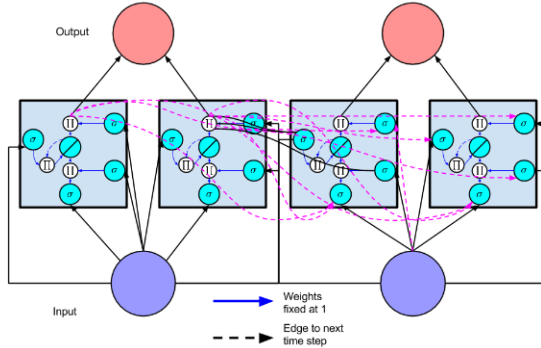
2.2.5 Time-Series Neural Network

NN punya kecenderungan untuk menemukan pola pada sebuah data masukan. Penemuan pola tersebut didasarkan pada *input* sebelumnya, sehingga data yang telah di proses akan mempengaruhi bobot ke depannya. Permasalahannya adalah NN konvensional cenderung “lupa” apabila data yang dipanggil terlalu lama, dan NN konvensional tidak sensitif terhadap waktu. Karena itu, NN konvensional kurang baik untuk mengelola data dengan waktu sensitif seperti ramalan curah hujan dan harga saham. Permasalahan tersebut dapat diselesaikan dengan beberapa metode, yaitu pembuatan *time-series neural network*. Banyak metode pembuatan *time-series* NN, seperti *recurrent NN*, dan yang terbaru adalah LSTM.

Recurrent NN sebenarnya adalah modifikasi dari *feed-forward Neural Network* dengan menambahkan komponen waktu. *Recurrent NN* mampu mengingat data yang telah lampau dengan urutan yang baik. Kelemahannya adalah *Recurrent NN* tidak dapat memilih konstanta mana yang harus diingat dan dilupakan, sehingga muncul metode baru yaitu *Long-Short Term Memory* (LSTM).

LSTM memiliki kelebihan yaitu adanya unit tambahan yang berfungsi untuk mengurutkan data sesuai dengan urutan kejadian waktu dan mengingatnya (seperti unit memori). Unit tersebut memiliki *gate* khusus dengan dimulai dari nilai *Input Node* (**g**), *Input Gate* (**i**), *Forget Gate* (**f**), *Output Gate* (**o**). *Gate* akan menentukan memori mana yang harus dipertahankan atau dilupakan, yang akan mempengaruhi perhitungan dari bobot. Kita lihat dari Gambar 2. 5. Skema LSTM dan unit memorinya ada 3 *gate* (lingkaran berwarna putih), dengan *gate* pertama yaitu

Input Gate untuk memberikan perubahan ke data input, lalu *Forget Gate* akan menentukan apakah ada konstanta yang harus dilupakan, apabila ada maka akan dihapus dan digantikan pada *Output Gate* (Lipton et al., 2015).



Gambar 2. 5. Skema LSTM dan unit memorinya

2.2.6 K-Nearest Neighbours

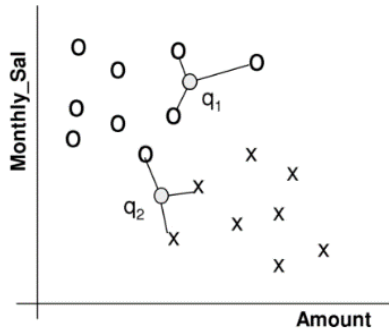
Metode klasifikasi memiliki banyak bentuk, *K-Nearest Neighbours* memanfaatkan data sekitarnya untuk mengklasifikasikan data yang ditunjuk. Apabila kita lihat pada Gambar 2. 6. K-Nearest Neighbours untuk data banyaknya gaji bulanan tiap pekerja., q_1 dan q_2 adalah data yang ingin diklasifikasikan. Data yang ingin diklasifikasikan tersebut diukur jaraknya dengan beberapa *tetangga terdekat* yang telah diketahui labelnya. Dari pengukuran jarak terhadap data dengan label yang diketahui, q_1 dan q_2 dapat diklasifikasikan (Cunningham & Delany, 2020).

Perhitungan jarak dengan tetangga terdekatnya dapat dituliskan dengan model matematis sebagai berikut:

$$d(q, x_i) = \sum_j w_j \delta(q_j, x_{if}) \quad (2.3)$$

$d(q, x_i)$ adalah perhitungan jarak dari data yang ingin diklasifikasikan q dan data yang telah diberikan label x . w_j

adalah bobot klasifikasi, $\delta(q_j, x_{if})$ adalah penentuan atribut klasifikasi antara q dan w (Cunningham & Delany, 2020).



Gambar 2. 6. K-Nearest Neighbours untuk data banyaknya gaji bulanan tiap pekerja.

2.2.7 Loss Function

Loss function merupakan fungsi untuk menentukan seberapa besar kesalahan yang didapat dari *training* maupun *fitting* ML. Pada regresi, umumnya menggunakan *root mean square error* (RMSE) atau menggunakan *mean absolute error* (MAE). Penggunaannya berdasarkan model matematis yang ditunjukkan pada persamaan (2. 4) untuk RMSE dan persamaan (2. 5) untuk MAE.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n e_i^2} \quad (2. 4)$$

$$MAE = \frac{1}{n} \sum_{i=1}^n |e_i| \quad (2. 5)$$

RMSE dan MAE digunakan berdasarkan kebutuhan, dan jenis data yang digunakan. Sebagian merujuk bahwa MAE memiliki keunggulan untuk menunjukkan *loss function* yang stabil sehingga dapat merepresentasikan data dengan benar, tetapi pada umumnya RMSE menjadi standar untuk dijadikan *loss function* (Chai & Draxler, 2014).

2.3 Python

Python merupakan bahasa pemrograman tingkat tinggi yang memiliki banyak *library* untuk mendukung penelitian. Bahasa tingkat tinggi berarti Python mudah dipahami oleh pengguna, karena cenderung lebih familier dengan bahasa manusia. Walaupun tergolong bahasa tingkat tinggi, Python merupakan salah satu bahasa yang efektif digunakan untuk melakukan penelitian numerik. Walaupun tidak se-efisien Fortran atau C++, Python memiliki kelebihan lain dengan banyaknya *library* yang mendukung banyak penelitian, seperti penggunaan NumPy, Pandas, TensorFlow, dan lain sebagainya (Dubois et al., 2007).

2.3.1 TensorFlow

TensorFlow adalah bagian dari *library* yang disediakan oleh python. TensorFlow banyak digunakan untuk melakukan penelitian terkait dengan ML, karena tersedia banyak *syntax* yang mempermudah penggunaan ML. TensorFlow dapat melakukan pekerjaan paralel untuk mengerjakan tugas komputasi pada algoritma ML ketika proses *training* dilaksanakan.

TensorFlow dapat meningkatkan kinerja dari sumber daya komputasi, karena kalkulasi yang dilakukan dapat juga memanfaatkan GPU yang telah di-*support*, walaupun penggunaan CPU secara konvensional masih bisa digunakan (Abadi, 2016).

2.3.2 Keras

Keras adalah *library* yang dimiliki oleh bahasa python. Keras dapat berjalan pada Theano ataupun TensorFlow. Keras mempermudah pengerjaan pembuatan model *deep learning* untuk pengembangan dan penelitian. Karena berjalan pada TensorFlow, Keras juga mampu memanfaatkan sumber daya

komputasi GPU, tidak hanya CPU, menjadikan Keras *library* pilihan untuk menjalankan model ML (Brownlee, 2019).

2.3.3 Scikit-Learn

Scikit-Learn adalah alternatif *library* yang dapat digunakan di Python untuk keperluan penelitian dan pengembangan ML. Scikit-Learn memiliki banyak *toolkit* untuk mempermudah penelitian terkait ML, dengan banyak metode yang bisa dipilih. Pada kasus klasifikasi, Scikit-Learn juga menyediakan *syntax* untuk menyelesaikan masalah klasifikasi dengan metode *K-Nearest Neighbor*s (Pedregosa et al., 2014).

BAB III

METODOLOGI

3.1 Waktu dan Tempat Pelaksanaan

Penelitian ini dilakukan pada bulan Maret 2021 sampai bulan Mei 2021 di Jalan Srigading Dalam No. 58K Kota Malang.

3.2 Alat dan Bahan

Alat yang digunakan dalam penelitian ini adalah sebagai berikut:

1. Sebuah Laptop Asus dengan Spesifikasi Processor Intel i7 770HQ CPU @ 2.8 GHz, 4 core(s), 8 Logical Processor(s), RAM DDR4 16.00 GB, SSD 512 GB, NVIDIA Geforce GTX 1050.
2. Sebuah *Operating System* (OS) Windows 10 Pro
3. Beberapa *library* Python yaitu NumPy, Pandas, Matplotlib, TensorFlow, Keras, Seaborn dan Scikit-Learn.
4. NVIDIA-CUDA untuk membuka akses penggunaan GPU pada *library* TensorFlow.
5. IDE (*Integrated Development Environment*) berupa Spyder dengan versi instalasi Python 3.8.3.
6. Data cuaca tiap jam yang didapatkan dari laman penyedia data cuaca dengan alamat: <https://www.visualcrossing.com/weather/weather-data-services#/>

3.3 Tahapan Penelitian

Pada penelitian ini terdapat beberapa tahapan dengan keterangan lengkap sebagai berikut:

1. Persiapan Komputasi
2. Pengolahan Data
3. Pembuatan *Model Neural Network* dan *K-Nearest Neighbours*

4. Pelatihan *Neural Network* dan *K-Nearest Neighbours*
5. Pengujian *Neural Network* dan *K-Nearest Neighbours*
6. Evaluasi dan Visualisasi ML

3.3.1 Persiapan Komputasi

Sebelum penelitian dimulai, berbagai perangkat lunak dipersiapkan dan diinstalasi terlebih dahulu. Pertama perangkat lunak IDE Spyder dengan instalasi Python 3.8.3. Perangkat lunak selanjutnya adalah instalasi NVIDIA-CUDA agar GPU pada perangkat keras bisa digunakan untuk menjalankan komputasi guna melancarkan dan mempercepat proses komputasi. Selanjutnya instalasi *library* Python dengan membuka *Windows Command Prompt* (CMD) dan mengetikkan perintah sebagai berikut:

1. Pip install NumPy
2. Pip install Pandas
3. Pip Install Matplotlib
4. Pip install TensorFlow
5. Pip install Keras
6. Pip install seaborn
7. Pip install scikit-learn

Setelah instalasi *library* Python selesai dilakukan, maka IDE Spyder dapat dijalankan dan pembuatan kode program dapat dimulai.

3.3.2 Pengolahan Data

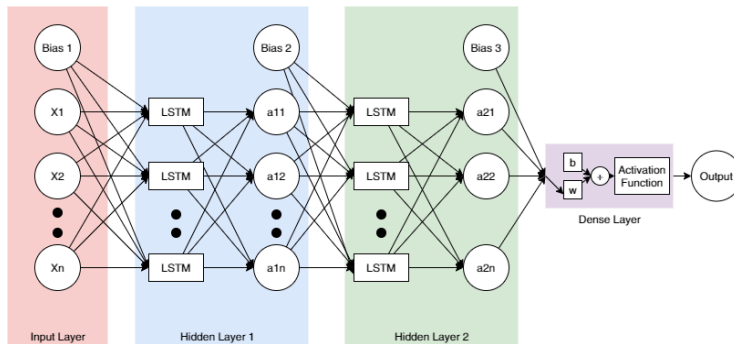
Data yang telah didapatkan dari laman penyedia data cuaca kemudian dibersihkan terlebih dahulu. Data yang didapatkan disaring untuk melihat apakah ada yang parameternya tidak memiliki nilai (NaN). Apabila ada, maka data tersebut dihapus, karena akan mempengaruhi kualitas dari model yang akan dijalankan.

Selanjutnya dari data yang didapatkan diekspansi untuk memenuhi syarat *time-series neural network*, yaitu dengan mengelompokkan data menjadi *time-step (lag)*. Misalkan *time-step* diatur selama 12 jam, maka akan ada 12 data uji kemudian data ke 13 menjadi label keluaran untuk dibandingkan hasilnya, lalu data bergeser dari 2-13 dengan data ke 14 menjadi label keluaran dan seterusnya. Pengolahan data ini bermaksud untuk memenuhi syarat *time series neural network*.

Setelah data tersebut selesai disiapkan, maka perlu diberikan skala agar mengurangi beban komputasi dan mengurangi *error*. Data dengan jangkauan (*range*) yang berbeda-beda diskalakan dengan ukuran 0-1, yang nantinya akan dikembalikan ke nilai sesungguhnya di akhir pengujian.

3.3.3 Pembuatan Model Neural Network dan K-Nearest Neighbours

Model NN yang dibuat memenuhi kriteria arsitektur regresi NN dengan metode LSTM. *Hyperparameter* akan disesuaikan tiap *training* dilaksanakan. Nilai awal yang di masukan sebelum penyesuaian adalah 1 *layer input*, 2 *hidden layer* LSTM, dan 1 *output layer*. Unit pada *hidden layer* pertama sebanyak 64, dan pada *hidden layer* kedua sebanyak 32. *optimizer* yang digunakan adalah 'adam' dengan opsi *learning rate* 'default'. *Loss function* yang digunakan adalah RMSE. *Activation function* yang digunakan adalah relu. Gambaran arsitektur layer pada model NN dapat dilihat pada Gambar 3. 1. Arsitektur LSTM dengan arah dari kiri ke kanan, dengan *input layer* berwarna biru, *hidden layer* dengan LSTM berwarna merah muda dan *output layer* berwarna hijau.



Gambar 3. 1. Arsitektur LSTM

Model pada *K-Nearest Neighbours* dapat merujuk pada persamaan (2. 3). *K-Nearest Neighbours* tidak memiliki arsitektur khusus seperti *Neural Network* karena tidak memiliki neuron.

3.3.4 Pelatihan Neural Network dan K-Nearest Neighbours

Setelah model selesai dibuat dan data telah dibersihkan, maka proses pelatihan dapat dimulai. Pelatihan dimulai terlebih dahulu pada *Neural Network*, dengan membagi data latih dan uji. Apabila merujuk pada Gambar 2. 5. Skema LSTM dan unit memorinya maka konstanta dari model akan didapatkan terlebih dahulu menggunakan data latih. Tiap *epoch* atau pengulangan yang dilakukan, akan ada *update* bobot dan bias sesuai dengan persamaan (2. 5). Dengan menentukan jenis *loss function* RMSE (sebagai acuan awal), maka data akan diolah terus hingga nilai RMSE mengecil sembari memberikan *update* ke bias dan bobot. Apabila *epoch* telah selesai dilakukan, maka pelatihan *Neural Network* selesai.

Pelatihan *K-Nearest Neighbours* hanya dengan memberikan data yang telah diolah dan membaginya seperti *Neural Network*, lalu model akan pelatihan akan dilakukan dengan memindai daerah dan memberikan klasifikasi berdasarkan jarak ke tetangga terdekat. Setelah pelatihan selesai dengan dari data set latih, maka pelatihan selesai.

3.3.5 Pengujian Neural Network dan K-Nearest Neighbours

Pelaksanaan pengujian dibagi menjadi dua bagian, yaitu pengujian hasil regresi dan klasifikasi. Pengujian dilakukan dengan menggunakan model yang telah dilatih dengan data set, kemudian model tersebut digunakan untuk melakukan regresi pada *Neural Network*, dan klasifikasi pada *K-Nearest Neighbours* dengan menggunakan data set uji yang telah dikhususkan untuk menjadi data uji.

Pengujian model menggunakan tiga jangkauan data set ditunjukkan pada Tabel 3. 1. Tabel jangkauan waktu latih dan uji Perbedaan jangkauan dapat menunjukkan performa model ketika menghadapi jumlah data yang berbeda-beda.

Tabel 3. 1. Tabel jangkauan waktu latih dan uji

No.	Panjang bulan (Latih)	Panjang bulan (Uji)
1.	Januari – November 2010	Desember 2010
2.	Januari – November 2011	Desember 2011
3.	Januari – November 2012	Desember 2012
4.	Januari – November 2013	Desember 2013
5.	Januari – November 2014	Desember 2014
6.	Januari – November 2015	Desember 2015
7.	Januari 2010 – Juni 2012	Juli – Desember 2012
8.	Januari 2013 – Juni 2015	Juli – Desember 2015
9.	Januari 2016 – Juni 2018	Juli – Desember 2018
10.	Januari 2010 – Desember 2014	Januari – Desember 2015
11.	Januari 2015 – Desember 2019	Januari – Desember 2020

Hasil akhir yang diharapkan setelah *Neural Network* selesai menjalani pengujian adalah data regresi tersebut akan

diklasifikasikan oleh *K-Nearest Neighbours*, kemudian akan dibandingkan dengan nilai sesungguhnya hingga tercipta prediksi cuaca. Pengujian didasarkan pada nilai RMSE untuk regresi dan nilai akurasi untuk klasifikasi, dengan acuan semakin kecil RMSE pada regresi maka hasil pengujian model semakin baik, sebaliknya semakin besar akurasi maka semakin baik hasil dari pengujian model klasifikasi.

3.3.6 Evaluasi dan Visualisasi ML

Setelah seluruh tahapan penelitian dilakukan, maka tahapan akhir adalah evaluasi dari model dan *hyperparameter* dari modelnya. Apabila hasil *loss function* masih terlalu tinggi (RMSE terlalu tinggi pada regresi, dan akurasi terlalu kecil pada klasifikasi), maka model dapat dimodifikasi sesuai kebutuhan. Setelah hasil didapatkan dari regresi selanjutnya dievaluasi tiap parameter yang memiliki nilai RMSE paling kecil dan dibandingkan antar parameter lainnya. Korelasi antar parameter diuji dengan cara meninggalkan tiap-tiap parameter dan menjalankan program klasifikasi. Korelasi parameter terbaik akan terlihat ketika nilai akurasi meningkat sehingga prediksi kondisi cuaca semakin akurat.

