CHAPTER 4
ANALYSIS AND DESIGN

4.1. Data Preprocessing

Preprocessing the raw data is the first step in building deep learning models. This step is crucial when creating deep learning models because if this step goes wrong, then the other part will also go wrong as well. Data preprocecessing step is part where the raw data proccessed before inputed into deep learning models. Pre processing part wiil be splited into 3 parts:

6 Data Selection and Variable Analysis

7 Feature Extraction

8 Split Data

9 Data Analysis and feature scaling

4.1.1. Data Selection and Variable Analysis

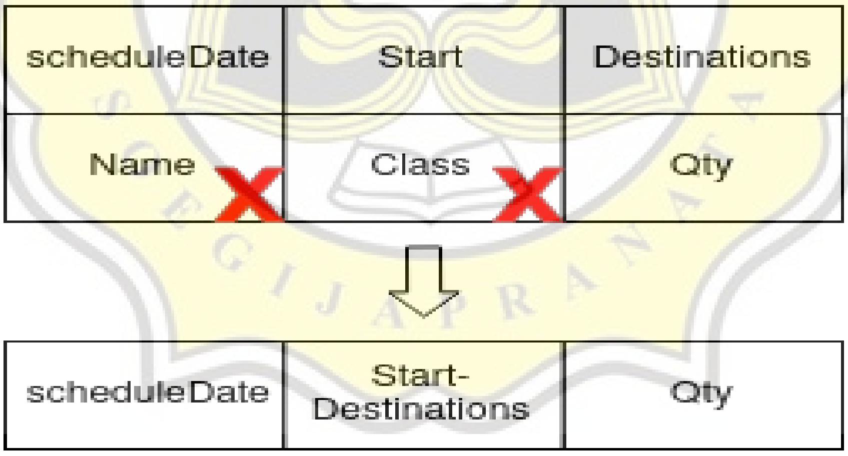
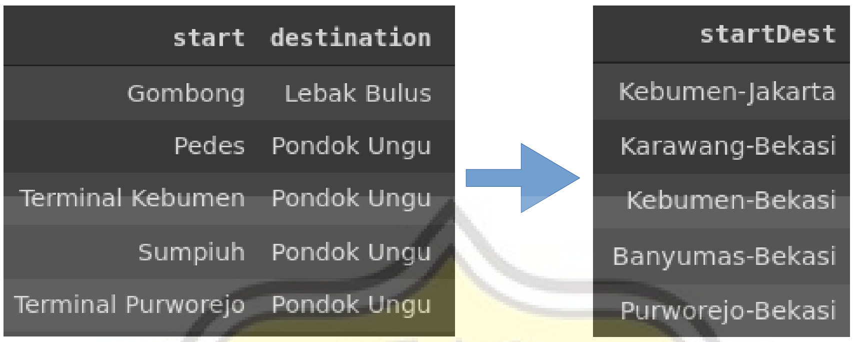


Figure 4.1: Data Selection

The first step is to selecting the data variable and drop variable that will not used. As shown in figure 4.1 in this problem Name and Class wariable will be dropped since rus route deman will rise when something triggering the people to use busses not what the bus company give facility so Class variable doesent effect on bus route demand, and Name variable is also droped and start, destination vaariable will combined into 1 variable. The start and destination data are initially the names of the departure and destination points for passengers and one city may have more than 1 point so that the values for start and destination are changed to the name of the city of the departure and destination points.