Visualisasi dapat menggunakan *library* Matplotlib dan Seaborn untuk melihat perbandingan *loss function* pada data *training* dan data uji sehingga dapat dievaluasi apakah model sudah *fit*, atau masih *undefit/overfit*. Selanjutnya visualisasi korelasi antara parameter fisis cuaca dibuat juga sebagai pembanding dari hasil prediksi tiap parameter yang digunakan.

BAB IV

HASIL DAN PEMBAHASAN

4.1 Perbandingan Hasil Regresi Neural Network Tiap Parameter

Perbandingan pertama berdasarkan hasil uji regresi dari tiap parameter cuaca. Pengujian dilakukan dengan jangkauan waktu yang telah ditentukan pada Tabel 4. 1. Hasil tiap parameter dengan rasio bulan latih dan uji 11:1, menunjukkan perbandingan hasil regresi pada tiap parameter dengan jangkauan waktu latih 11 bulan dan waktu uji 1 bulan, Tabel 4. 2. Hasil tiap parameter dengan rasio bulan latih dan uji 30:6 dengan jangkauan waktu latih 30 bulan dan waktu uji 6 bulan, Tabel 4. 3. Hasil tiap parameter dengan rasio bulan latih dan uji 60:12 dengan jangkauan waktu latih 60 bulan dan waktu uji 12 bulan. *Hyperparameter* yang dirubah dari tiap jangkauan waktu hanya *batch size* dengan nilai 30 untuk 11:1, 50 untuk 30:6 dan 100 untuk 60:12. Hasil yang diperoleh dan dibandingkan adalah nilai *Root Mean Square Error* (RMSE), dengan keterangan semakin kecil nilai RMSE maka semakin baik hasil dari model regresi yang dilakukan.

Tabel 4. 1. Hasil tiap parameter dengan rasio bulan latih dan uji 11:1

Parameter	RMSE					
	2010	2011	2012	2013	2014	2015
Temperatur	1.22	1.33	1.29	1.08	1.22	1.29
Indeks Panas	1.85	2.34	2.05	1.61	1.87	1.93
Presipitasi	0.49	0.91	0.55	0.48	0.34	0.57
Kecepatan Angin	6.13	4.34	4.26	5.27	4.95	4.38
Arah Angin	107.6 5	99.9 8	95.3 8	80.1 6	96.6 6	76.0 6
Visibilitas	1.72	11.9 5	1.89	21.1 0	1.64	1.31
Tutupan Awan	18.95	16.6 0	19.7 8	17.6 5	18.0 6	16.2 2

Kelembapan Relatif	6.39	6.63	6.69	5.76	6.11	6.44
--------------------	------	------	------	------	------	------

Tabel 4. 2. Hasil tiap parameter dengan rasio bulan latih dan uji 30:6

Parameter	RMSE		
	2012	2015	2018
Temperatur	1.06	0.99	1.28
Indeks Panas	1.50	1.39	1.85
Presipitasi	0.36	0.31	1.05
Kecepatan Angin	5.68	6.86	4.51
Arah Angin	61.41	48.37	67.14
Visibilitas	1.34	1.20	1.13
Tutupan Awan	14.50	11.07	16.22
Kelembapan Relatif	6.24	5.96	7.66

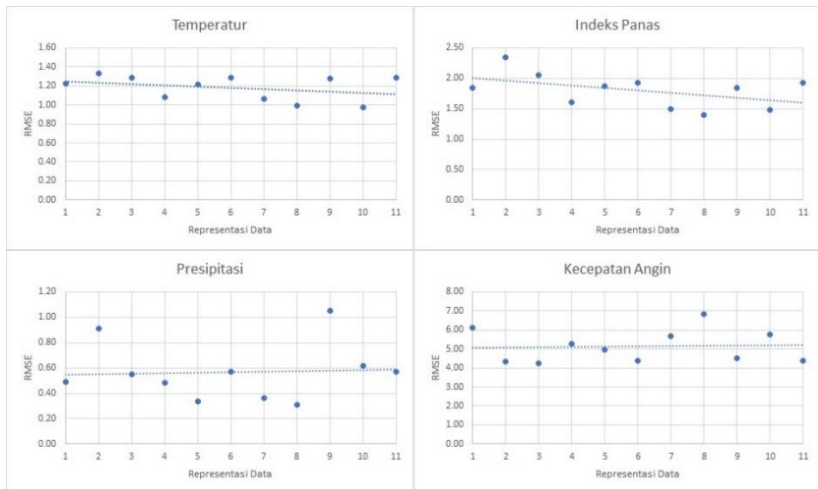
Tabel 4. 3. Hasil tiap parameter dengan rasio bulan latih dan uji 60:12

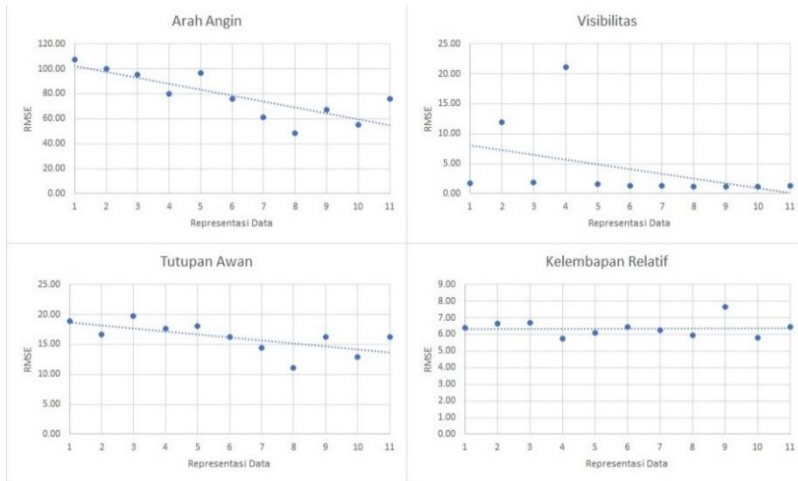
Parameter	RMSE	
	2015	2020
Temperatur	0.98	1.29
Indeks Panas	1.48	1.93
Presipitasi	0.62	0.57
Kecepatan Angin	5.75	4.38
Arah Angin	54.91	76.06
Visibilitas	1.16	1.31
Tutupan Awan	12.88	16.22
Kelembapan Relatif	5.79	6.44

Dilihat dari hasil yang didapat, RMSE pada parameter temperatur, indeks panas dan presipitasi cenderung rendah dengan kisaran nilai 0.62 – 2.34. Visibilitas fluktuatif dengan nilai 1.16 sampai 21.1. Parameter kecepatan angin, tutupan awan dan

kelembapan relatif memiliki RMSE yang sedang tetapi stabil, dan terakhir arah angin memiliki nilai RMSE sangat besar. Nilai RMSE yang didapat menggambarkan bahwa beberapa parameter tidak memiliki hasil regresi yang baik, yang akan mempengaruhi hasil klasifikasi di tahapan selanjutnya. Hasil RMSE ini akan menjadi pertimbangan parameter mana saja yang akan dihilangkan pada saat pengujian prediksi cuaca dengan parameter terbaik.

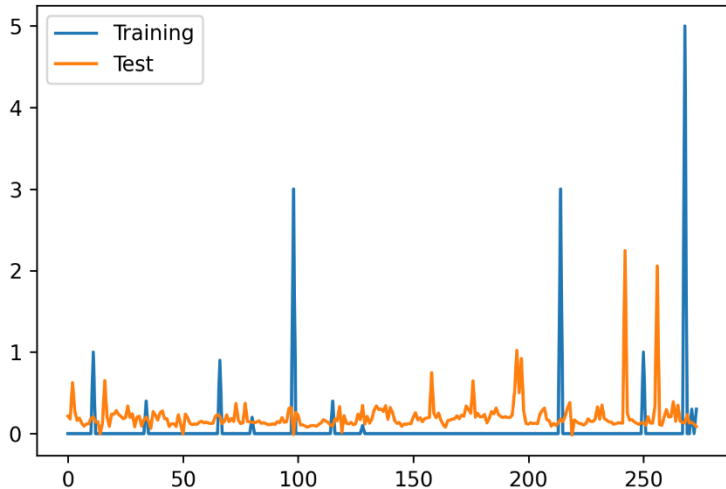
Pada Gambar 4. 1. Grafik RMSE tiap parameter dengan jangkauan waktu berbeda. menunjukan penurunan nilai RMSE pada 5 parameter dan kenaikan nilai RMSE pada 3 parameter. Representasi data yang dimaksud pada gambar adalah data dengan jangkauan waktu berbeda yang telah ditentukan (1-6 rasio jangkauan waktu 11:1, 7-9 rasio jangkauan waktu 30:6, 10-11 rasio jangkauan waktu 60:12). Pada gambar ini juga menjelaskan bahwa ada kecenderungan peningkatan kinerja model ketika rasio pelatihan dan pengujian semakin besar, sesuai dengan karakteristik dari ML yang akan memberikan model terbaik ketika data latih semakin banyak.





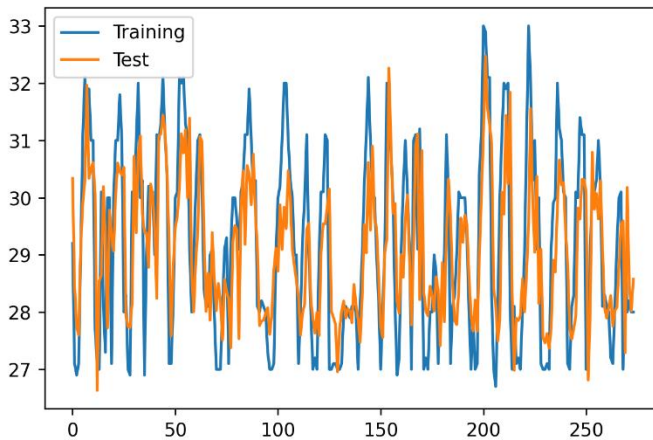
Gambar 4. 1. Grafik RMSE tiap parameter dengan jangkauan waktu berbeda.

Hasil RMSE yang didapatkan memiliki beberapa kekurangan, apabila diperhatikan lebih lanjut pada hasil perbandingan nilai latih dan uji pada Gambar 4. 2 pada parameter presipitasi memiliki nilai RMSE rendah tetapi tidak mencerminkan nilai sesungguhnya.

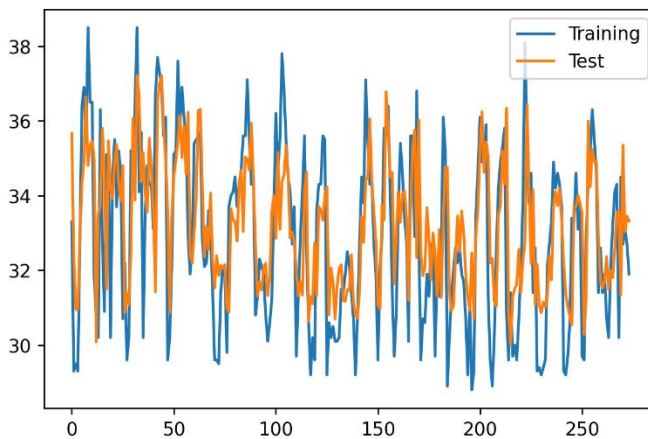


Gambar 4. 2. Perbandingan hasil uji dan latih pada parameter presipitasi (hasil regresi Desember 2010)

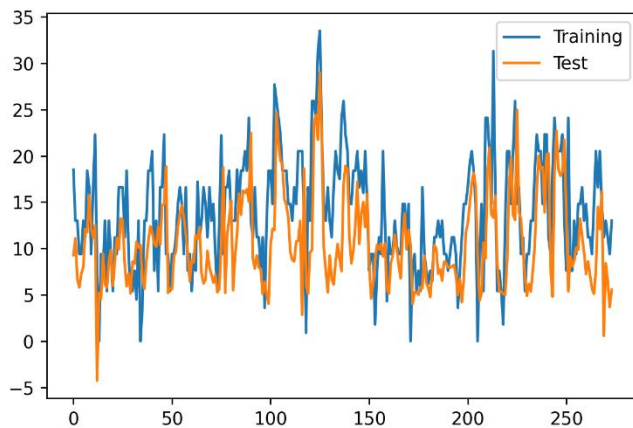
Parameter temperatur, indeks panas, arah angin, visibilitas, dan kelembapan relatif memiliki nilai RMSE kecil dengan perbandingan nilai uji dan latih menyerupai, sehingga parameter temperatur, indeks panas, arah angin, visibilitas, dan kelembapan relatif merupakan kriteria parameter terbaik yang dapat digunakan untuk tahapan klasifikasi selanjutnya.



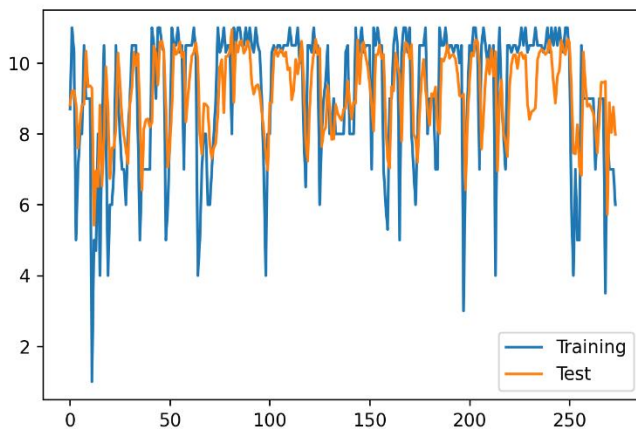
Gambar 4. 3. Perbandingan hasil uji dan latih pada parameter temperatur (hasil regresi Desember 2010)



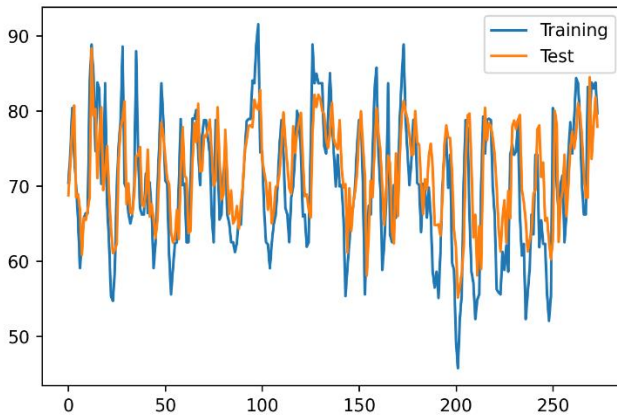
Gambar 4. 4. Perbandingan hasil uji dan latih pada parameter indeks panas (hasil regresi Desember 2010)



Gambar 4. 5. Perbandingan hasil uji dan latih pada parameter kecepatan angin (hasil regresi Desember 2010)

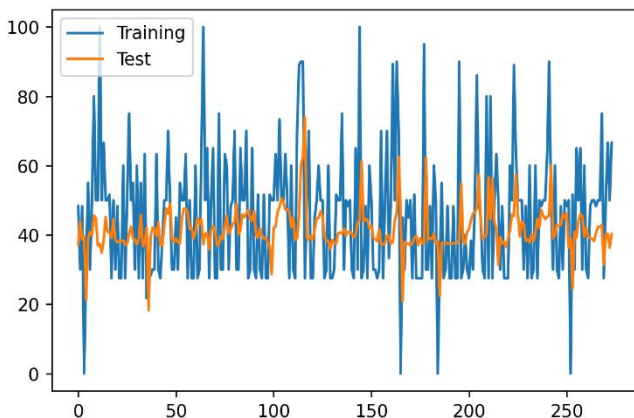


Gambar 4. 6. Perbandingan hasil uji dan latih pada parameter visibilitas (hasil regresi Desember 2010)

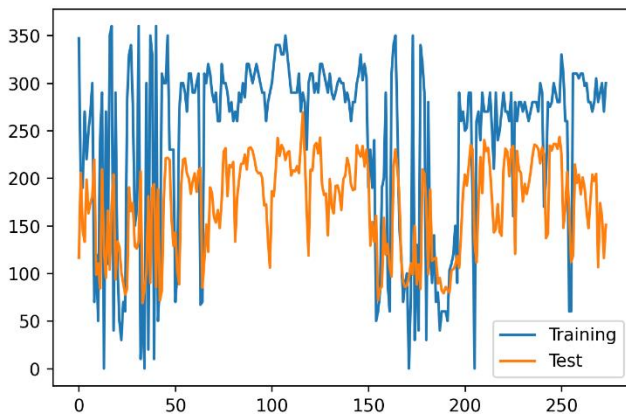


Gambar 4. 7. Perbandingan hasil uji dan latih pada parameter kelembapan relatif (hasil regresi Desember 2010)

Parameter tutupan awan dan arah angin memiliki nilai RMSE tidak terlalu besar dengan perbandingan nilai uji dan latih cukup menyerupai, sehingga perlu pengujian lanjut pada tahapan klasifikasi untuk menentukan apakah parameter tutupan awan dan arah angin perlu dihilangkan atau tetap bisa dipertahankan. Hasil RMSE serta perbandingan nilai uji dan latih cenderung memiliki pola yang sama pada parameter yang telah disebutkan untuk jangka waktu yang berbeda, sehingga kesimpulan yang ditarik dapat digunakan untuk seluruh parameter pada seluruh jangkauan waktu yang diuji. Gambar perbandingan uji dan latih pada jangkauan waktu berbeda dapat dilihat pada LAMPIRAN A.