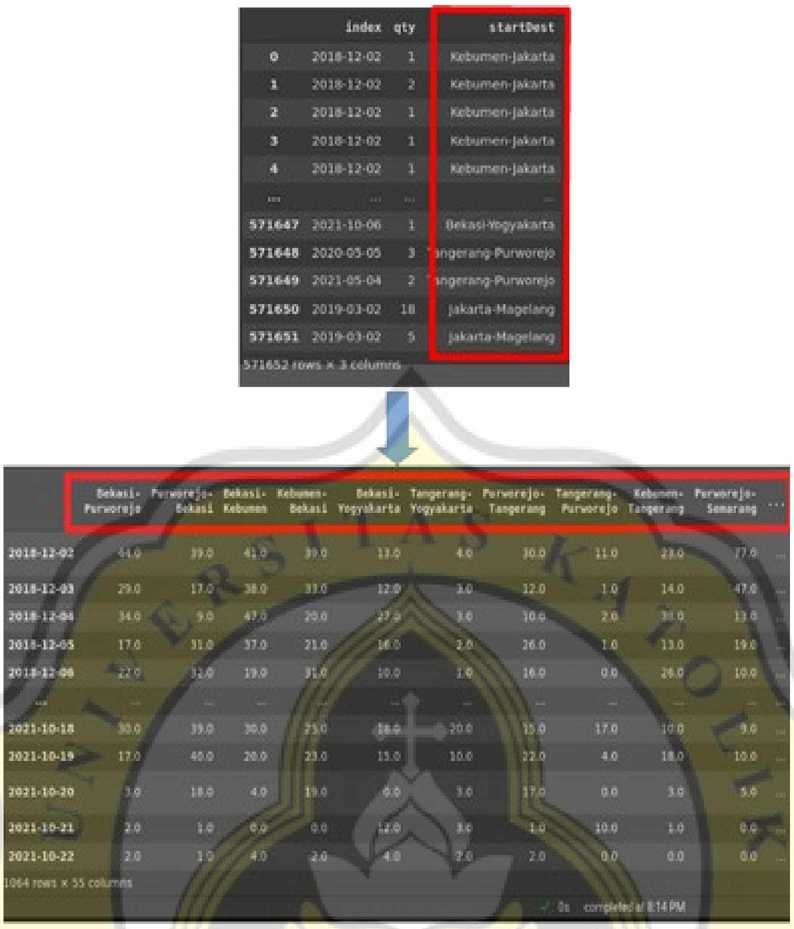


start	destination
Gombong	Lebak Bulus
Pedes	Pondok Ungu
Terminal Kebumen	Pondok Ungu
Sumpiuh	Pondok Ungu
Terminal Purworejo	Pondok Ungu

startDest
Kebumen-Jakarta
Karawang-Bekasi
Kebumen-Bekasi
Banyumas-Bekasi
Purworejo-Bekasi

Figure 4.2: start-destination transformation

As shown in figure 4.2, the start and destination variable are transformed to its city name then combined, this method will reduce the amount of start-destinations count without loosing information like in figure 4.2 first data, the “gembong” is inside the city of Kebumen and lebak bulus is inside city of Jakarta. Then after this step the data will be pivoted so each unique startDest will become a new variable and the value of each variable is the quantity of passenger on a certain date to be clear lets see figure below:



	index	qty	startDest
0	2018-12-02	1	Kebumen-Jakarta
1	2018-12-02	2	Kebumen-Jakarta
2	2018-12-02	1	Kebumen-Jakarta
3	2018-12-02	1	Kebumen-Jakarta
4	2018-12-02	1	Kebumen-Jakarta
...	...	...	...
571647	2021-10-06	1	Bekasi-Yogyakarta
571648	2020-05-05	3	Tangerang-Purworejo
571649	2021-05-04	2	Tangerang-Purworejo
571650	2019-03-02	18	Jakarta-Magelang
571651	2019-03-02	5	Jakarta-Magelang

571652 rows × 3 columns

	Bekasi-Purworejo	Purworejo-Bekasi	Bekasi-Kebumen	Kebumen-Bekasi	Bekasi-Yogyakarta	Tangerang-Yogyakarta	Purworejo-Tangerang	Tangerang-Purworejo	Kebumen-Tangerang	Purworejo-Semarang	...
2018-12-02	44.0	39.0	41.0	39.0	13.0	4.0	30.0	11.0	23.0	77.0	...
2018-12-03	29.0	17.0	38.0	33.0	12.0	3.0	12.0	1.0	14.0	47.0	...
2018-12-04	34.0	9.0	47.0	20.0	27.0	3.0	10.0	2.0	38.0	13.0	...
2018-12-05	17.0	31.0	37.0	21.0	16.0	2.0	26.0	1.0	13.0	19.0	...
2018-12-06	22.0	32.0	19.0	31.0	10.0	1.0	16.0	0.0	26.0	10.0	...
...	...	...	...	...	...	...	...	...	...	...	...
2021-10-18	30.0	39.0	30.0	23.0	18.0	20.0	15.0	17.0	10.0	9.0	...
2021-10-19	17.0	40.0	20.0	23.0	15.0	10.0	22.0	4.0	18.0	10.0	...
2021-10-20	3.0	18.0	4.0	19.0	0.0	3.0	17.0	0.0	3.0	5.0	...
2021-10-21	2.0	1.0	0.0	0.0	12.0	3.0	1.0	10.0	1.0	0.0	...
2021-10-22	2.0	1.0	4.0	2.0	4.0	7.0	2.0	0.0	0.0	0.0	...

1064 rows × 55 columns

Figure 4.3: Pivoting each unique startDest

From the figure 4.3 it shows that the data have new structure with the variable now is the startDest and the value for each unique startDest is presenting its own passenger quantity at certain date so the goal of this research that to predict each route on certain date can achieved.

4.1.2. Feature Extraction

After Data Selection and Variable Analysis is performed the next part is to find supporting variable that needed to get better model performance result later. Although the data of route itself when fed into deep learning is possible, but we need to figure how to make deep learning models can perform better job on this, so we should think what thing should affect the bus route demand. As mentioned last chapter we know that many features can be extracted from date variable so we should analyze more about what feature can be extracted from a date time variable.