Gambar 4. 8. Perbandingan hasil uji dan latih pada parameter tutupan awan (hasil regresi Desember 2010)



Gambar 4. 9. Perbandingan hasil uji dan latih pada parameter arah angin (hasil regresi Desember 2010)

4.2 Prediksi Cuaca Dengan Seluruh Parameter

Hasil klasifikasi pada tahapan selanjutnya menghasilkan prediksi cuaca dengan jangkauan waktu uji. Pengujian dapat dinilai dengan melihat nilai akurasi hasil prediksi dibandingkan data sebenarnya. Pengujian prediksi cuaca pertama dilakukan dengan memasukan seluruh parameter dan melihat hasil akurasinya. Pada Tabel 4. 4. Tabel hasil akurasi dengan seluruh parameter menunjukkan hasil akurasi sangat bervariasi bergantung pada jangkauan waktu. Hasil ini diakibatkan kualitas data yang bervariasi pada tiap jangkauan waktu. Secara keseluruhan, hasil akurasi yang didapatkan dengan melibatkan seluruh parameter menghasilkan akurasi 66% - 94%.

Tabel 4. 4. Tabel hasil akurasi dengan seluruh parameter

Rasio Bulan Latih dan Uji	Bulan Prediksi	Hasil Akurasi
11:1	Desember 2010	86%
	Desember 2011	91%
	Desember 2012	87%
	Desember 2013	80%
	Desember 2014	77%
	Desember 2015	88%
30:6	Juli – Desember 2012	75%
	Juli – Desember 2015	94%
	Juli – Desember 2018	66%
60:12	Januari – Desember 2015	85%
	Januari – Desember 2020	88%

Apabila diperhatikan, hasil klasifikasi yang didapatkan sudah cukup baik untuk memprediksi secara umum kondisi cuaca yang akan terjadi. Hasil prediksi dengan melibatkan seluruh parameter menunjukkan bagaimana model bisa memprediksi tanpa melihat kualitas data, sehingga untuk menghasilkan hasil yang lebih baik, parameter akan diuji dengan menghapus tiap-tiap parameter dan menguji akurasinya.

4.3 Prediksi Cuaca Dengan Parameter Pilihan

Pengujian selanjutnya pada model prediksi kondisi cuaca dengan klasifikasi adalah melihat korelasi tiap parameter pada data. Pengujian ini dilakukan dengan menjalankan program klasifikasi secara berulang dengan 3 tahapan, yaitu menggunakan seluruh parameter, lalu meninggalkan satu parameter secara bergantian, kemudian meninggalkan parameter yang dilihat memiliki dampak buruk pada hasil akurasi.

Pada Tabel 4. 5, Tabel 4. 6 dan Tabel 4. 7 menunjukkan bahwa akurasi akan berkurang ketika parameter yang dihilangkan ternyata berperan penting dalam memberikan analisis pola di ML. Parameter yang penting tersebut adalah temperatur, indeks panas dan kecepatan angin. Parameter selanjutnya akan mengurangi akurasi pada beberapa jangkauan waktu tetapi tidak semuanya, sehingga parameter ini masih dapat dimasukkan ke parameter yang cukup penting. Parameter cukup penting adalah visibilitas dan kelembapan relatif. Selanjutnya, pada parameter yang memberikan dampak kurang baik bagi model akan meningkatkan akurasi ketika parameter tersebut dihilangkan, parameter yang memiliki dampak kurang baik tersebut adalah presipitasi, tutupan awan dan arah angin dengan urutan yang sesuai dari parameter yang memiliki dampak paling buruk ke model berdasarkan nilai akurasi.

Tabel 4. 5. Tabel akurasi dengan berbagai penggunaan parameter pada rasio latih dan uji bulanan 11:1

Penggunaan Parameter	Akurasi Prediksi					
	2010	2011	2012	2013	2014	2015
Seluruh Parameter	86%	91%	87%	80%	77%	88%
Tanpa Temperatur	86%	90%	86%	78%	75%	89%
Tanpa Indeks Panas	86%	90%	86%	78%	69%	89%
Tanpa Presipitasi	86%	91%	89%	87%	91%	90%
Tanpa Kecepatan Angin	85%	90%	86%	76%	73%	88%
Tanpa Arah Angin	85%	90%	83%	60%	69%	88%
Tanpa Visibilitas	86%	90%	86%	80%	77%	88%
Tanpa Tutupan Awan	86%	92%	86%	82%	61%	89%
Tanpa Kelembapan Relatif	86%	91%	88%	78%	70%	88%

Tabel 4. 6. Tabel akurasi dengan berbagai penggunaan parameter pada rasio latih dan uji bulanan 30:6

Penggunaan Parameter	Akurasi Prediksi		
	2012	2015	2018
Seluruh Parameter	75%	94%	66%
Tanpa Temperatur	76%	94%	60%
Tanpa Indeks Panas	76%	94%	57%
Tanpa Presipitasi	79%	94%	88%
Tanpa Kecepatan Angin	75%	84%	59%
Tanpa Arah Angin	74%	94%	54%
Tanpa Visibilitas	72%	94%	53%
Tanpa Tutupan Awan	75%	94%	49%
Tanpa Kelembapan Relatif	76%	94%	53%

Tabel 4. 7. Tabel akurasi dengan berbagai penggunaan parameter pada rasio latih dan uji bulanan 60:12

Penggunaan Parameter	Akurasi Prediksi	
	2015	2020
Seluruh Parameter	85%	88%
Tanpa Temperatur	86%	89%
Tanpa Indeks Panas	87%	89%
Tanpa Presipitasi	86%	90%
Tanpa Kecepatan Angin	78%	88%
Tanpa Arah Angin	84%	88%
Tanpa Visibilitas	90%	88%
Tanpa Tutupan Awan	83%	89%
Tanpa Kelembapan Relatif	87%	88%

Berdasarkan pengujian sebelumnya dapat disimpulkan bahwa ada tiga parameter yang memberikan dampak negatif ke model, secara berurut dari yang memiliki dampak terburuk adalah presipitasi, tutupan awan dan arah angin. Pengujian selanjutnya dilakukan dengan menjalankan program klasifikasi dengan meninggalkan parameter presipitasi dan tutupan awan pada pengujian pertama, lalu meninggalkan parameter presipitasi, tutupan awan serta arah angin pada pengujian kedua.

Hasil yang didapatkan dari pengujian ini dapat dilihat pada Tabel 4. 8, Tabel 4. 9 dan Tabel 4. 10 untuk tiga jenis rasio jangkauan waktu. Apabila hanya parameter presipitasi saja yang dihilangkan, peningkatan akurasi terjadi pada 8 dari 11 representasi data. Menghilangkan parameter presipitasi dan tutupan awan meningkatkan akurasi pada 9 dari 11 representasi data dan menghilangkan parameter presipitasi, tutupan awan serta arah angin meningkatkan akurasi pada 10 dari 11 representasi data. Tidak berpengaruh atau bahkan berkurangnya akurasi ketika parameter kurang baik dihilangkan dapat terjadi apabila parameter terbaik lainnya mampu memberikan pola

prediksi yang sangat baik sehingga secara otomatis model akan mengurangi bagian dari parameter yang kurang baik ketika proses pelatihan dijalankan.

Tabel 4. 8. Tabel akurasi dengan perbandingan penggunaan parameter terbaik pada rasio latih uji 11:1

Penggunaan Parameter	Akurasi Prediksi					
	2010	2011	2012	2013	2014	2015
Seluruh Parameter	86%	91%	87%	80%	77%	88%
Tanpa Presipitasi	86%	91%	89%	87%	91%	90%
Tanpa Presipitasi dan Tutupan Awan	86%	93%	89%	91%	85%	91%
Tanpa Presipitasi, Tutupan Awan dan Arah Angin	87%	93%	89%	90%	87%	91%

Tabel 4. 9. Tabel akurasi dengan perbandingan penggunaan parameter terbaik pada rasio latih uji 30:6

Penggunaan Parameter	Akurasi Prediksi		
	2012	2015	2018
Seluruh Parameter	75%	94%	66%
Tanpa Presipitasi	79%	94%	88%
Tanpa Presipitasi dan Tutupan Awan	79%	94%	86%
Tanpa Presipitasi, Tutupan Awan dan Arah Angin	79%	95%	82%

Tabel 4. 10. Tabel akurasi dengan perbandingan penggunaan parameter terbaik pada rasio latih uji 60:12

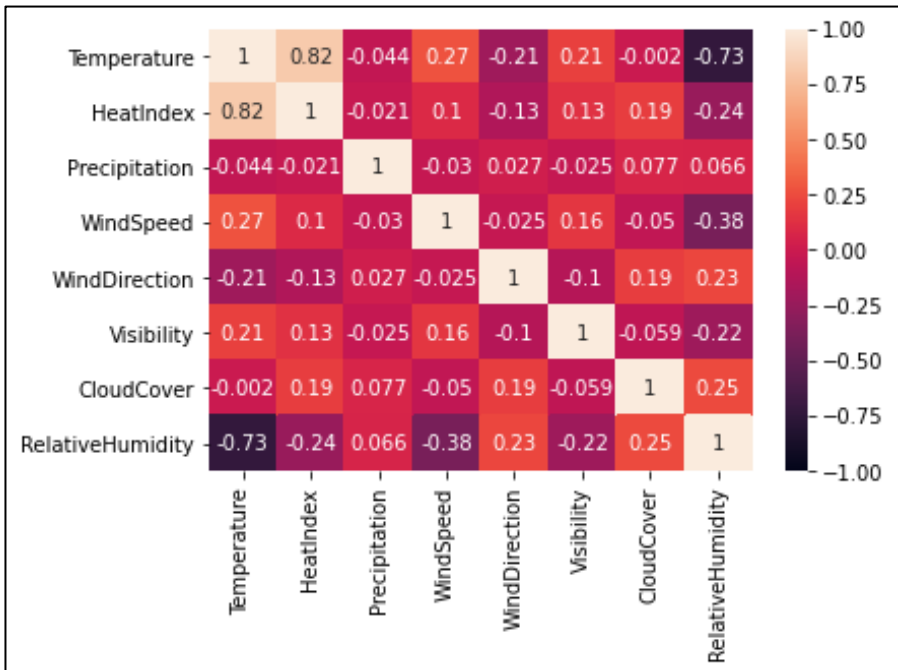
Penggunaan Parameter	Akurasi Prediksi	
	2015	2020
Seluruh Parameter	85%	88%
Tanpa Presipitasi	86%	90%
Tanpa Presipitasi dan Tutupan Awan	84%	91%
Tanpa Presipitasi, Tutupan Awan dan Arah Angin	84%	91%

4.4 Korelasi Parameter Pada Model dan Fenomena Fisis

Pengujian korelasi antar parameter bisa menggunakan *heat map correlation* yang dijalankan dengan kode program python. *Heat map correlation* dapat melihat keterikatan antar parameter berdasarkan data set. Hubungan antara parameter tersebut belum tentu berlanjut hingga pembuatan model ML, dikarenakan model ML lebih melihat kualitas data dan pola yang dibuat dari data tersebut, apabila bentuk data kurang baik maka sebaik apapun korelasi yang ditampilkan di *heat map correlation* menjadi tidak relevan. Sehingga *heat map correlation* digunakan untuk menggambarkan korelasi parameter di alam, bukan korelasi parameter di ML yang dibangun.

Gambar 4. 10. Heat Map Correlation dari data set menunjukan *heat map correlation*, indeks panas terikat dengan temperatur dan kelembapan relatif, dengan indeks panas sendiri merupakan hasil perhitungan dari temperatur dan kelembapan relatif. Beberapa

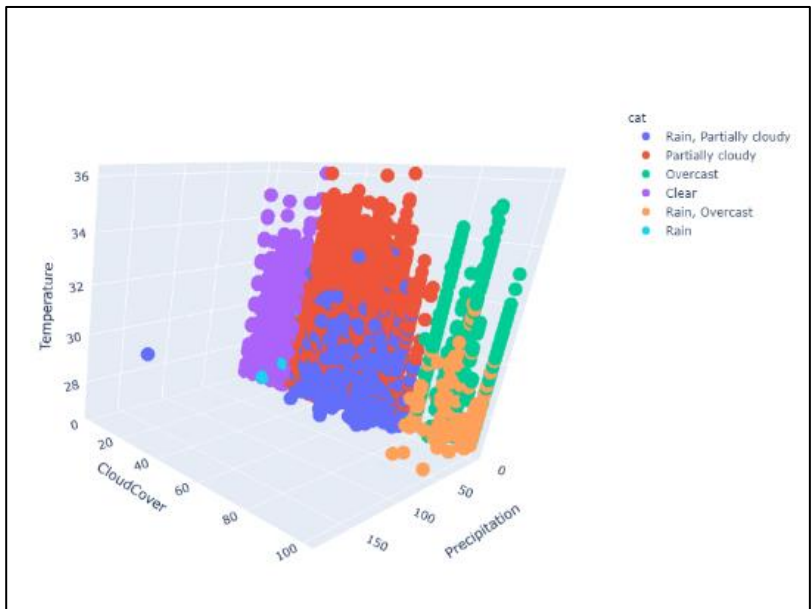
parameter hampir tidak memiliki korelasi sama sekali seperti tutupan awan dan temperatur yang memiliki nilai korelasi 0.002.



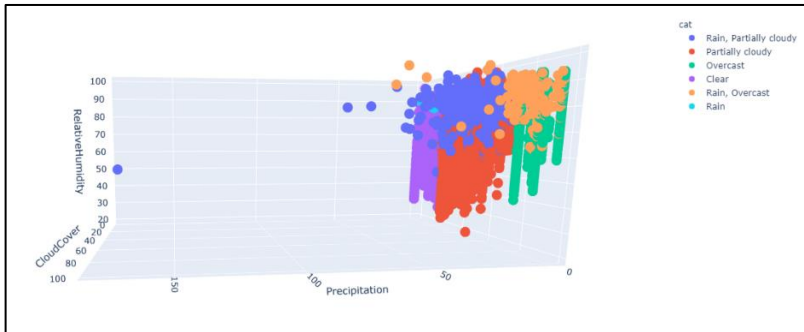
Gambar 4. 10. Heat Map Correlation dari data set

Penggunaan algoritma klasifikasi *K – Nearest Neighbours* berjalan dengan cara mengidentifikasi *tetangga* pada sekitaran data yang ingin diklasifikasikan hingga ditemukan pola yang membentuk data tersebut. Apabila data set divisualisasikan maka tiap-tiap parameter akan membentuk pola tertentu yang menggambarkan posisi mereka ketika kondisi cuaca tertentu terjadi seperti yang ditampilkan pada Gambar 4. 11, Gambar 4. 12, Gambar 4. 13, dan Gambar 4. 14. Terlihat bahwa ketika cuaca sedang hujan, mendung hujan dan

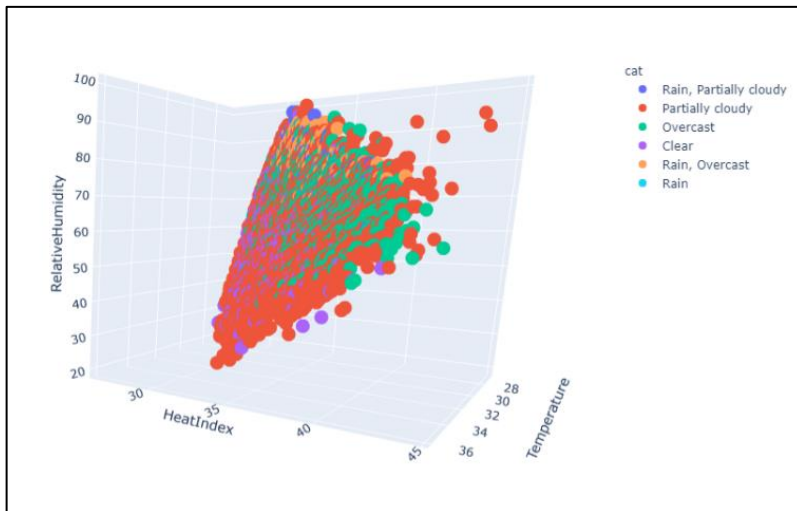
konduksi cuaca lainnya memiliki parameter khusus dan pola yang menggambarkan kondisi cuaca tersebut, sehingga penggunaan algoritma klasifikasi *K – Nearest Neighbours* cukup tepat, bukan hanya dilihat dari cukup tingginya akurasi dengan pengujian sebelumnya tetapi juga dari hasil visualisasi data set.



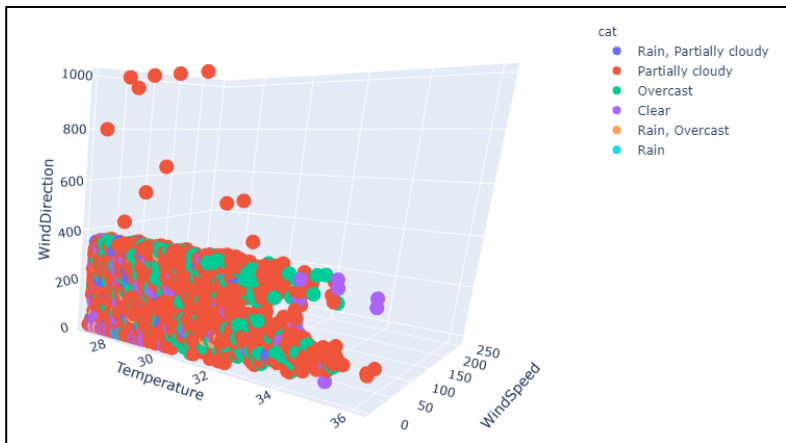
Gambar 4. 11. Visualisasi data untuk korelasi parameter temperatur, tutupan awan dan presipitasi



Gambar 4. 12. Visualisasi data untuk korelasi parameter presipitasi, tutupan awan dan kelembapan relatif.



Gambar 4. 13. Visualisasi data untuk korelasi parameter temperatur, indeks panas dan kelembapan relatif.



Gambar 4. 14. Visualisasi data untuk korelasi parameter temperatur, arah angin dan kecepatan angin.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan hasil penelitian yang telah dilakukan, maka kesimpulan dari penelitian ini adalah sebagai berikut:

1. Model regresi parameter LSTM dengan 2 *hidden layer* terdiri dari 32 neuron pada *hidden layer* 1 dan 64 neuron pada *hidden layer* 2 mampu memberikan prediksi untuk tiap parameter, dan model klasifikasi mampu memberikan prediksi kondisi cuaca berdasarkan parameter yang telah di regresikan tersebut.
2. Hasil regresi parameter menghasilkan beberapa prediksi parameter yang kurang baik dikarenakan kurang baik juga pola parameter dari data set yang diberikan. Prediksi parameter kurang baik tersebut adalah parameter presipitasi, tutupan awan, dan arah angin. Hasil regresi parameter dievaluasi dengan melihat nilai RMSE lalu mencari nilai terkecilnya. Hasil regresi juga semakin baik apabila rasio data latih dan uji semakin besar (nilai RMSE semakin kecil ketika rasio latih dan uji 60:12 dibandingkan dengan rasio latih uji 11:1).
3. Model klasifikasi mampu memberikan prediksi kondisi cuaca dengan akurasi cukup baik pada kisaran 77% - 91% ketika menggunakan seluruh parameter. Apabila parameter kurang baik seperti presipitasi, tutupan awan dan arah angin ditinggalkan, maka terdapat peningkatan akurasi dari prediksi kondisi cuaca.
4. Korelasi antar parameter yang membentuk kondisi cuaca secara fisis tidak berpengaruh terhadap korelasi antar parameter di model ML, dikarenakan ML membentuk pola prediksi berdasarkan data, sehingga apabila bentuk data yang didapatkan di data set kurang baik, maka parameter tersebut akan memiliki korelasi yang rendah sebesar apapun pengaruhnya pada fenomena fisis sesungguhnya.

5.2 Saran

Pada penelitian selanjutnya, dapat dilakukan dengan beberapa saran sebagai berikut:

1. Mencari data set yang memiliki parameter lebih lengkap sehingga korelasi antar parameter dapat diteliti lebih lanjut.
2. Menggunakan model ML yang berbeda dengan *loss function* yang berbeda pula untuk menguji model tersebut.
3. Memprediksi kondisi cuaca secara regional berdasarkan kondisi cuaca regional sekitarnya.

DAFTAR PUSTAKA

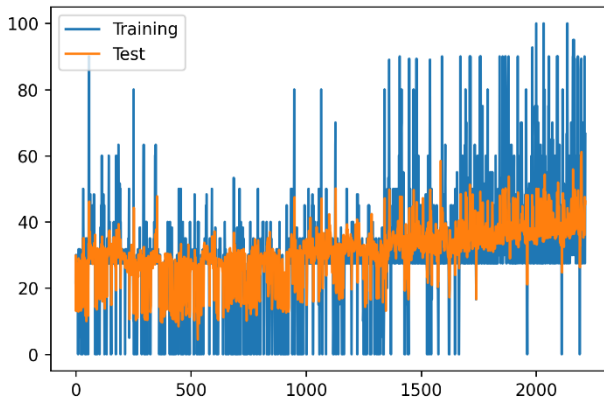
- Abadi, M. (2016). TensorFlow: learning functions at scale. *ACM SIGPLAN Notices*, 51(9), 1–1. <https://doi.org/10.1145/3022670.2976746>
- Brownlee, J. (2019, September 13). *Introduction to Python Deep Learning with Keras*. <https://machinelearningmastery.com/introduction-python-deep-learning-library-keras/>
- Chai, T., & Draxler, R. R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? -Arguments against avoiding RMSE in the literature. *Geoscientific Model Development*, 7(3), 1247–1250. <https://doi.org/10.5194/gmd-7-1247-2014>
- Cogato, A., Meggio, F., Migliorati, M. D. A., & Marinello, F. (2019). Extreme weather events in agriculture: A systematic review. *Sustainability (Switzerland)*, 11(9), 1–18. <https://doi.org/10.3390/su11092547>
- Cunningham, P., & Delany, S. J. (2020). k-Nearest Neighbors classifiers 2nd edition (with python examples). *ArXiv*, 1, 1–22.
- Czum, J. M. (2020). Dive Into Deep Learning. *Journal of the American College of Radiology*, 17(5), 637–638. <https://doi.org/10.1016/j.jacr.2020.02.005>
- Dubois, P. F., Oliphant, T. E., Pérez, F., Granger, B. E., & Greenfield, P. (2007). *PYTHON: Guest Editor 's Introduction Python for Scientific Computing IPython: A System for Interactive Scientific Computing Reaching for the Stars with Python* (Issue June).
- Khosravi, A., Koury, R. N. N., Machado, L., & Pabon, J. J. G. (2018). Prediction of wind speed and wind direction using artificial neural network, support vector regression and adaptive neuro-fuzzy inference system. *Sustainable Energy Technologies and Assessments*, 25(December 2017), 146–160. <https://doi.org/10.1016/j.seta.2018.01.001>

- Kirono, D. G. C., Butler, J. R. A., McGregor, J. L., Ripaldi, A., Katzfey, J., & Nguyen, K. (2016). Historical and future seasonal rainfall variability in Nusa Tenggara Barat Province, Indonesia: Implications for the agriculture and water sectors. *Climate Risk Management*, 12, 45–58. <https://doi.org/10.1016/j.crm.2015.12.002>
- Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. <https://doi.org/10.1038/nature14539>
- Lions, J. L., Temam, R., & Wang, S. (1992). New formulations of the primitive equations of atmosphere and applications. *Nonlinearity*, 5(2), 237–288. <https://doi.org/10.1088/0951-7715/5/2/001>
- Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). *A Critical Review of Recurrent Neural Networks for Sequence Learning*. 1–38. <http://arxiv.org/abs/1506.00019>
- Lu, H., Li, Y., Chen, M., Kim, H., & Serikawa, S. (2017). *Brain Intelligence: Go Beyond Artificial Intelligence*. 24(2). <http://arxiv.org/abs/1706.01040>
- Morgan, P. (2018). Machine Learning Is Changing the Rules. In *O'Reilly* (Issue December). <https://www.safaribooksonline.com/library/view/machine-learning-is/9781492035367/>
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2014). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 39(2014), i–ii.
- Potter, T. D., & Coleman, B. R. (2003). *HANDBOOK OF WEATHER, CLIMATE, AND WATER Dynamics, Climate, Physical Meteorology, Weather Systems, and Measurements*. John Wiley and Sons, Inc.

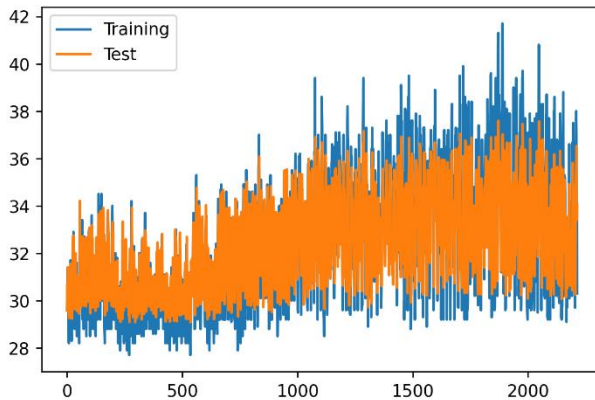
- Raschka, S., & Mirjalili, V. (2017). *Python Machine Learning* (Second Edi). Packt Publishing Ltd.
https://books.google.co.id/books?hl=en&lr=&id=_plGDwAAQBAJ&oi=fnd&pg=PP1&dq=python+machine+learning&ots=8tCNgWbmDH&sig=e4rQjE95vOAFhZeMIwTu6UdDIlg&redir_esc=y#v=onepage&q&f=false
- Tosepu, R., Gunawan, J., Effendy, D. S., Ahmad, L. O. A. I., Lestari, H., Bahar, H., & Asfian, P. (2020). Correlation between weather and Covid-19 pandemic in Jakarta, Indonesia. *Science of the Total Environment*, 725, 138436.
<https://doi.org/10.1016/j.scitotenv.2020.138436>
- Volokitin, A., Timofte, R., & Van Gool, L. (2016). Deep Features or Not: Temperature and Time Prediction in Outdoor Scenes. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 1136–1144.
<https://doi.org/10.1109/CVPRW.2016.145>
- Watts, A. (2014). *The weather handbook* (3rd ed.). Adlard Coles Nautical Press.
https://books.google.co.id/books?id=c63vAwAAQBAJ&dq=The Weather Handbook&source=gbs_book_other_versions
- Wirjohamidjojo, S., & Swarinoto, Y. (2010). *Iklim Kawasan Indonesia (Dari Aspek Dinamik - Sinoptik)*. Badan Meteorologi Klimatologi dan Geofisika.
- Zhang, Z., Ma, H., Fu, H., & Zhang, C. (2016). Scene-free multi-class weather classification on single images. *Neurocomputing*, 207, 365–373. <https://doi.org/10.1016/j.neucom.2016.05.015>

LAMPIRAN A

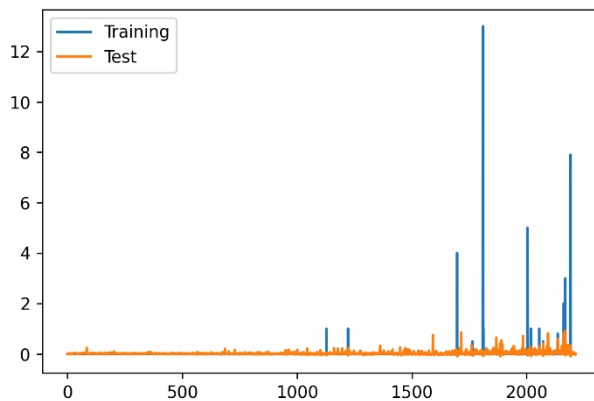
DATA HASIL PENELITIAN



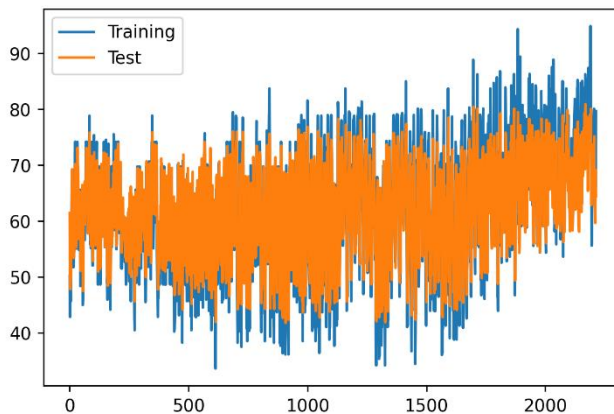
Gambar A. 1. Perbandingan hasil uji dan latih pada parameter tutupan awan (hasil regresi Juli - Desember 2012)



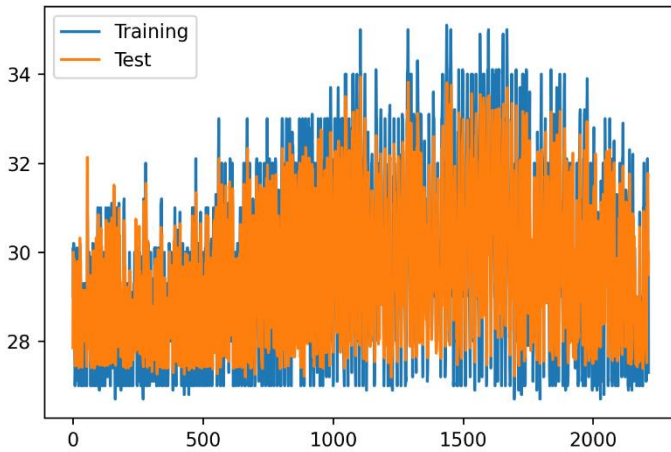
Gambar A. 2. Perbandingan hasil uji dan latih pada parameter indeks panas (hasil regresi Juli - Desember 2012)



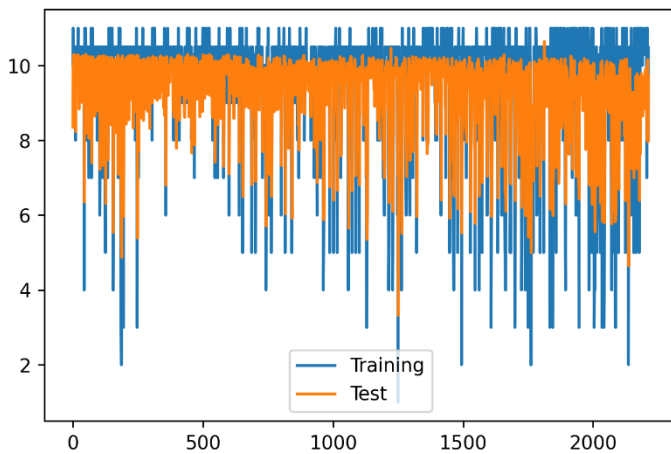
Gambar A. 3. Perbandingan hasil uji dan latih pada parameter presipitasi (hasil regresi Juli - Desember 2012)



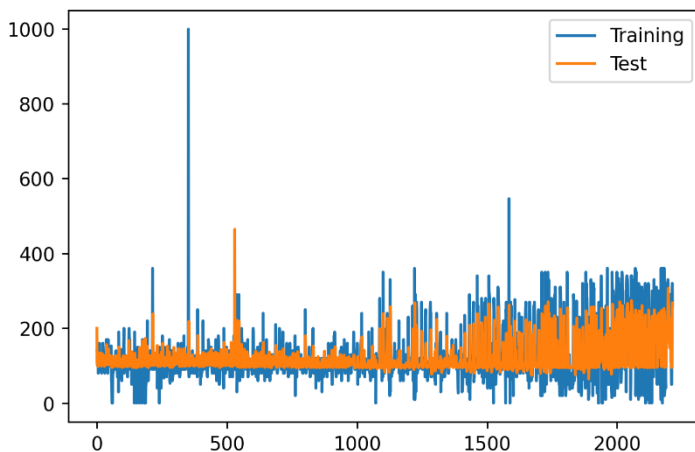
Gambar A. 4. Perbandingan hasil uji dan latih pada parameter kelembapan relatif (hasil regresi Juli - Desember 2012)



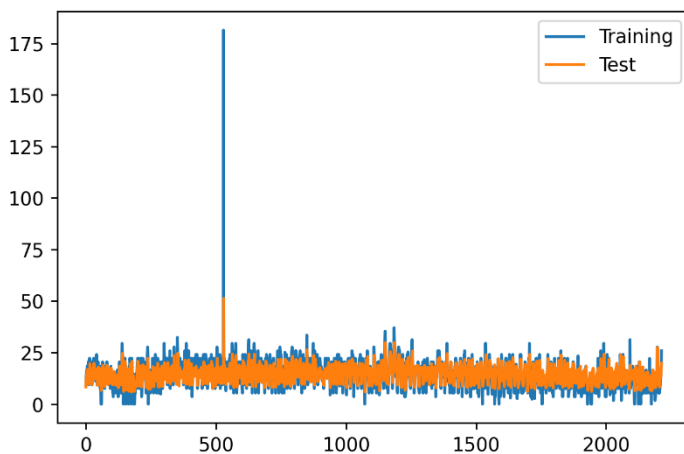
Gambar A. 5. Perbandingan hasil uji dan latih pada parameter temperatur (hasil regresi Juli - Desember 2012)



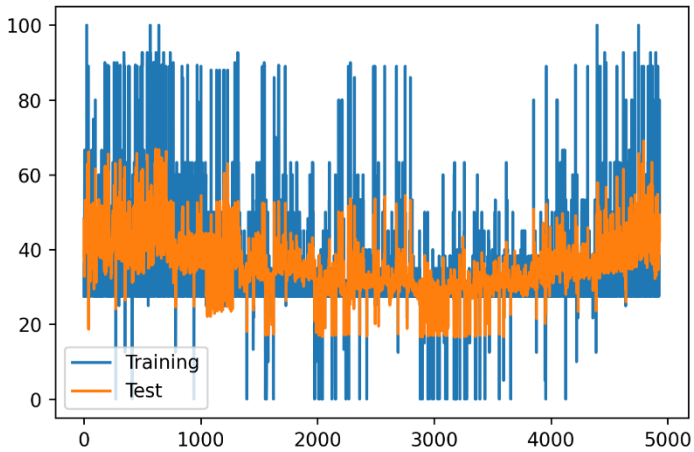
Gambar A. 6. Perbandingan hasil uji dan latih pada parameter visibilitas (hasil regresi Juli - Desember 2012)



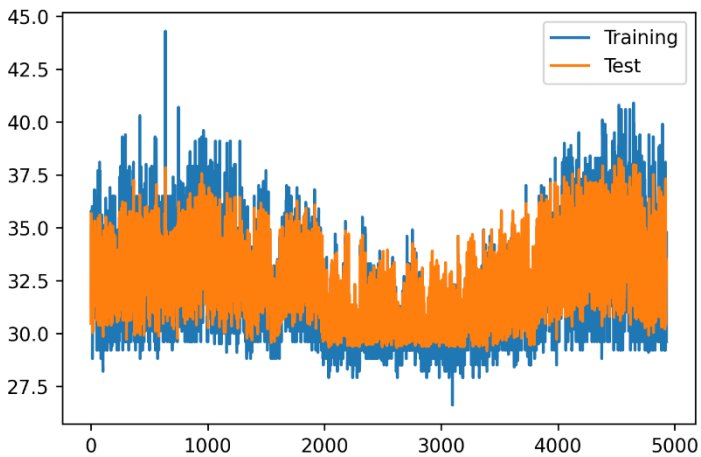
Gambar A. 7. Perbandingan hasil uji dan latih pada parameter arah angin (hasil regresi Juli - Desember 2012)