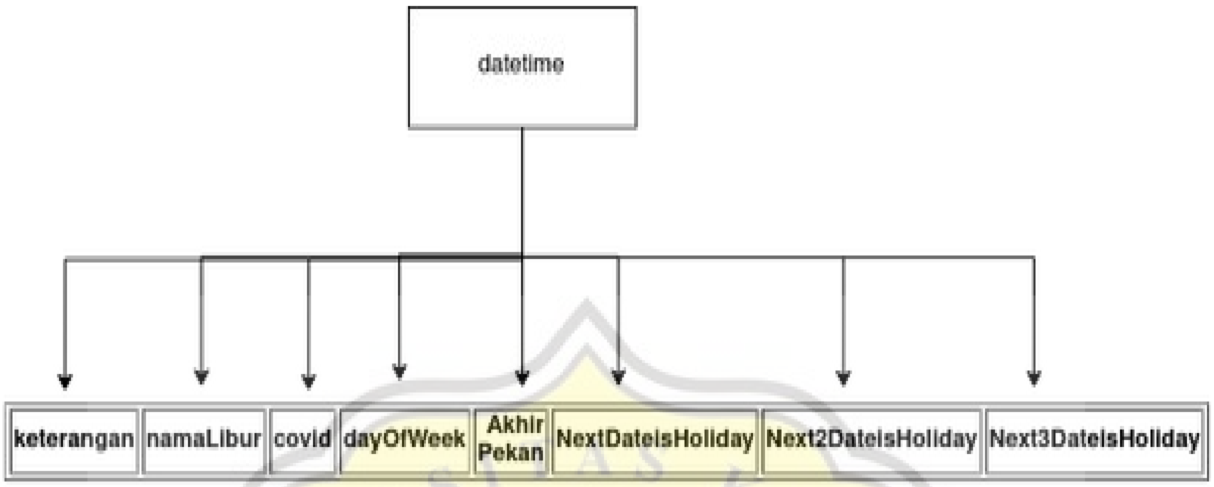


Figure 4.4: date time feature extraction

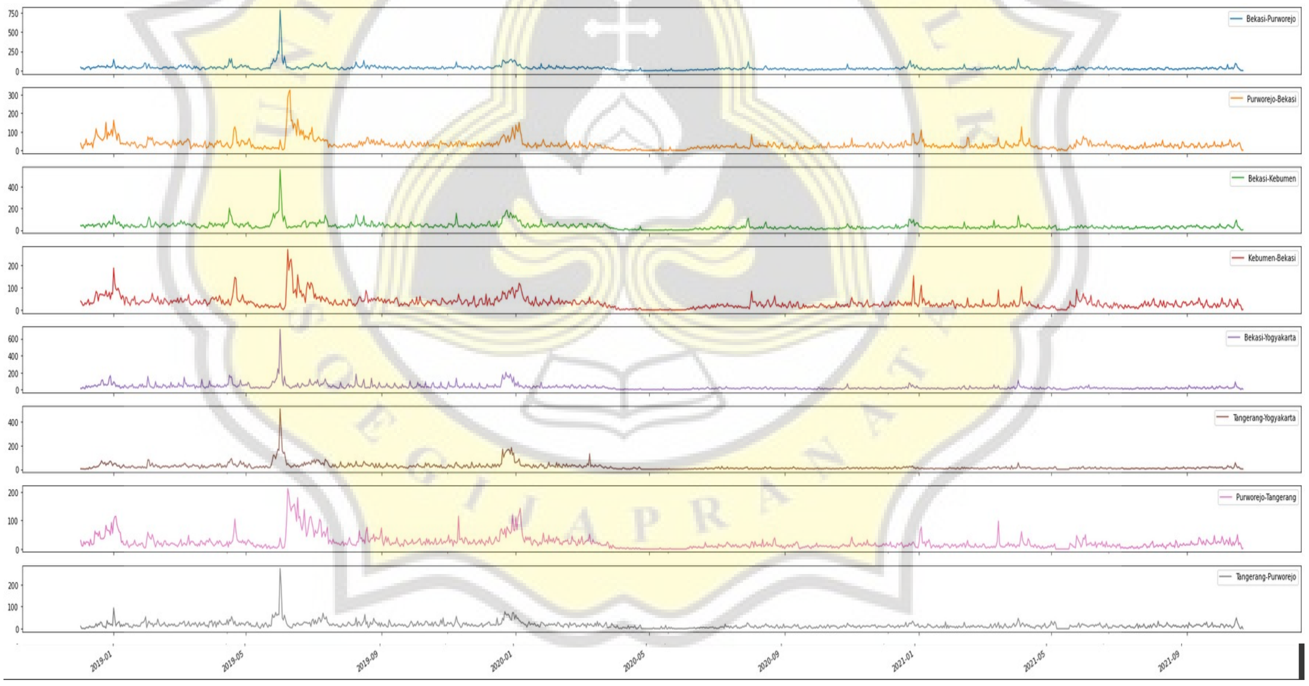
As shown in Figure 4.4 from date time data many variables can be extracted. The variable extracted is “keterangan”, “namaLibur”, “covid”, “dayOfWeek”, “Akhir Pekan”, “nextDateisHoliday”, “Next2DateisHoliday”, “Next3DateisHoliday” is obtained. Keterangan is variable that represent about certain date is holiday, normal day, and weekend, if holiday then keterangan value is either be “liburNasional” or “cutiBersama” depends on the namaLibur variable. Namalibur variable is representing the name of holiday, like “Libur Natal”, “Libur lebaran” etc. Then from date time we should know the certain date is in COVID-19 pandemic or not because since COVID-19 pandemics the bus demand is hugely drop. Then day of week is represented name of the day, for example Sunday, Monday, Tuesday, etc. nextDateisHoliday is represents that in certain date plus one day is a holiday or not, this should impact bus route demand because people tend to go to their hometown or taking a vacation and probably use bus as their transportation when the next day is holiday, and for the next2Dateisholiday is like the nextDateisHoliday but this variable represents that in certain date plus two day is either holiday or not, and the last one next3DateisHolidat is a same as nextDateisHoliday but his variable represents that in certain date plus three day is either holiday or not.