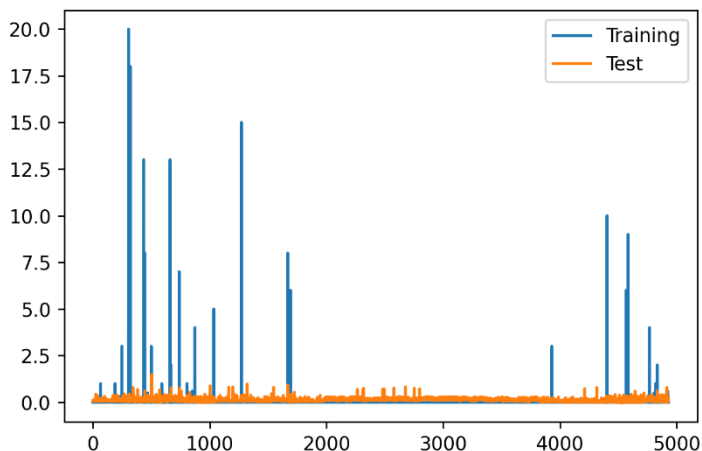
Gambar A. 8. Perbandingan hasil uji dan latih pada parameter kecepatan angin (hasil regresi Juli - Desember 2012)



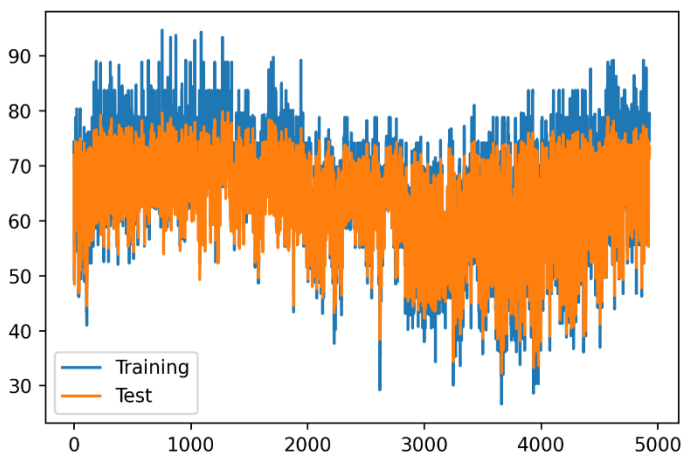
Gambar A. 9. Perbandingan hasil uji dan latih pada parameter tutupan awan (hasil regresi Januari - Desember 2015)



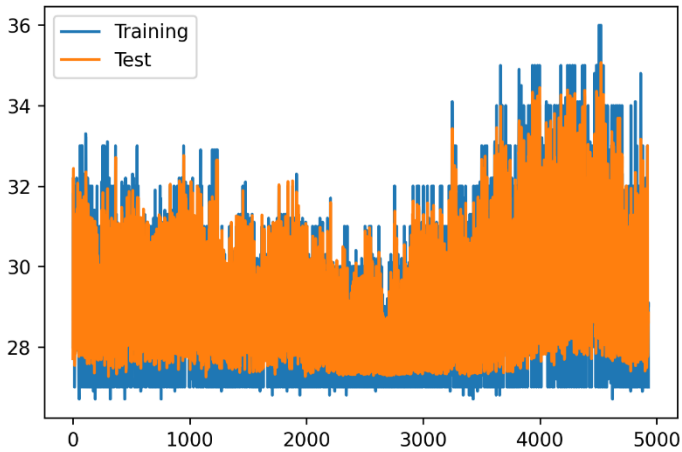
Gambar A. 10. Perbandingan hasil uji dan latih pada parameter indeks panas (hasil regresi Januari - Desember 2015)



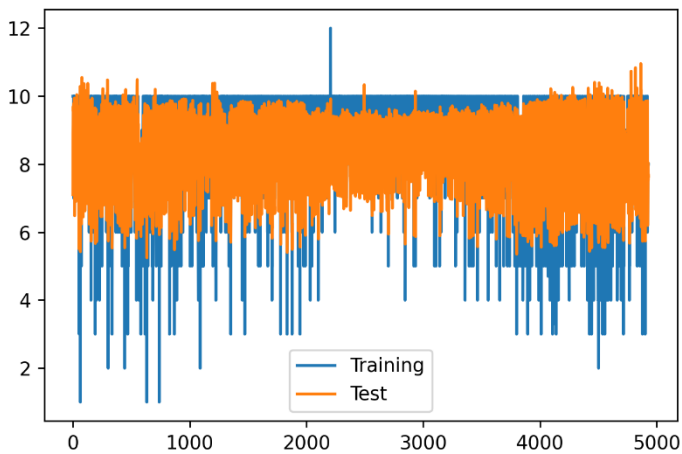
Gambar A. 11. Perbandingan hasil uji dan latih pada parameter presipitasi (hasil regresi Januari - Desember 2015)



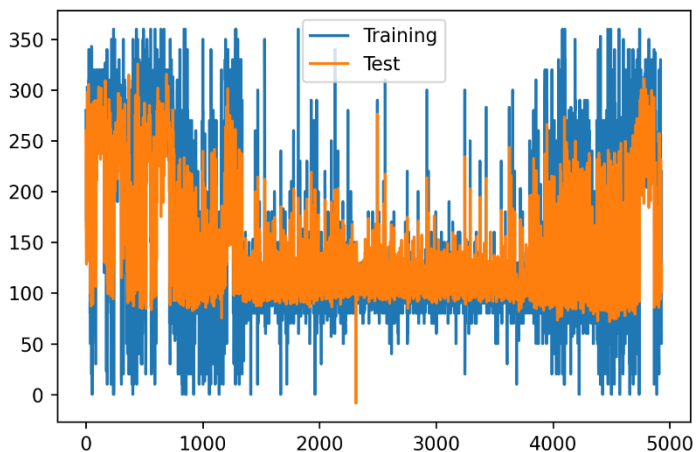
Gambar A. 12. Perbandingan hasil uji dan latih pada parameter kelembapan relatif (hasil regresi Januari - Desember 2015)



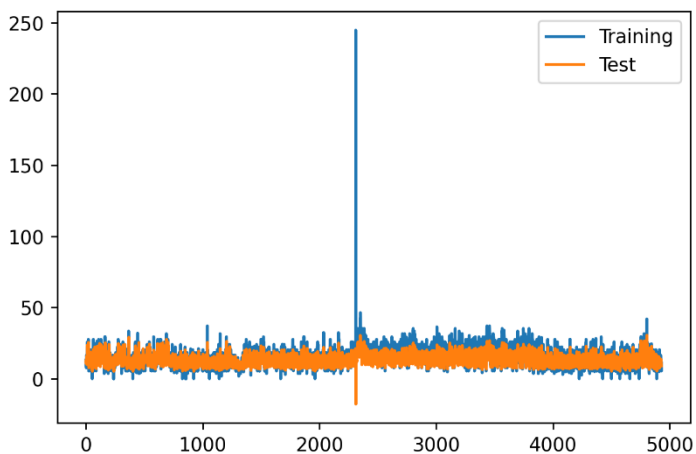
Gambar A. 13. Perbandingan hasil uji dan latih pada parameter temperatur (hasil regresi Januari - Desember 2015)



Gambar A. 14. Perbandingan hasil uji dan latih pada parameter visibilitas (hasil regresi Januari - Desember 2015)



Gambar A. 15. Perbandingan hasil uji dan latih pada parameter arah angin (hasil regresi Januari - Desember 2015)



Gambar A. 16. Perbandingan hasil uji dan latih pada parameter kecepatan angin (hasil regresi Januari - Desember 2015)

LAMPIRAN B KODE PROGRAM

Kode Program RNN-LSTM

```
##Importing library##
import numpy as np
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler,
OneHotEncoder

##Inserting data as needed##
nskip = 12305+4336+4842
ndata = 4932
ntrain = 4497
nbatch = 30
df = pd.read_csv('dataMalang-Modified.csv', nrows =
ndata, skiprows = nskip ) #, nrows=365*30, skiprow,
skipfooter
df = df.dropna()
df.columns = ['Date',
'Temperature', 'HeatIndex', 'Precipitation', 'WindSpeed',
'WindDirection', 'Visibility', 'CloudCover', 'RelativeHumid
ity',
'Conditions']
#####
#####
##Temperature##
## Reading data ##
Predict_Var = 0 #Choosing variable to do regression
Namefile = 'PredTemperature.csv'
# File Name: PredTemperature.csv, PredHeatIndex.csv,
PredPrecipitation.csv,
# PredWindSpeed.csv, PredWindDirection.csv,
PredVisibility.csv, PredCloudCover.csv,
# PredRelativeHumidity.csv
```

```

## Separating dates ##
data_dates = pd.to_datetime(df['Date'])

## Choosing variabel to use ##
cols = list(df)[1:10]
encoder = OneHotEncoder()
df_for_training = df[cols]
values = df_for_training.values
valuesT = values[:,8]
valuesT = valuesT.reshape((valuesT.shape[0], 1))
valuesT = encoder.fit_transform(valuesT).toarray()
values = np.delete(values,8,1)
values = np.append(values, valuesT, axis = 1)
df_for_training = pd.DataFrame(values)

## Normalize the dataset with range 0-1 ##
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(df_for_training)
df_for_training_scaled =
scaler.transform(df_for_training)

##Splitting data to test and training##
trainX = []
trainY = []
testX = []
testY = []
ntraining = ntrain
train = len(df_for_training_scaled[:ntraining,-1])
test = len(df_for_training_scaled[ntraining:-1])

n_future = 1    # Number of days we want to predict into
the future
n_past = 1      # Number of past days we want to use to
predict the future

##Create time series data ##
for i in range(n_past, len(df_for_training_scaled) -
n_future +1):
    if i <= len(df_for_training_scaled[:ntraining,-1]):
        trainX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])

```

```

        trainY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
    else:
        testX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
        testY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
trainX, trainY, testX, testY = np.array(trainX),
np.array(trainY), np.array(testX), np.array(testY)

## Check data shape ##
print('trainX shape == {}'.format(trainX.shape))
print('trainY shape == {}'.format(trainY.shape))
print('testX shape == {}'.format(testX.shape))
print('testY shape == {}'.format(testY.shape))

## Train Dates adjustment ##
train_dates = data_dates[:ntraining]
test_dates = data_dates[(ntraining+1):]
train_dates = pd.DataFrame(train_dates)
test_dates = pd.DataFrame(test_dates)
train_dates.reset_index(drop=True, inplace=True)
test_dates.reset_index(drop=True, inplace=True)

## Create LSTM Model ##
model = Sequential()
model.add(LSTM(64, activation='relu',
input_shape=(trainX.shape[1], trainX.shape[2]),
return_sequences=True))
model.add(LSTM(32, activation='relu',
return_sequences=False))
model.add(Dense(trainY.shape[1]))
model.compile(optimizer='adam', loss='mse')
model.summary()

## Fitting Model ##
history = model.fit(trainX, trainY, epochs=100,
batch_size=nbatch,
validation_data=(testX, testY),
verbose=1, shuffle=False)
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation

```

```

loss')
plt.legend()
plt.show()

## Make prediction based on Model ##
yhat = model.predict(testX)
testX = testX.reshape((testX.shape[0], testX.shape[2]))
yhat = yhat.reshape((yhat.shape[0]))
trainX = trainX.reshape((trainX.shape[0],
trainX.shape[2]))
trainY = trainY.reshape((trainY.shape[0]))

## Revert parameter variable ##
inv_x = testX
inv_x = scaler.inverse_transform(inv_x)
inv_x = np.delete(inv_x, [8,9,10,11,12], axis = 1)

## Revert scaling prediction ##
inv_yhat = testX
inv_yhat[:,Predict_Var] = yhat
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat1 = inv_yhat[:,Predict_Var] #Just taking
predicted variable

## Revert scaling actual data ##
testY = testY.reshape((len(testY)))
inv_y = testX
inv_y[:,Predict_Var] = testY
inv_y = scaler.inverse_transform(inv_y)
inv_y1 = inv_y[:,Predict_Var] #Just taking predicted
variable

## y1 = actual data result, yhat1 = predicted data
result

## calculate MSE ##
mse = mean_squared_error(inv_y1, inv_yhat1)
rmse = np.roots(mse)
print('Test RMSE: %.3f' % rmse)
plt.plot(inv_y1, label='Training')
plt.plot(inv_yhat1, label='Test')
plt.legend()

```

```

plt.savefig('Temperature.png', dpi=250)
plt.show()
f = open('RMSE.txt','a')
print("Temperature",mse, file=f)
f.close()

## Create training data for classification ##
inv_xtest = trainX
inv_xtest = scaler.inverse_transform(inv_xtest)
inv_xtest = np.delete(inv_xtest, [8,9,10,11,12], axis =
1)
train_var_name = cols
train_var_name.remove('Conditions')
train_var_name.append('Dates')
train_x = pd.DataFrame(inv_xtest)
PrintTrain_var = train_x
PrintTrain_var['Dates'] = train_dates
PrintTrain_var.columns = [train_var_name]
PrintTrain_var = PrintTrain_var[['Dates',
'Temperature', 'HeatIndex', 'Precipitation',
'WindSpeed', 'WindDirection', 'Visibility',
'CloudCover', 'RelativeHumidity']]
PrintTrain_var.to_csv('TrainParameter.csv', index =
False)

## Create actual data for classification ##
test_var_name = cols
test_x = pd.DataFrame(inv_x)
PrintTest_var = test_x
PrintTest_var['Dates'] = test_dates
PrintTest_var.columns = [test_var_name]
PrintTest_var = PrintTest_var[['Dates',
'Temperature', 'HeatIndex', 'Precipitation',
'WindSpeed', 'WindDirection', 'Visibility',
'CloudCover', 'RelativeHumidity']]
PrintTest_var.to_csv('TestParameter.csv', index = False)

## Create predicted data for classification ##

```

```

pred_var_name = cols[Predict_Var]
pred_y = pd.DataFrame(inv_yhat1)
PrintPred_var = pred_y
PrintPred_var['Date'] = test_dates
PrintPred_var.columns =[pred_var_name, 'Date']
PrintPred_var = PrintPred_var[['Date',pred_var_name]]
PrintPred_var.to_csv(Namefile, index = False)
#Note: Change csv name per variable predicted
(Temperature, HeatIndex, etc)

## Create actual conditions for train classification ##
train_cond = df['Conditions']
train_cond = train_cond[:ntraining]
train_cond = pd.DataFrame(train_cond)
train_cond.reset_index(drop=True, inplace=True)
PrintTrain_cond = train_cond
PrintTrain_cond['Date'] = train_dates
PrintTrain_cond.columns =['Conditions', 'Date']
PrintTrain_cond = PrintTrain_cond[['Date','Conditions']]
PrintTrain_cond.to_csv('TrainConditions.csv', index =
False)

## Create actual conditions for test classification ##
act_cond = df['Conditions']
act_cond = act_cond[(ntraining+1):]
act_cond = pd.DataFrame(act_cond)
act_cond.reset_index(drop=True, inplace=True)
PrintAct_cond = act_cond
PrintAct_cond['Date'] = test_dates
PrintAct_cond.columns =['Conditions', 'Date']
PrintAct_cond = PrintAct_cond[['Date','Conditions']]
PrintAct_cond.to_csv('TestConditions.csv', index =
False)

#####
#####
##Heat Index##
## Reading data ##
Predict_Var = 1 #Choosing variable to do regression
Namefile = 'PredHeatIndex.csv'
# File Name: PredTemperature.csv, PredHeatIndex.csv,
PredPrecipitation.csv,

```



```

# PredWindSpeed.csv, PredWindDirection.csv,
PredVisibility.csv, PredCloudCover.csv,
# PredRelativeHumidity.csv

## Separating dates ##
data_dates = pd.to_datetime(df['Date'])

## Choosing variabel to use ##
cols = list(df)[1:10]
encoder = OneHotEncoder()
df_for_training = df[cols]
values = df_for_training.values
valuesT = values[:,8]
valuesT = valuesT.reshape((valuesT.shape[0], 1))
valuesT = encoder.fit_transform(valuesT).toarray()
values = np.delete(values,8,1)
values = np.append(values, valuesT, axis = 1)
df_for_training = pd.DataFrame(values)

## Normalize the dataset with range 0-1 ##
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(df_for_training)
df_for_training_scaled =
scaler.transform(df_for_training)

##Splitting data to test and training##
trainX = []
trainY = []
testX = []
testY = []
ntraining = ntrain
train = len(df_for_training_scaled[:ntraining,-1])
test = len(df_for_training_scaled[ntraining:,-1])

n_future = 1    # Number of days we want to predict into
the future
n_past = 1      # Number of past days we want to use to
predict the future

##Create time series data ##
for i in range(n_past, len(df_for_training_scaled) -
n_future +1):

```

```

        if i <= len(df_for_training_scaled[:ntraining,-1]):
            trainX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
            trainY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
        else:
            testX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
            testY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
trainX, trainY, testX, testY = np.array(trainX),
np.array(trainY), np.array(testX), np.array(testY)

## Check data shape ##
print('trainX shape == {}'.format(trainX.shape))
print('trainY shape == {}'.format(trainY.shape))
print('testX shape == {}'.format(testX.shape))
print('testY shape == {}'.format(testY.shape))

## Train Dates adjustment ##
train_dates = data_dates[:ntraining]
test_dates = data_dates[(ntraining+1):]
train_dates = pd.DataFrame(train_dates)
test_dates = pd.DataFrame(test_dates)
train_dates.reset_index(drop=True, inplace=True)
test_dates.reset_index(drop=True, inplace=True)

## Create LSTM Model ##
model = Sequential()
model.add(LSTM(64, activation='relu',
input_shape=(trainX.shape[1], trainX.shape[2]),
return_sequences=True))
model.add(LSTM(32, activation='relu',
return_sequences=False))
model.add(Dense(trainY.shape[1]))
model.compile(optimizer='adam', loss='mse')
model.summary()

## Fitting Model ##
history = model.fit(trainX, trainY, epochs=100,
batch_size=nbatch,
validation_data=(testX, testY),

```

```

verbose=1, shuffle=False)
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation
loss')
plt.legend()
plt.show()

## Make prediction based on Model ##
yhat = model.predict(testX)
testX = testX.reshape((testX.shape[0], testX.shape[2]))
yhat = yhat.reshape((yhat.shape[0]))
trainX = trainX.reshape((trainX.shape[0],
trainX.shape[2]))
trainY = trainY.reshape((trainY.shape[0]))

## Revert parameter variable ##
inv_x = testX
inv_x = scaler.inverse_transform(inv_x)
inv_x = np.delete(inv_x, [8,9,10,11,12], axis = 1)

## Revert scaling prediction ##
inv_yhat = testX
inv_yhat[:,Predict_Var] = yhat
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat1 = inv_yhat[:,Predict_Var] #Just taking
predicted variable

## Revert scaling actual data ##
testY = testY.reshape((len(testY)))
inv_y = testX
inv_y[:,Predict_Var] = testY
inv_y = scaler.inverse_transform(inv_y)
inv_y1 = inv_y[:,Predict_Var] #Just taking predicted
variable

## y1 = actual data result, yhat1 = predicted data
result

## calculate MSE ##
mse = mean_squared_error(inv_y1, inv_yhat1)
rmse = np.roots(mse)
print('Test RMSE: %.3f' % rmse)

```

```

plt.plot(inv_y1, label='Training')
plt.plot(inv_yhat1, label='Test')
plt.legend()
plt.savefig('Heat Index.png', dpi=250)
plt.show()
f = open('RMSE.txt','a')
print("Heat Index",mse, file=f)
f.close()

## Create training data for classification ##
inv_xtest = trainX
inv_xtest = scaler.inverse_transform(inv_xtest)
inv_xtest = np.delete(inv_xtest, [8,9,10,11,12], axis =
1)
train_var_name = cols
train_var_name.remove('Conditions')
train_var_name.append('Dates')
train_x = pd.DataFrame(inv_xtest)
PrintTrain_var = train_x
PrintTrain_var['Dates'] = train_dates
PrintTrain_var.columns = [train_var_name]
PrintTrain_var = PrintTrain_var[['Dates',
'Temperature','HeatIndex','Precipitation',
'WindSpeed','WindDirection','Visibility',
'CloudCover','RelativeHumidity']]
PrintTrain_var.to_csv('TrainParameter.csv', index =
False)

## Create actual data for classification ##
test_var_name = cols
test_x = pd.DataFrame(inv_x)
PrintTest_var = test_x
PrintTest_var['Dates'] = test_dates
PrintTest_var.columns = [test_var_name]
PrintTest_var = PrintTest_var[['Dates',
'Temperature','HeatIndex','Precipitation',
'WindSpeed','WindDirection','Visibility',
'CloudCover','RelativeHumidity']]

```

```

PrintTest_var.to_csv('TestParameter.csv', index = False)

## Create predicted data for classification ##
pred_var_name = cols[Predict_Var]
pred_y = pd.DataFrame(inv_yhat1)
PrintPred_var = pred_y
PrintPred_var['Date'] = test_dates
PrintPred_var.columns = [pred_var_name, 'Date']
PrintPred_var = PrintPred_var[['Date', pred_var_name]]
PrintPred_var.to_csv(Namefile, index = False)
#Note: Change csv name per variable predicted
(Temperature, HeatIndex, etc)

## Create actual conditions for train classification ##
train_cond = df['Conditions']
train_cond = train_cond[:ntraining]
train_cond = pd.DataFrame(train_cond)
train_cond.reset_index(drop=True, inplace=True)
PrintTrain_cond = train_cond
PrintTrain_cond['Date'] = train_dates
PrintTrain_cond.columns = ['Conditions', 'Date']
PrintTrain_cond = PrintTrain_cond[['Date', 'Conditions']]
PrintTrain_cond.to_csv('TrainConditions.csv', index =
False)

## Create actual conditions for test classification ##
act_cond = df['Conditions']
act_cond = act_cond[(ntraining+1):]
act_cond = pd.DataFrame(act_cond)
act_cond.reset_index(drop=True, inplace=True)
PrintAct_cond = act_cond
PrintAct_cond['Date'] = test_dates
PrintAct_cond.columns = ['Conditions', 'Date']
PrintAct_cond = PrintAct_cond[['Date', 'Conditions']]
PrintAct_cond.to_csv('TestConditions.csv', index =
False)

#####
#####
##Precipitation##
## Reading data ##
Predict_Var = 2 #Choosing variable to do regression

```

```

Namefile = 'PredPrecipitation.csv'
# File Name: PredTemperature.csv, PredHeatIndex.csv,
PredPrecipitation.csv,
# PredWindSpeed.csv, PredWindDirection.csv,
PredVisibility.csv, PredCloudCover.csv,
# PredRelativeHumidity.csv

## Separating dates ##
data_dates = pd.to_datetime(df['Date'])

## Choosing variabel to use ##
cols = list(df)[1:10]
encoder = OneHotEncoder()
df_for_training = df[cols]
values = df_for_training.values
valuesT = values[:,8]
valuesT = valuesT.reshape((valuesT.shape[0], 1))
valuesT = encoder.fit_transform(valuesT).toarray()
values = np.delete(values,8,1)
values = np.append(values, valuesT, axis = 1)
df_for_training = pd.DataFrame(values)

## Normalize the dataset with range 0-1 ##
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(df_for_training)
df_for_training_scaled =
scaler.transform(df_for_training)

##Splitting data to test and training##
trainX = []
trainY = []
testX = []
testY = []
ntraining = ntrain
train = len(df_for_training_scaled[:ntraining,-1])
test = len(df_for_training_scaled[ntraining:,-1])

n_future = 1    # Number of days we want to predict into
the future
n_past = 1      # Number of past days we want to use to
predict the future

```

```

##Create time series data ##
for i in range(n_past, len(df_for_training_scaled) -
n_future + 1):
    if i <= len(df_for_training_scaled[:ntraining,-1]):
        trainX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
        trainY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
    else:
        testX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
        testY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
trainX, trainY, testX, testY = np.array(trainX),
np.array(trainY), np.array(testX), np.array(testY)

## Check data shape ##
print('trainX shape == {}'.format(trainX.shape))
print('trainY shape == {}'.format(trainY.shape))
print('testX shape == {}'.format(testX.shape))
print('testY shape == {}'.format(testY.shape))

## Train Dates adjustment ##
train_dates = data_dates[:ntraining]
test_dates = data_dates[(ntraining+1):]
train_dates = pd.DataFrame(train_dates)
test_dates = pd.DataFrame(test_dates)
train_dates.reset_index(drop=True, inplace=True)
test_dates.reset_index(drop=True, inplace=True)

## Create LSTM Model ##
model = Sequential()
model.add(LSTM(64, activation='relu',
input_shape=(trainX.shape[1], trainX.shape[2]),
return_sequences=True))
model.add(LSTM(32, activation='relu',
return_sequences=False))
model.add(Dense(trainY.shape[1]))
model.compile(optimizer='adam', loss='mse')
model.summary()

## Fitting Model ##

```

```

history = model.fit(trainX, trainY, epochs=100,
                    batch_size=nbatch,
                    validation_data=(testX, testY),
                    verbose=1, shuffle=False)
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.legend()
plt.show()

## Make prediction based on Model ##
yhat = model.predict(testX)
testX = testX.reshape((testX.shape[0], testX.shape[2]))
yhat = yhat.reshape((yhat.shape[0]))
trainX = trainX.reshape((trainX.shape[0],
                        trainX.shape[2]))
trainY = trainY.reshape((trainY.shape[0]))

## Revert parameter variable ##
inv_x = testX
inv_x = scaler.inverse_transform(inv_x)
inv_x = np.delete(inv_x, [8,9,10,11,12], axis = 1)

## Revert scaling prediction ##
inv_yhat = testX
inv_yhat[:,Predict_Var] = yhat
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat1 = inv_yhat[:,Predict_Var] #Just taking
predicted variable

## Revert scaling actual data ##
testY = testY.reshape((len(testY)))
inv_y = testX
inv_y[:,Predict_Var] = testY
inv_y = scaler.inverse_transform(inv_y)
inv_y1 = inv_y[:,Predict_Var] #Just taking predicted
variable

## y1 = actual data result, yhat1 = predicted data
result

## calculate MSE ##

```



```

mse = mean_squared_error(inv_y1, inv_yhat1)
rmse = np.roots(mse)
print('Test RMSE: %.3f' % rmse)
plt.plot(inv_y1, label='Training')
plt.plot(inv_yhat1, label='Test')
plt.legend()
plt.savefig('Precipitation.png', dpi=250)
plt.show()
f = open('RMSE.txt','a')
print("Precipitation",mse, file=f)
f.close()

## Create training data for classification ##
inv_xtest = trainX
inv_xtest = scaler.inverse_transform(inv_xtest)
inv_xtest = np.delete(inv_xtest, [8,9,10,11,12], axis =
1)
train_var_name = cols
train_var_name.remove('Conditions')
train_var_name.append('Dates')
train_x = pd.DataFrame(inv_xtest)
PrintTrain_var = train_x
PrintTrain_var['Dates'] = train_dates
PrintTrain_var.columns = [train_var_name]
PrintTrain_var = PrintTrain_var[['Dates',
'Temperature', 'HeatIndex', 'Precipitation',

'WindSpeed', 'WindDirection', 'Visibility',

'CloudCover', 'RelativeHumidity']]
PrintTrain_var.to_csv('TrainParameter.csv', index =
False)

## Create actual data for classification ##
test_var_name = cols
test_x = pd.DataFrame(inv_x)
PrintTest_var = test_x
PrintTest_var['Dates'] = test_dates
PrintTest_var.columns = [test_var_name]
PrintTest_var = PrintTest_var[['Dates',
'Temperature', 'HeatIndex', 'Precipitation',

```

```

'WindSpeed','WindDirection','Visibility',
'CloudCover','RelativeHumidity']]
PrintTest_var.to_csv('TestParameter.csv', index = False)

## Create predicted data for classification ##
pred_var_name = cols[Predict_Var]
pred_y = pd.DataFrame(inv_yhat1)
PrintPred_var = pred_y
PrintPred_var['Date'] = test_dates
PrintPred_var.columns = [pred_var_name, 'Date']
PrintPred_var = PrintPred_var[['Date', pred_var_name]]
PrintPred_var.to_csv(Namefile, index = False)
#Note: Change csv name per variable predicted
(Temperature, HeatIndex, etc)

## Create actual conditions for train classification ##
train_cond = df['Conditions']
train_cond = train_cond[:ntraining]
train_cond = pd.DataFrame(train_cond)
train_cond.reset_index(drop=True, inplace=True)
PrintTrain_cond = train_cond
PrintTrain_cond['Date'] = train_dates
PrintTrain_cond.columns = ['Conditions', 'Date']
PrintTrain_cond = PrintTrain_cond[['Date', 'Conditions']]
PrintTrain_cond.to_csv('TrainConditions.csv', index =
False)

## Create actual conditions for test classification ##
act_cond = df['Conditions']
act_cond = act_cond[(ntraining+1):]
act_cond = pd.DataFrame(act_cond)
act_cond.reset_index(drop=True, inplace=True)
PrintAct_cond = act_cond
PrintAct_cond['Date'] = test_dates
PrintAct_cond.columns = ['Conditions', 'Date']
PrintAct_cond = PrintAct_cond[['Date', 'Conditions']]
PrintAct_cond.to_csv('TestConditions.csv', index =
False)

#####
#####

```

```

##Wind Speed##
## Reading data ##
Predict_Var = 3 #Choosing variable to do regression
Namefile = 'PredWindSpeed.csv'
# File Name: PredTemperature.csv, PredHeatIndex.csv,
PredPrecipitation.csv,
# PredWindSpeed.csv, PredWindDirection.csv,
PredVisibility.csv, PredCloudCover.csv,
# PredRelativeHumidity.csv

## Separating dates ##
data_dates = pd.to_datetime(df['Date'])

## Choosing variabel to use ##
cols = list(df)[1:10]
encoder = OneHotEncoder()
df_for_training = df[cols]
values = df_for_training.values
valuesT = values[:,8]
valuesT = valuesT.reshape((valuesT.shape[0], 1))
valuesT = encoder.fit_transform(valuesT).toarray()
values = np.delete(values,8,1)
values = np.append(values, valuesT, axis = 1)
df_for_training = pd.DataFrame(values)

## Normalize the dataset with range 0-1 ##
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(df_for_training)
df_for_training_scaled =
scaler.transform(df_for_training)

##Splitting data to test and training##
trainX = []
trainY = []
testX = []
testY = []
ntraining = ntrain
train = len(df_for_training_scaled[:ntraining,-1])
test = len(df_for_training_scaled[ntraining:,-1])

n_future = 1 # Number of days we want to predict into
the future

```

```

n_past = 1      # Number of past days we want to use to
predict the future

##Create time series data ##
for i in range(n_past, len(df_for_training_scaled) -
n_future +1):
    if i <= len(df_for_training_scaled[:ntraining,-1]):
        trainX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
        trainY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
    else:
        testX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
        testY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
trainX, trainY, testX, testY = np.array(trainX),
np.array(trainY), np.array(testX), np.array(testY)

## Check data shape ##
print('trainX shape == {}'.format(trainX.shape))
print('trainY shape == {}'.format(trainY.shape))
print('testX shape == {}'.format(testX.shape))
print('testY shape == {}'.format(testY.shape))

## Train Dates adjustment ##
train_dates = data_dates[:ntraining]
test_dates = data_dates[(ntraining+1):]
train_dates = pd.DataFrame(train_dates)
test_dates = pd.DataFrame(test_dates)
train_dates.reset_index(drop=True, inplace=True)
test_dates.reset_index(drop=True, inplace=True)

## Create LSTM Model ##
model = Sequential()
model.add(LSTM(64, activation='relu',
input_shape=(trainX.shape[1], trainX.shape[2]),
return_sequences=True))
model.add(LSTM(32, activation='relu',
return_sequences=False))
model.add(Dense(trainY.shape[1]))
model.compile(optimizer='adam', loss='mse')

```

```

model.summary()

## Fitting Model ##
history = model.fit(trainX, trainY, epochs=100,
                    batch_size=nbatch,
                    validation_data=(testX, testY),
                    verbose=1, shuffle=False)
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation
loss')
plt.legend()
plt.show()

## Make prediction based on Model ##
yhat = model.predict(testX)
testX = testX.reshape((testX.shape[0], testX.shape[2]))
yhat = yhat.reshape((yhat.shape[0]))
trainX = trainX.reshape((trainX.shape[0],
trainX.shape[2]))
trainY = trainY.reshape((trainY.shape[0]))

## Revert parameter variable ##
inv_x = testX
inv_x = scaler.inverse_transform(inv_x)
inv_x = np.delete(inv_x, [8,9,10,11,12], axis = 1)

## Revert scaling prediction ##
inv_yhat = testX
inv_yhat[:,Predict_Var] = yhat
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat1 = inv_yhat[:,Predict_Var] #Just taking
predicted variable

## Revert scaling actual data ##
testY = testY.reshape((len(testY)))
inv_y = testX
inv_y[:,Predict_Var] = testY
inv_y = scaler.inverse_transform(inv_y)
inv_y1 = inv_y[:,Predict_Var] #Just taking predicted
variable

## y1 = actual data result, yhat1 = predicted data

```

```

result

## calculate MSE ##
mse = mean_squared_error(inv_y1, inv_yhat1)
rmse = np.roots(mse)
print('Test RMSE: %.3f' % rmse)
plt.plot(inv_y1, label='Training')
plt.plot(inv_yhat1, label='Test')
plt.legend()
plt.savefig('Wind Speed.png', dpi=250)
plt.show()
f = open('RMSE.txt', 'a')
print("Wind Speed", mse, file=f)
f.close()

## Create training data for classification ##
inv_x_test = trainX
inv_x_test = scaler.inverse_transform(inv_x_test)
inv_x_test = np.delete(inv_x_test, [8,9,10,11,12], axis =
1)
train_var_name = cols
train_var_name.remove('Conditions')
train_var_name.append('Dates')
train_x = pd.DataFrame(inv_x_test)
PrintTrain_var = train_x
PrintTrain_var['Dates'] = train_dates
PrintTrain_var.columns = [train_var_name]
PrintTrain_var = PrintTrain_var[['Dates',
'Temperature', 'HeatIndex', 'Precipitation',

'WindSpeed', 'WindDirection', 'Visibility',

'CloudCover', 'RelativeHumidity']]
PrintTrain_var.to_csv('TrainParameter.csv', index =
False)

## Create actual data for classification ##
test_var_name = cols
test_x = pd.DataFrame(inv_x)
PrintTest_var = test_x
PrintTest_var['Dates'] = test_dates
PrintTest_var.columns = [test_var_name]

```

```

PrintTest_var = PrintTest_var[['Dates',
'Temperature','HeatIndex','Precipitation',

'WindSpeed','WindDirection','Visibility',

'CloudCover','RelativeHumidity']]
PrintTest_var.to_csv('TestParameter.csv', index = False)

## Create predicted data for classification ##
pred_var_name = cols[Predict_Var]
pred_y = pd.DataFrame(inv_yhat1)
PrintPred_var = pred_y
PrintPred_var['Date'] = test_dates
PrintPred_var.columns =[pred_var_name, 'Date']
PrintPred_var = PrintPred_var[['Date',pred_var_name]]
PrintPred_var.to_csv(Namefile, index = False)
#Note: Change csv name per variable predicted
(Temperature, HeatIndex, etc)

## Create actual conditions for train classification ##
train_cond = df['Conditions']
train_cond = train_cond[:ntraining]
train_cond = pd.DataFrame(train_cond)
train_cond.reset_index(drop=True, inplace=True)
PrintTrain_cond = train_cond
PrintTrain_cond['Date'] = train_dates
PrintTrain_cond.columns =['Conditions', 'Date']
PrintTrain_cond = PrintTrain_cond[['Date','Conditions']]
PrintTrain_cond.to_csv('TrainConditions.csv', index =
False)

## Create actual conditions for test classification ##
act_cond = df['Conditions']
act_cond = act_cond[(ntraining+1):]
act_cond = pd.DataFrame(act_cond)
act_cond.reset_index(drop=True, inplace=True)
PrintAct_cond = act_cond
PrintAct_cond['Date'] = test_dates
PrintAct_cond.columns =['Conditions', 'Date']
PrintAct_cond = PrintAct_cond[['Date','Conditions']]
PrintAct_cond.to_csv('TestConditions.csv', index =
False)