4.1.3. Split Data

The data that was previously analyzed will be entered and split in this stage. The data will be divided into two parts, the bus route variable and the categorical variable that extracted from the date time variable. The bus route variable is containing the data of the bus route passenger sum on certain date. The data then will be divided into two parts, 80% training set and 20% test set.

4.1.4. Data Analysis and feature scaling

This part is to analyze the data that has gone through the previous two stages, this step will use seaborn to visualize the corelation of each variable and data distribution will be visualized with histogram then feature scaling method will performed. Before that the author visualize the data first in order to get track of missing values. Figure below is the visualization of some of the route data.



From figure abov we can see that the data there was a very significant decrease in the number of passengers in Marc and that time is the first time COVID-19 outbrakes in Indonesia. Besides that there are no evidence of missing values in this data.

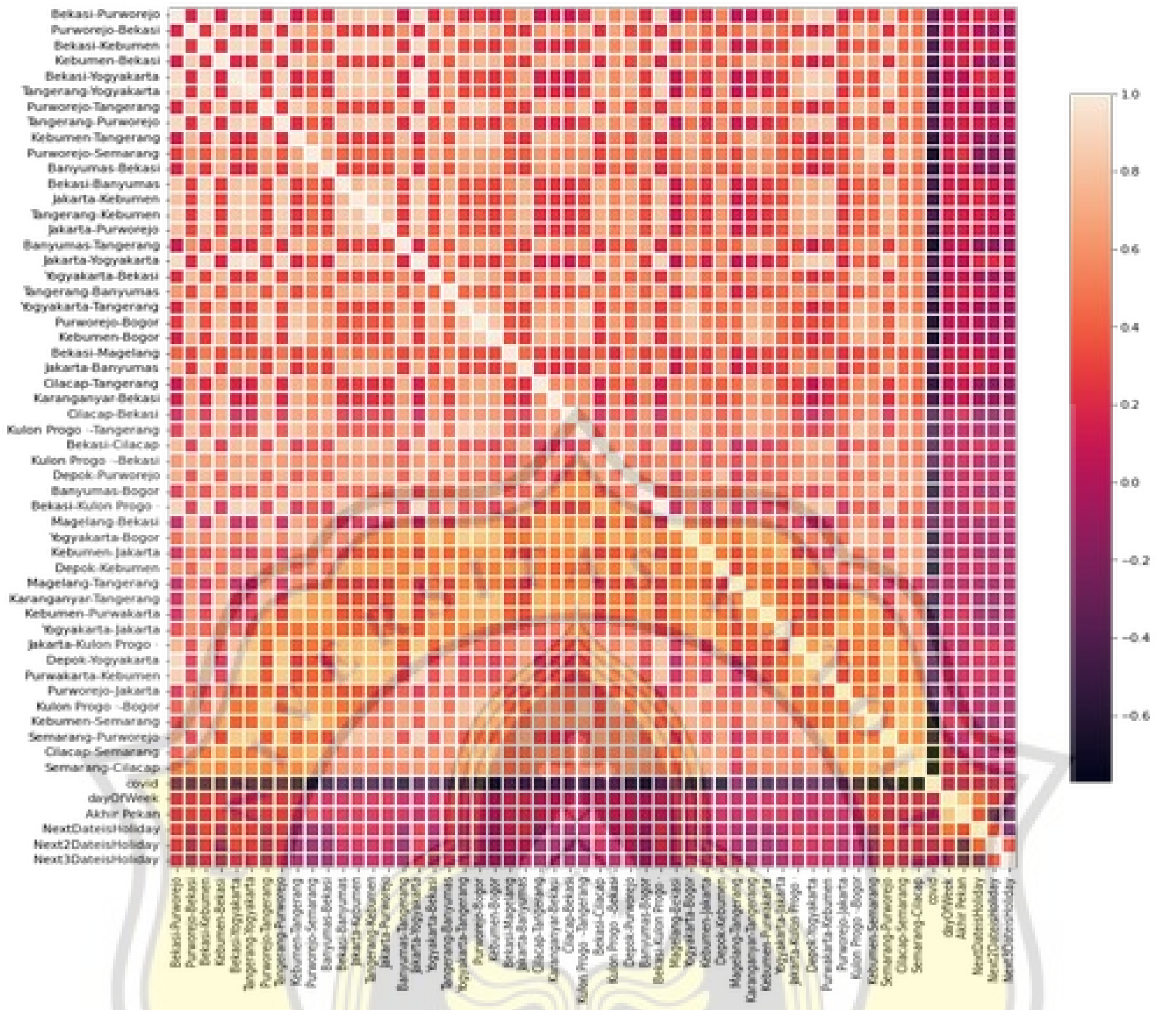


Figure 4.6: Heatmap

The value represent how each variable correlate with each other, 0 value means no correlation at all, 1 means the variable have strong positive correlation. This means when the value is going up the other value is going up as well with other variable, and negative value means the value has negative correlation with other value this means whenever variable value goes up, the value of other variables is going down since the correlation is negative.