```

```
#####
#####
##Wind Direction##
## Reading data ##
Predict_Var = 4 #Choosing variable to do regression
Namefile = 'PredWindDirection.csv'
# File Name: PredTemperature.csv, PredHeatIndex.csv,
PredPrecipitation.csv,
# PredWindSpeed.csv, PredWindDirection.csv,
PredVisibility.csv, PredCloudCover.csv,
# PredRelativeHumidity.csv

## Separating dates ##
data_dates = pd.to_datetime(df['Date'])

## Choosing variabel to use ##
cols = list(df)[1:10]
encoder = OneHotEncoder()
df_for_training = df[cols]
values = df_for_training.values
valuesT = values[:,8]
valuesT = valuesT.reshape((valuesT.shape[0], 1))
valuesT = encoder.fit_transform(valuesT).toarray()
values = np.delete(values,8,1)
values = np.append(values, valuesT, axis = 1)
df_for_training = pd.DataFrame(values)

## Normalize the dataset with range 0-1 ##
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(df_for_training)
df_for_training_scaled =
scaler.transform(df_for_training)

##Splitting data to test and training##
trainX = []
trainY = []
testX = []
testY = []
ntraining = ntrain
train = len(df_for_training_scaled[:ntraining,-1])
test = len(df_for_training_scaled[ntraining:-1])
```



```

n_future = 1    # Number of days we want to predict into
the future
n_past = 1      # Number of past days we want to use to
predict the future

##Create time series data ##
for i in range(n_past, len(df_for_training_scaled) -
n_future + 1):
    if i <= len(df_for_training_scaled[:ntraining,-1]):
        trainX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
        trainY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
    else:
        testX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
        testY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
trainX, trainY, testX, testY = np.array(trainX),
np.array(trainY), np.array(testX), np.array(testY)

## Check data shape ##
print('trainX shape == {}'.format(trainX.shape))
print('trainY shape == {}'.format(trainY.shape))
print('testX shape == {}'.format(testX.shape))
print('testY shape == {}'.format(testY.shape))

## Train Dates adjustment ##
train_dates = data_dates[:ntraining]
test_dates = data_dates[(ntraining+1):]
train_dates = pd.DataFrame(train_dates)
test_dates = pd.DataFrame(test_dates)
train_dates.reset_index(drop=True, inplace=True)
test_dates.reset_index(drop=True, inplace=True)

## Create LSTM Model ##
model = Sequential()
model.add(LSTM(64, activation='relu',
input_shape=(trainX.shape[1], trainX.shape[2]),
return_sequences=True))
model.add(LSTM(32, activation='relu',

```

```

return_sequences=False))
model.add(Dense(trainY.shape[1]))
model.compile(optimizer='adam', loss='mse')
model.summary()

## Fitting Model ##
history = model.fit(trainX, trainY, epochs=100,
batch_size=batch_size,
                    validation_data=(testX, testY),
verbose=1, shuffle=False)
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation
loss')
plt.legend()
plt.show()

## Make prediction based on Model ##
yhat = model.predict(testX)
testX = testX.reshape((testX.shape[0], testX.shape[2]))
yhat = yhat.reshape((yhat.shape[0]))
trainX = trainX.reshape((trainX.shape[0],
trainX.shape[2]))
trainY = trainY.reshape((trainY.shape[0]))

## Revert parameter variable ##
inv_x = testX
inv_x = scaler.inverse_transform(inv_x)
inv_x = np.delete(inv_x, [8,9,10,11,12], axis = 1)

## Revert scaling prediction ##
inv_yhat = testX
inv_yhat[:,Predict_Var] = yhat
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat1 = inv_yhat[:,Predict_Var] #Just taking
predicted variable

## Revert scaling actual data ##
testY = testY.reshape((len(testY)))
inv_y = testX
inv_y[:,Predict_Var] = testY
inv_y = scaler.inverse_transform(inv_y)
inv_y1 = inv_y[:,Predict_Var] #Just taking predicted

```

```

variable

## y1 = actual data result, yhat1 = predicted data
result

## calculate MSE ##
mse = mean_squared_error(inv_y1, inv_yhat1)
rmse = np.roots(mse)
print('Test RMSE: %.3f' % rmse)
plt.plot(inv_y1, label='Training')
plt.plot(inv_yhat1, label='Test')
plt.legend()
plt.savefig('Wind Direction.png', dpi=250)
plt.show()
f = open('RMSE.txt', 'a')
print("Wind Direction", mse, file=f)
f.close()

## Create training data for classification ##
inv_xtest = trainX
inv_xtest = scaler.inverse_transform(inv_xtest)
inv_xtest = np.delete(inv_xtest, [8,9,10,11,12], axis =
1)
train_var_name = cols
train_var_name.remove('Conditions')
train_var_name.append('Dates')
train_x = pd.DataFrame(inv_xtest)
PrintTrain_var = train_x
PrintTrain_var['Dates'] = train_dates
PrintTrain_var.columns = [train_var_name]
PrintTrain_var = PrintTrain_var[['Dates',
'Temperature', 'HeatIndex', 'Precipitation',

'WindSpeed', 'WindDirection', 'Visibility',

'CloudCover', 'RelativeHumidity']]
PrintTrain_var.to_csv('TrainParameter.csv', index =
False)

## Create actual data for classification ##
test_var_name = cols
test_x = pd.DataFrame(inv_x)

```

```

PrintTest_var = test_x
PrintTest_var['Dates'] = test_dates
PrintTest_var.columns = [test_var_name]
PrintTest_var = PrintTest_var[['Dates',
'Temperature', 'HeatIndex', 'Precipitation',

'WindSpeed', 'WindDirection', 'Visibility',

'CloudCover', 'RelativeHumidity']]
PrintTest_var.to_csv('TestParameter.csv', index = False)

## Create predicted data for classification ##
pred_var_name = cols[Predict_Var]
pred_y = pd.DataFrame(inv_yhat1)
PrintPred_var = pred_y
PrintPred_var['Date'] = test_dates
PrintPred_var.columns = [pred_var_name, 'Date']
PrintPred_var = PrintPred_var[['Date', pred_var_name]]
PrintPred_var.to_csv(Namefile, index = False)
#Note: Change csv name per variable predicted
(Temperature, HeatIndex, etc)

## Create actual conditions for train classification ##
train_cond = df['Conditions']
train_cond = train_cond[:ntraining]
train_cond = pd.DataFrame(train_cond)
train_cond.reset_index(drop=True, inplace=True)
PrintTrain_cond = train_cond
PrintTrain_cond['Date'] = train_dates
PrintTrain_cond.columns = ['Conditions', 'Date']
PrintTrain_cond = PrintTrain_cond[['Date', 'Conditions']]
PrintTrain_cond.to_csv('TrainConditions.csv', index =
False)

## Create actual conditions for test classification ##
act_cond = df['Conditions']
act_cond = act_cond[(ntraining+1):]
act_cond = pd.DataFrame(act_cond)
act_cond.reset_index(drop=True, inplace=True)
PrintAct_cond = act_cond
PrintAct_cond['Date'] = test_dates
PrintAct_cond.columns = ['Conditions', 'Date']

```

```

PrintAct_cond = PrintAct_cond[['Date','Conditions']]
PrintAct_cond.to_csv('TestConditions.csv', index =
False)

#####
#####
##Visibility##
## Reading data ##
Predict_Var = 5 #Choosing variable to do regression
Namefile = 'PredVisibility.csv'
# File Name: PredTemperature.csv, PredHeatIndex.csv,
PredPrecipitation.csv,
# PredWindSpeed.csv, PredWindDirection.csv,
PredVisibility.csv, PredCloudCover.csv,
# PredRelativeHumidity.csv

## Separating dates ##
data_dates = pd.to_datetime(df['Date'])

## Choosing variabel to use ##
cols = list(df)[1:10]
encoder = OneHotEncoder()
df_for_training = df[cols]
values = df_for_training.values
valuesT = values[:,8]
valuesT = valuesT.reshape((valuesT.shape[0], 1))
valuesT = encoder.fit_transform(valuesT).toarray()
values = np.delete(values,8,1)
values = np.append(values, valuesT, axis = 1)
df_for_training = pd.DataFrame(values)

## Normalize the dataset with range 0-1 ##
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(df_for_training)
df_for_training_scaled =
scaler.transform(df_for_training)

##Splitting data to test and training##
trainX = []
trainY = []
testX = []
testY = []

```

```

ntraining = ntrain
train = len(df_for_training_scaled[:ntraining,-1])
test = len(df_for_training_scaled[ntraining:-1])

n_future = 1    # Number of days we want to predict into
the future
n_past = 1      # Number of past days we want to use to
predict the future

##Create time series data ##
for i in range(n_past, len(df_for_training_scaled) -
n_future +1):
    if i <= len(df_for_training_scaled[:ntraining,-1]):
        trainX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
        trainY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
    else:
        testX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
        testY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
trainX, trainY, testX, testY = np.array(trainX),
np.array(trainY), np.array(testX), np.array(testY)

## Check data shape ##
print('trainX shape == {}'.format(trainX.shape))
print('trainY shape == {}'.format(trainY.shape))
print('testX shape == {}'.format(testX.shape))
print('testY shape == {}'.format(testY.shape))

## Train Dates adjustment ##
train_dates = data_dates[:ntraining]
test_dates = data_dates[(ntraining+1):]
train_dates = pd.DataFrame(train_dates)
test_dates = pd.DataFrame(test_dates)
train_dates.reset_index(drop=True, inplace=True)
test_dates.reset_index(drop=True, inplace=True)

## Create LSTM Model ##
model = Sequential()
model.add(LSTM(64, activation='relu',

```

```

input_shape=(trainX.shape[1], trainX.shape[2]),
              return_sequences=True))
model.add(LSTM(32, activation='relu',
              return_sequences=False))
model.add(Dense(trainY.shape[1]))
model.compile(optimizer='adam', loss='mse')
model.summary()

## Fitting Model ##
history = model.fit(trainX, trainY, epochs=100,
                    batch_size=batch_size,
                    validation_data=(testX, testY),
                    verbose=1, shuffle=False)
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation
loss')
plt.legend()
plt.show()

## Make prediction based on Model ##
yhat = model.predict(testX)
testX = testX.reshape((testX.shape[0], testX.shape[2]))
yhat = yhat.reshape((yhat.shape[0]))
trainX = trainX.reshape((trainX.shape[0],
trainX.shape[2]))
trainY = trainY.reshape((trainY.shape[0]))

## Revert parameter variable ##
inv_x = testX
inv_x = scaler.inverse_transform(inv_x)
inv_x = np.delete(inv_x, [8,9,10,11,12], axis = 1)

## Revert scaling prediction ##
inv_yhat = testX
inv_yhat[:,Predict_Var] = yhat
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat1 = inv_yhat[:,Predict_Var] #Just taking
predicted variable

## Revert scaling actual data ##
testY = testY.reshape((len(testY)))
inv_y = testX

```

```

inv_y[:,Predict_Var] = testY
inv_y = scaler.inverse_transform(inv_y)
inv_y1 = inv_y[:,Predict_Var] #Just taking predicted
variable

## y1 = actual data result, yhat1 = predicted data
result

## calculate MSE ##
mse = mean_squared_error(inv_y1, inv_yhat1)
rmse = np.roots(mse)
print('Test RMSE: %.3f' % rmse)
plt.plot(inv_y1, label='Training')
plt.plot(inv_yhat1, label='Test')
plt.legend()
plt.savefig('Visibility.png', dpi=250)
plt.show()
f = open('RMSE.txt','a')
print("Visibility",mse, file=f)
f.close()

## Create training data for classification ##
inv_xtest = trainX
inv_xtest = scaler.inverse_transform(inv_xtest)
inv_xtest = np.delete(inv_xtest, [8,9,10,11,12], axis =
1)
train_var_name = cols
train_var_name.remove('Conditions')
train_var_name.append('Dates')
train_x = pd.DataFrame(inv_xtest)
PrintTrain_var = train_x
PrintTrain_var['Dates'] = train_dates
PrintTrain_var.columns = [train_var_name]
PrintTrain_var = PrintTrain_var[['Dates',
'Temperature','HeatIndex','Precipitation',

'WindSpeed','WindDirection','Visibility',

'CloudCover','RelativeHumidity']]
PrintTrain_var.to_csv('TrainParameter.csv', index =
False)

```



```

## Create actual data for classification ##
test_var_name = cols
test_x = pd.DataFrame(inv_x)
PrintTest_var = test_x
PrintTest_var['Dates'] = test_dates
PrintTest_var.columns = [test_var_name]
PrintTest_var = PrintTest_var[['Dates',
'Temperature','HeatIndex','Precipitation',

'WindSpeed','WindDirection','Visibility',

'CloudCover','RelativeHumidity']]
PrintTest_var.to_csv('TestParameter.csv', index = False)

## Create predicted data for classification ##
pred_var_name = cols[Predict_Var]
pred_y = pd.DataFrame(inv_yhat1)
PrintPred_var = pred_y
PrintPred_var['Date'] = test_dates
PrintPred_var.columns =[pred_var_name, 'Date']
PrintPred_var = PrintPred_var[['Date',pred_var_name]]
PrintPred_var.to_csv(Namefile, index = False)
#Note: Change csv name per variable predicted
(Temperature, HeatIndex, etc)

## Create actual conditions for train classification ##
train_cond = df['Conditions']
train_cond = train_cond[:ntraining]
train_cond = pd.DataFrame(train_cond)
train_cond.reset_index(drop=True, inplace=True)
PrintTrain_cond = train_cond
PrintTrain_cond['Date'] = train_dates
PrintTrain_cond.columns =['Conditions', 'Date']
PrintTrain_cond = PrintTrain_cond[['Date','Conditions']]
PrintTrain_cond.to_csv('TrainConditions.csv', index =
False)

## Create actual conditions for test classification ##
act_cond = df['Conditions']
act_cond = act_cond[(ntraining+1):]
act_cond = pd.DataFrame(act_cond)
act_cond.reset_index(drop=True, inplace=True)

```

```

PrintAct_cond = act_cond
PrintAct_cond['Date'] = test_dates
PrintAct_cond.columns = ['Conditions', 'Date']
PrintAct_cond = PrintAct_cond[['Date', 'Conditions']]
PrintAct_cond.to_csv('TestConditions.csv', index =
False)

#####
#####
##Cloud Cover##
## Reading data ##
Predict_Var = 6 #Choosing variable to do regression
Namefile = 'PredCloudCover.csv'
# File Name: PredTemperature.csv, PredHeatIndex.csv,
PredPrecipitation.csv,
# PredWindSpeed.csv, PredWindDirection.csv,
PredVisibility.csv, PredCloudCover.csv,
# PredRelativeHumidity.csv

## Separating dates ##
data_dates = pd.to_datetime(df['Date'])

## Choosing variabel to use ##
cols = list(df)[1:10]
encoder = OneHotEncoder()
df_for_training = df[cols]
values = df_for_training.values
valuesT = values[:,8]
valuesT = valuesT.reshape((valuesT.shape[0], 1))
valuesT = encoder.fit_transform(valuesT).toarray()
values = np.delete(values,8,1)
values = np.append(values, valuesT, axis = 1)
df_for_training = pd.DataFrame(values)

## Normalize the dataset with range 0-1 ##
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(df_for_training)
df_for_training_scaled =
scaler.transform(df_for_training)

##Splitting data to test and training##
trainX = []

```

```

trainY = []
testX = []
testY = []
ntraining = ntrain
train = len(df_for_training_scaled[:ntraining,-1])
test = len(df_for_training_scaled[ntraining:-1])

n_future = 1    # Number of days we want to predict into
the future
n_past = 1      # Number of past days we want to use to
predict the future

##Create time series data ##
for i in range(n_past, len(df_for_training_scaled) -
n_future + 1):
    if i <= len(df_for_training_scaled[:ntraining,-1]):
        trainX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
        trainY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
    else:
        testX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
        testY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
trainX, trainY, testX, testY = np.array(trainX),
np.array(trainY), np.array(testX), np.array(testY)

## Check data shape ##
print('trainX shape == {}'.format(trainX.shape))
print('trainY shape == {}'.format(trainY.shape))
print('testX shape == {}'.format(testX.shape))
print('testY shape == {}'.format(testY.shape))

## Train Dates adjustment ##
train_dates = data_dates[:ntraining]
test_dates = data_dates[(ntraining+1):]
train_dates = pd.DataFrame(train_dates)
test_dates = pd.DataFrame(test_dates)
train_dates.reset_index(drop=True, inplace=True)
test_dates.reset_index(drop=True, inplace=True)

```

```

## Create LSTM Model ##
model = Sequential()
model.add(LSTM(64, activation='relu',
input_shape=(trainX.shape[1], trainX.shape[2]),
              return_sequences=True))
model.add(LSTM(32, activation='relu',
return_sequences=False))
model.add(Dense(trainY.shape[1]))
model.compile(optimizer='adam', loss='mse')
model.summary()

## Fitting Model ##
history = model.fit(trainX, trainY, epochs=100,
batch_size=nbatch,
                    validation_data=(testX, testY),
verbose=1, shuffle=False)
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation
loss')
plt.legend()
plt.show()

## Make prediction based on Model ##
yhat = model.predict(testX)
testX = testX.reshape((testX.shape[0], testX.shape[2]))
yhat = yhat.reshape((yhat.shape[0]))
trainX = trainX.reshape((trainX.shape[0],
trainX.shape[2]))
trainY = trainY.reshape((trainY.shape[0]))

## Revert parameter variable ##
inv_x = testX
inv_x = scaler.inverse_transform(inv_x)
inv_x = np.delete(inv_x, [8,9,10,11,12], axis = 1)

## Revert scaling prediction ##
inv_yhat = testX
inv_yhat[:,Predict_Var] = yhat
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat1 = inv_yhat[:,Predict_Var] #Just taking
predicted variable

```

```

## Revert scalling actual data ##
testY = testY.reshape((len(testY)))
inv_y = testX
inv_y[:,Predict_Var] = testY
inv_y = scaler.inverse_transform(inv_y)
inv_y1 = inv_y[:,Predict_Var] #Just taking predicted
variable

## y1 = actual data result, yhat1 = predicted data
result

## calculate MSE ##
mse = mean_squared_error(inv_y1, inv_yhat1)
rmse = np.roots(mse)
print('Test RMSE: %.3f' % rmse)
plt.plot(inv_y1, label='Training')
plt.plot(inv_yhat1, label='Test')
plt.legend()
plt.savefig('Cloud Cover.png', dpi=250)
plt.show()
f = open('RMSE.txt','a')
print("Cloud Cover",mse, file=f)
f.close()

## Create training data for classification ##
inv_xtest = trainX
inv_xtest = scaler.inverse_transform(inv_xtest)
inv_xtest = np.delete(inv_xtest, [8,9,10,11,12], axis =
1)
train_var_name = cols
train_var_name.remove('Conditions')
train_var_name.append('Dates')
train_x = pd.DataFrame(inv_xtest)
PrintTrain_var = train_x
PrintTrain_var['Dates'] = train_dates
PrintTrain_var.columns = [train_var_name]
PrintTrain_var = PrintTrain_var[['Dates',
'Temperature','HeatIndex','Precipitation',
'WindSpeed','WindDirection','Visibility',
'CloudCover','RelativeHumidity']]

```

```

PrintTrain_var.to_csv('TrainParameter.csv', index =
False)

## Create actual data for classification ##
test_var_name = cols
test_x = pd.DataFrame(inv_x)
PrintTest_var = test_x
PrintTest_var['Dates'] = test_dates
PrintTest_var.columns = [test_var_name]
PrintTest_var = PrintTest_var[['Dates',
'Temperature', 'HeatIndex', 'Precipitation',

'WindSpeed', 'WindDirection', 'Visibility',

'CloudCover', 'RelativeHumidity']]
PrintTest_var.to_csv('TestParameter.csv', index = False)

## Create predicted data for classification ##
pred_var_name = cols[Predict_Var]
pred_y = pd.DataFrame(inv_yhat1)
PrintPred_var = pred_y
PrintPred_var['Date'] = test_dates
PrintPred_var.columns = [pred_var_name, 'Date']
PrintPred_var = PrintPred_var[['Date', pred_var_name]]
PrintPred_var.to_csv(Namefile, index = False)
#Note: Change csv name per variable predicted
(Temperature, HeatIndex, etc)

## Create actual conditions for train classification ##
train_cond = df['Conditions']
train_cond = train_cond[:ntraining]
train_cond = pd.DataFrame(train_cond)
train_cond.reset_index(drop=True, inplace=True)
PrintTrain_cond = train_cond
PrintTrain_cond['Date'] = train_dates
PrintTrain_cond.columns = ['Conditions', 'Date']
PrintTrain_cond = PrintTrain_cond[['Date', 'Conditions']]
PrintTrain_cond.to_csv('TrainConditions.csv', index =
False)

## Create actual conditions for test classification ##
act_cond = df['Conditions']

```

```

act_cond = act_cond[(ntraining+1):]
act_cond = pd.DataFrame(act_cond)
act_cond.reset_index(drop=True, inplace=True)
PrintAct_cond = act_cond
PrintAct_cond['Date'] = test_dates
PrintAct_cond.columns = ['Conditions', 'Date']
PrintAct_cond = PrintAct_cond[['Date', 'Conditions']]
PrintAct_cond.to_csv('TestConditions.csv', index =
False)

#####
#####
##Relative Humidity##
## Reading data ##
Predict_Var = 7 #Choosing variable to do regression
Namefile = 'PredRelativeHumidity.csv'
# File Name: PredTemperature.csv, PredHeatIndex.csv,
PredPrecipitation.csv,
# PredWindSpeed.csv, PredWindDirection.csv,
PredVisibility.csv, PredCloudCover.csv,
# PredRelativeHumidity.csv

## Separating dates ##
data_dates = pd.to_datetime(df['Date'])

## Choosing variabel to use ##
cols = list(df)[1:10]
encoder = OneHotEncoder()
df_for_training = df[cols]
values = df_for_training.values
valuesT = values[:,8]
valuesT = valuesT.reshape((valuesT.shape[0], 1))
valuesT = encoder.fit_transform(valuesT).toarray()
values = np.delete(values,8,1)
values = np.append(values, valuesT, axis = 1)
df_for_training = pd.DataFrame(values)

## Normalize the dataset with range 0-1 ##
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(df_for_training)
df_for_training_scaled =
scaler.transform(df_for_training)

```

```

##Splitting data to test and training##
trainX = []
trainY = []
testX = []
testY = []
ntraining = ntrain
train = len(df_for_training_scaled[:ntraining,-1])
test = len(df_for_training_scaled[ntraining:,-1])

n_future = 1    # Number of days we want to predict into
the future
n_past = 1      # Number of past days we want to use to
predict the future

##Create time series data ##
for i in range(n_past, len(df_for_training_scaled) -
n_future +1):
    if i <= len(df_for_training_scaled[:ntraining,-1]):
        trainX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
        trainY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
    else:
        testX.append(df_for_training_scaled[i -
n_past:i, 0:df_for_training.shape[1]])
        testY.append(df_for_training_scaled[i +
n_future - 1:i + n_future, Predict_Var])
trainX, trainY, testX, testY = np.array(trainX),
np.array(trainY), np.array(testX), np.array(testY)

## Check  data shape ##
print('trainX shape == {}'.format(trainX.shape))
print('trainY shape == {}'.format(trainY.shape))
print('testX shape == {}'.format(testX.shape))
print('testY shape == {}'.format(testY.shape))

## Train Dates adjustment ##
train_dates = data_dates[:ntraining]
test_dates = data_dates[(ntraining+1):]
train_dates = pd.DataFrame(train_dates)
test_dates = pd.DataFrame(test_dates)

```



```

train_dates.reset_index(drop=True, inplace=True)
test_dates.reset_index(drop=True, inplace=True)

## Create LSTM Model ##
model = Sequential()
model.add(LSTM(64, activation='relu',
              input_shape=(trainX.shape[1], trainX.shape[2]),
              return_sequences=True))
model.add(LSTM(32, activation='relu',
              return_sequences=False))
model.add(Dense(trainY.shape[1]))
model.compile(optimizer='adam', loss='mse')
model.summary()

## Fitting Model ##
history = model.fit(trainX, trainY, epochs=100,
                    batch_size=nbatch,
                    validation_data=(testX, testY),
                    verbose=1, shuffle=False)
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation
loss')
plt.legend()
plt.show()

## Make prediction based on Model ##
yhat = model.predict(testX)
testX = testX.reshape((testX.shape[0], testX.shape[2]))
yhat = yhat.reshape((yhat.shape[0]))
trainX = trainX.reshape((trainX.shape[0],
trainX.shape[2]))
trainY = trainY.reshape((trainY.shape[0]))

## Revert parameter variable ##
inv_x = testX
inv_x = scaler.inverse_transform(inv_x)
inv_x = np.delete(inv_x, [8,9,10,11,12], axis = 1)

## Revert scaling prediction ##
inv_yhat = testX
inv_yhat[:, Predict_Var] = yhat
inv_yhat = scaler.inverse_transform(inv_yhat)

```

```

inv_yhat1 = inv_yhat[:,Predict_Var] #Just taking
predicted variable

## Revert scaling actual data ##
testY = testY.reshape((len(testY)))
inv_y = testX
inv_y[:,Predict_Var] = testY
inv_y = scaler.inverse_transform(inv_y)
inv_y1 = inv_y[:,Predict_Var] #Just taking predicted
variable

## y1 = actual data result, yhat1 = predicted data
result

## calculate MSE ##
mse = mean_squared_error(inv_y1, inv_yhat1)
rmse = np.roots(mse)
print('Test RMSE: %.3f' % rmse)
plt.plot(inv_y1, label='Training')
plt.plot(inv_yhat1, label='Test')
plt.legend()
plt.savefig('Relative Humidity.png', dpi=250)
plt.show()
f = open('RMSE.txt','a')
print("Relative Humidity",mse, file=f)
f.close()

## Create training data for classification ##
inv_xtest = trainX
inv_xtest = scaler.inverse_transform(inv_xtest)
inv_xtest = np.delete(inv_xtest, [8,9,10,11,12], axis =
1)
train_var_name = cols
train_var_name.remove('Conditions')
train_var_name.append('Dates')
train_x = pd.DataFrame(inv_xtest)
PrintTrain_var = train_x
PrintTrain_var['Dates'] = train_dates
PrintTrain_var.columns = [train_var_name]
PrintTrain_var = PrintTrain_var[['Dates',
'Temperature','HeatIndex','Precipitation',

```

```

'WindSpeed','WindDirection','Visibility',
'CloudCover','RelativeHumidity']]
PrintTrain_var.to_csv('TrainParameter.csv', index =
False)

## Create actual data for classification ##
test_var_name = cols
test_x = pd.DataFrame(inv_x)
PrintTest_var = test_x
PrintTest_var['Dates'] = test_dates
PrintTest_var.columns = [test_var_name]
PrintTest_var = PrintTest_var[['Dates',
'Temperature','HeatIndex','Precipitation',
'WindSpeed','WindDirection','Visibility',
'CloudCover','RelativeHumidity']]
PrintTest_var.to_csv('TestParameter.csv', index = False)

## Create predicted data for classification ##
pred_var_name = cols[Predict_Var]
pred_y = pd.DataFrame(inv_yhat1)
PrintPred_var = pred_y
PrintPred_var['Date'] = test_dates
PrintPred_var.columns =[pred_var_name, 'Date']
PrintPred_var = PrintPred_var[['Date',pred_var_name]]
PrintPred_var.to_csv(Namefile, index = False)
#Note: Change csv name per variable predicted
(Temperature, HeatIndex, etc)

## Create actual conditions for train classification ##
train_cond = df['Conditions']
train_cond = train_cond[:ntraining]
train_cond = pd.DataFrame(train_cond)
train_cond.reset_index(drop=True, inplace=True)
PrintTrain_cond = train_cond
PrintTrain_cond['Date'] = train_dates
PrintTrain_cond.columns =['Conditions', 'Date']
PrintTrain_cond = PrintTrain_cond[['Date','Conditions']]
PrintTrain_cond.to_csv('TrainConditions.csv', index =
False)

```

```

## Create actual conditions for test classification ##
act_cond = df['Conditions']
act_cond = act_cond[(ntraining+1):]
act_cond = pd.DataFrame(act_cond)
act_cond.reset_index(drop=True, inplace=True)
PrintAct_cond = act_cond
PrintAct_cond['Date'] = test_dates
PrintAct_cond.columns = ['Conditions', 'Date']
PrintAct_cond = PrintAct_cond[['Date', 'Conditions']]
PrintAct_cond.to_csv('TestConditions.csv', index =
False)

```

Kode Program Penyatuan Data

```

## Import Library ##
import pandas as pd

## Reading conditions on train and test data ##
TrainY = pd.read_csv('TrainConditions.csv')
TestY = pd.read_csv('TestConditions.csv')

## Reading parameter from regression for test in
classification ##
TestTemperature = pd.read_csv('PredTemperature.csv')
TestHeatIndex = pd.read_csv('PredHeatIndex.csv')
TestPrecipitation = pd.read_csv('PredPrecipitation.csv')
TestWindSpeed = pd.read_csv('PredWindSpeed.csv')
TestWindDirection = pd.read_csv('PredWindDirection.csv')
TestVisibility = pd.read_csv('PredVisibility.csv')
TestCloudCover = pd.read_csv('PredCloudCover.csv')
TestRelativeHumidity =
pd.read_csv('PredRelativeHumidity.csv')

## Append all parameter into one dataframe ##
TestParameter = TestTemperature
TestParameter['HeatIndex'] = TestHeatIndex['HeatIndex']
TestParameter['Precipitation'] =
TestPrecipitation['Precipitation']
TestParameter['WindSpeed'] = TestWindSpeed['WindSpeed']
TestParameter['WindDirection'] =
TestWindDirection['WindDirection']
TestParameter['Visibility'] =

```

```

TestVisibility['Visibility']
TestParameter['CloudCover'] =
TestCloudCover['CloudCover']
TestParameter['RelativeHumidity'] =
TestRelativeHumidity['RelativeHumidity']
TestParameter.to_csv('TestParameter.csv', index = False)

```

Code Program Klasifikasi *K-Nearest Neighbours*

```

from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
from sklearn.preprocessing import LabelEncoder

## Reading data ##
trainX = pd.read_csv('TrainParameter.csv')
trainX = trainX.drop(['Dates'], axis = 1)
trainX.dropna(inplace=True)

testX = pd.read_csv('TestParameter.csv')
testX = testX.drop(['Date'], axis = 1)
testX.dropna(inplace=True)

trainY = pd.read_csv('TrainConditions.csv')
trainY = trainY.drop(['Date'], axis = 1)
trainY.dropna(inplace=True)

testY = pd.read_csv('TestConditions.csv')
data_dates = pd.to_datetime(testY['Date'])
testY = testY.drop(['Date'], axis = 1)
testY.dropna(inplace=True)

## Encoder Y ##
encoderTrain = LabelEncoder()
encoderTest = LabelEncoder()
trainY = encoderTrain.fit_transform(trainY)
testY = encoderTest.fit_transform(testY)
trainY = pd.DataFrame(trainY)
testY = pd.DataFrame(testY)

## Scalling data ##
MinMaxScaler =

```

```

preprocessing.MinMaxScaler(feature_range=(0, 1))
trainXScale = MinMaxScaler.fit_transform(trainX)
testXScale = MinMaxScaler.fit_transform(testX)

X_train = pd.DataFrame(trainXScale, columns =
['Temperature', 'HeatIndex',

'Precipitation',

'WindSpeed',

'WindDirection', 'Visibility',

'CloudCover',

'RelativeHumidity'])
X_test = pd.DataFrame(testXScale, columns =
['Temperature', 'HeatIndex',

'Precipitation',

'WindSpeed',

'WindDirection', 'Visibility',

'CloudCover',

'RelativeHumidity'])
Y_train = trainY
Y_test = testY

## Creating model ##
knn_clf=KNeighborsClassifier()

## Fitting and training model ##
knn_clf.fit(X_train,Y_train)
Ypred=knn_clf.predict(X_test) #These are the predicted
output values

## Create confusion matrix as error analysis ##
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score

```

```

result = confusion_matrix(Y_test, Ypred)
print('Confusion Matrix:')
print(result)
result1 = classification_report(Y_test, Ypred)
print('Classification Report:',)
print (result1)
result2 = accuracy_score(Y_test,Ypred)
print('Accuracy:',result2)
f = open('accuracyClassification.txt','a')
print("Confusion matrix\n",result, file=f)
print("Classification Report\n",result1, file=f)
print("Accuracy\n",result2, file=f)
f.close()
Yact = encoderTrain.inverse_transform(testY)
Ypred = encoderTrain.inverse_transform(Ypred)
PdYact = pd.DataFrame(Yact)
PdYpred = pd.DataFrame(Ypred)
PdCon = pd.concat([PdYact, PdYpred], axis=1,
join='inner')
Pd = pd.concat([data_dates,PdCon], axis =1)
Pd.columns = ['Date', 'Actual Conditions','Predict
Conditions']
Pd.to_csv('Conditions Comparison.csv', index = False)

```

Code Program Visualisasi Data

```

import matplotlib.pyplot as plt
import pandas as pd
import plotly.express as px
import plotly.io as pio
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

dff = pd.read_csv('dataMalang-Modified.csv')
data = pd.read_csv('dataMalang-Modified.csv',
parse_dates=True)

def scatterplot(data):
    pio.renderers.default="browser"
    data.dropna(inplace=True)
    x = data.Precipitation
    y = data.CloudCover
    z = data.Temperature

```

```

c = data.Conditions
df = pd.DataFrame({
    'cat':c, 'Precipitation':x, 'CloudCover':y,
    'Temperature':z
})
df.head()
fig = px.scatter_3d(df, x='Precipitation',
y='CloudCover', z='Temperature',
                    color='cat',
                    title="3D Scatter Plot")

fig.show()

# def correlationplot(dff):
dff.dropna(inplace=True)
cols = list(dff)
d = pd.DataFrame(dff)
encoder = LabelEncoder()
values = d.values
values[:,9] = encoder.fit_transform(values[:,9])
df = pd.DataFrame(values)
df.columns = cols
# df.to_csv('dataMalang0E.csv', index=False)
print(df.shape)
correlation_mat = dff.corr()
sns.heatmap(correlation_mat, vmin=-1, vmax=1,
annot=True)
plt.show()

# dff = pd.read_csv('dataMalang0.csv')
# data = pd.read_csv('dataMalang0.csv',
parse_dates=True)

scatterplot(data)
# correlationplot(dff)

```