Figure 4.5 above shows that almost all variable have correlation with each other but not all variable correlate with each other with the same direction and this is normal, since each route will have negative correlation with its return route for example yogyakarta-bekasi will have high negative correlation with bekasi-yogyakarata but yogyakarta bekasi probably only have weak correlation with semarang-kebumen either negative or positive.

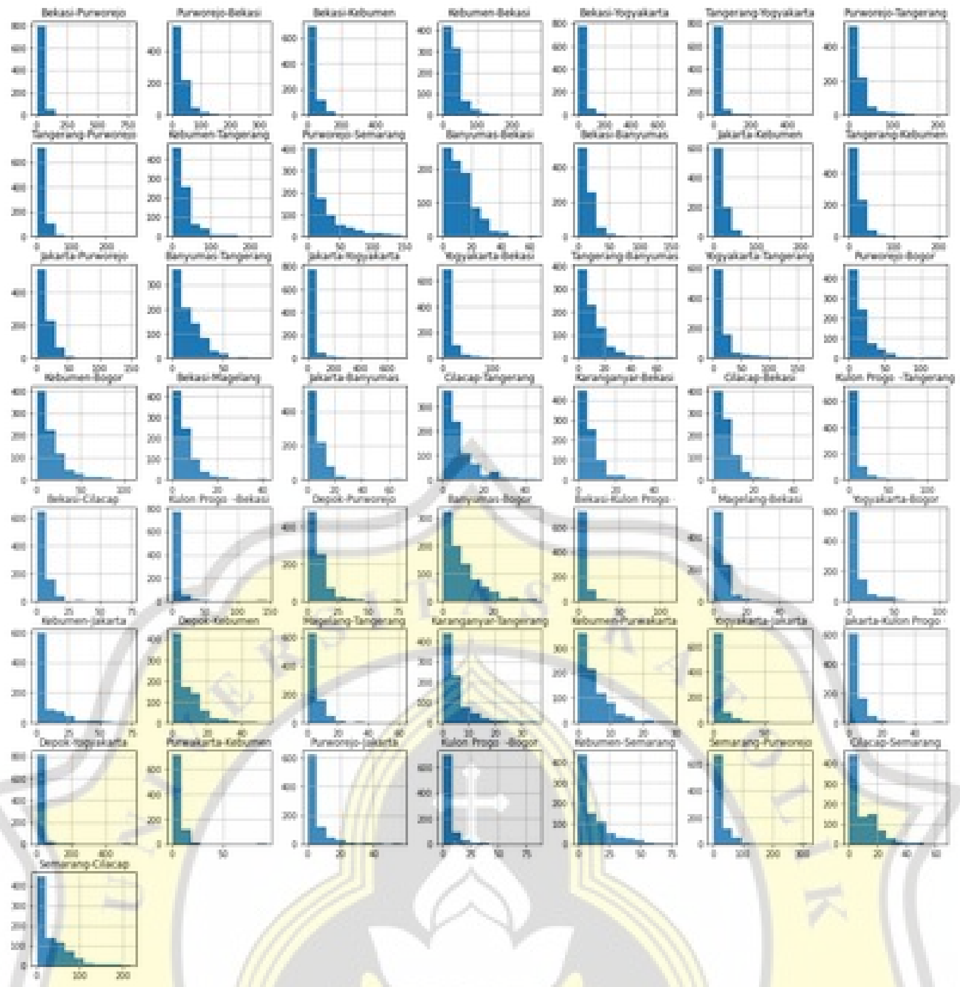


Figure 4.7: Data distribution

From the data histogram from each route in figure 4.6 it shows that for all routes the data distribution is skew, this could make the deep learning model perform bad. From this, something must be done to make the data distribution have more Gaussian like shape (bell shape) like as mentioned in last chapter to achieve better deep learning models performance. To overcome this issue the author using powertransformer from Sikit Learn library and pipeline from Sickit Learn, so with help of pipeline, the transformed data could be inverted back to its original form. In the powertransformer method there are 2 transformer method which is box-cox method and Yeo-johnson method and the method used is the Yeo-johnson method since the box-cox method is strictly positive. That means box-cox method only allowing positive value. Although the data can used both method, but in this study only yeo-johnson method will be used.

$$\psi(\lambda, y) = \begin{cases} ((y+1)^{\lambda} - 1)/\lambda & \text{if } \lambda \neq 0, y \geq 0 \\ \log(y+1) & \text{if } \lambda = 0, y \geq 0 \\ -[(-y+1)^{2-\lambda} - 1)]/(2-\lambda) & \text{if } \lambda \neq 2, y < 0 \\ -\log(-y+1) & \text{if } \lambda = 2, y < 0 \end{cases}$$

Figure 4.8: yeo-johnson formula

In this figure it can be seen that yeo-johnson formula have 4 conditional shape, and with this the yeo-johnson have the ability to transform data that are not positive. For the result after using powertransformer with yeo-johnson formula, the data distribution is having more Gaussian like shape which is good news. The result can be seen in the figure below:

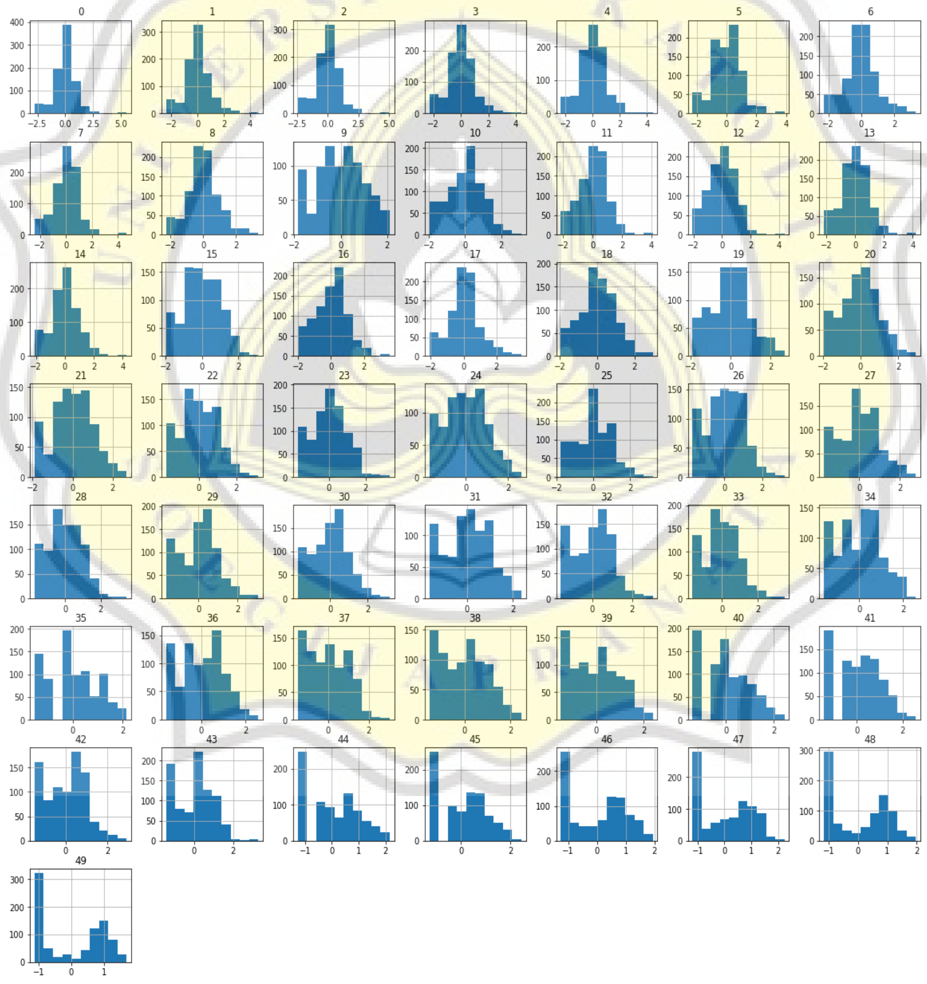


Figure 4.9: After using powertransformer

This method is for all the route variable but for the other variable ordinal scales and one hot encoder is used, ordinal encoder is mainly used to encode categorical data to numerical value, this is used for data that can be ordered, then each order could contribute for the training process such as dayofweek. When ordinal scales is use for dayofweek variable, it transforms the dayofweek value to numerical value (sunday to 0, monday to 1, etc). Then the last variable “namalibur” will be transformed with one hot encoder. One hot encoder here will transform the data variable to binary values to get clear picture see figure below:

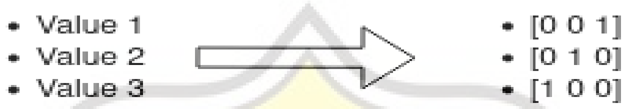


Figure 4.10: One Hot Encoder

The machine cannot understand words, so something must be done for the data, so the machine can understand it. One hot encoder and ordinal scaler is used to transform categorical data into numeric values, one hot encoder will transform data into biniary models that can be seen in figure 4.9, then the ordinal scaler will transform data into ordered integer.

4.2. Deeplearning Model

In this research 2 Deep learning will be created then the performance will be compared from each other. The first Models are LSTM Autoencoder-Bi-LSTM Hybrid Models then the second one is Bi-LSTM models. In this study Keras model that will be used is the functional models, because this problem needs more than 1 output that cannot be handled by sequential API from Keras.

This model then will be trained one by one first to save time then the trained models layer weight will be transferred for each combined model layer. This method is called transfer learning. The combined models will have multiple outputs.

4.2.1. LSTM Autoencoder-Bi-LSTM Hybrid

LSTM Autoencoers-Bi-LSTM model hybrid will have 2 part the first part is to create the LSTM autoencoders model and the second part will be creating Bi-LSTM for the output. Auto coders have 2 stages called the encoder stage then the decoder stages. Each stage will be stored into one function and act as layer. Encoder stages is a stage to compressing the data until it become into its latent space then from the latent space the decoder stages will be performed.

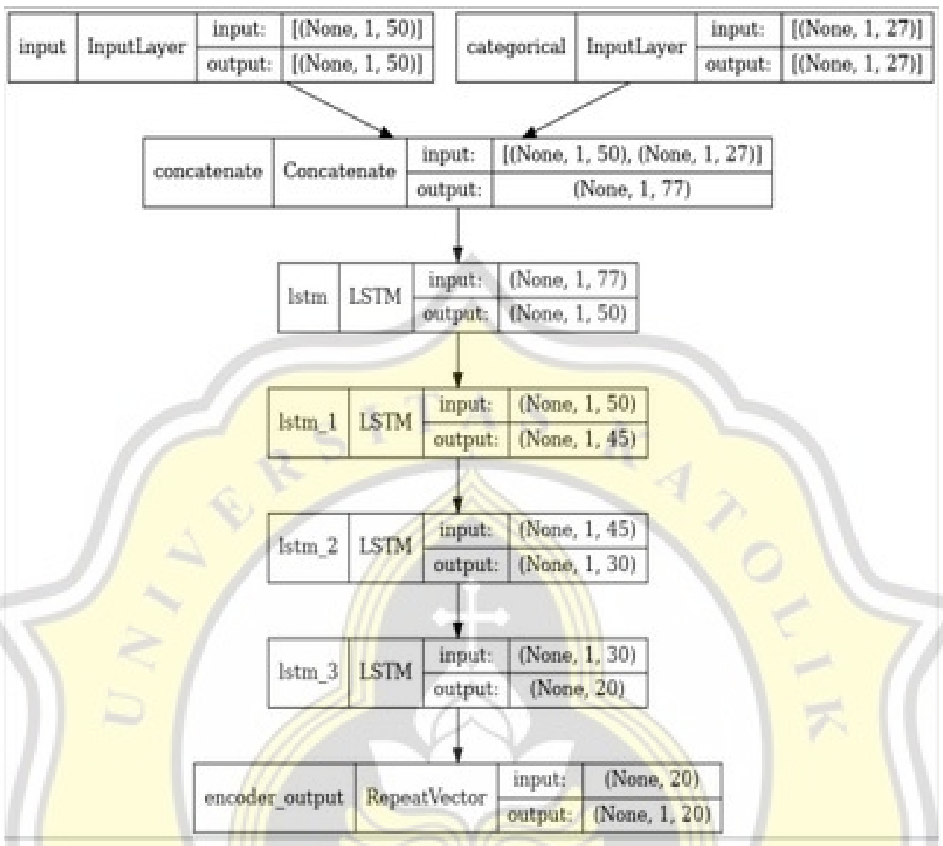
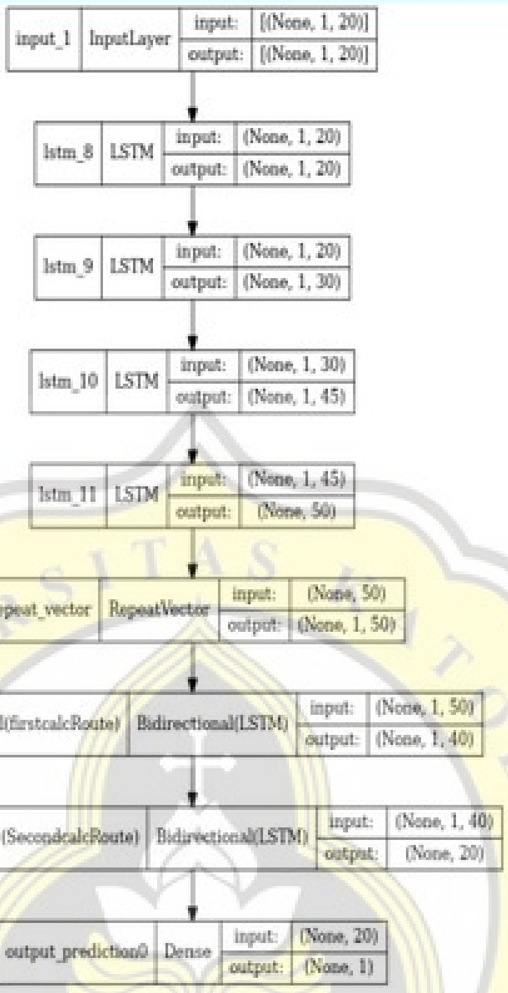

input | InputLayer | input: | [(None, 1, 50)]
output: | [(None, 1, 50)]
categorical | InputLayer | input: | [(None, 1, 27)]
output: | [(None, 1, 27)]
concatenate | Concatenate | input: | [(None, 1, 50), (None, 1, 27)]
output: | (None, 1, 77)
lstm | LSTM | input: | (None, 1, 77)
output: | (None, 1, 50)
lstm_1 | LSTM | input: | (None, 1, 50)
output: | (None, 1, 45)
lstm_2 | LSTM | input: | (None, 1, 45)
output: | (None, 1, 30)
lstm_3 | LSTM | input: | (None, 1, 30)
output: | (None, 20)
encoder_output | RepeatVector | input: | (None, 20)
output: | (None, 1, 20)

Figure 4.11: LSTM encoder Layers

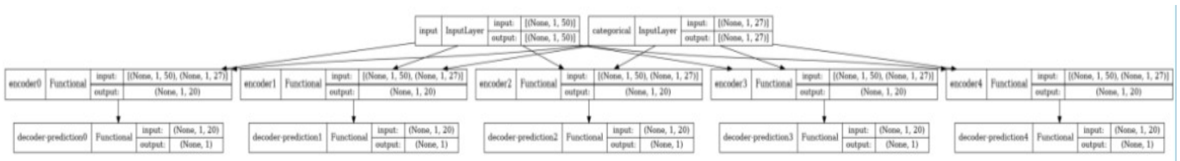
Figure 4.10 is showing how the encoder models built, from those figure, its shows there will have 2 input, the first input is for the route data variable, and for the second one then both of them are concatenated together and will have data with 77 neurons. Those 77 neurons then go into encoding stages and leaving it into latent space with only 20 neurons, each neuron are representing the data dimensions. This is the end of the encoding stages, the output is data with its latent form with 20 variable. Then those 20 variable will then be inputted to decoding stages. The decoding stages will be combined with bi-LSTM in the end for the prediction output.



input_1 | InputLayer | input: | [(None, 1, 20)]
output: | [(None, 1, 20)]
lstm_8 | LSTM | input: | (None, 1, 20)
output: | (None, 1, 20)
lstm_9 | LSTM | input: | (None, 1, 20)
output: | (None, 1, 30)
lstm_10 | LSTM | input: | (None, 1, 30)
output: | (None, 1, 45)
lstm_11 | LSTM | input: | (None, 1, 45)
output: | (None, 50)
repeat_vector | RepeatVector | input: | (None, 50)
output: | (None, 1, 50)
bidirectional(firstcalcRoute) | Bidirectional(LSTM) | input: | (None, 1, 50)
output: | (None, 1, 40)
bidirectional_1(SecondcalcRoute) | Bidirectional(LSTM) | input: | (None, 1, 40)
output: | (None, 20)
output_prediction0 | Dense | input: | (None, 20)
output: | (None, 1)

Figure 4.12: decoder-output Layer

Figure above shows the decoding-output models, start with the input which take from the encoder stages, then gradually rising the amount of variables then the output of decoder is directly inputted to bi-LSTM model with 20 neurons then the output will be inputted again into another bi-LSTM then for the last the output from second bi-LSTM layer then will be inputted to dense layer which represent the prediction of single route variable.



input | InputLayer | input: | [(None, 1, 50)] | output: | [(None, 1, 50)]
categorical | InputLayer | input: | [(None, 1, 27)] | output: | [(None, 1, 27)]
encoder0 | Functional | input: | [(None, 1, 50), (None, 1, 27)] | output: | (None, 1, 20)
encoder1 | Functional | input: | [(None, 1, 50), (None, 1, 27)] | output: | (None, 1, 20)
encoder2 | Functional | input: | [(None, 1, 50), (None, 1, 27)] | output: | (None, 1, 20)
encoder3 | Functional | input: | [(None, 1, 50), (None, 1, 27)] | output: | (None, 1, 20)
encoder4 | Functional | input: | [(None, 1, 50), (None, 1, 27)] | output: | (None, 1, 20)
decoder-prediction0 | Functional | input: | (None, 1, 20) | output: | (None, 1)
decoder-prediction1 | Functional | input: | (None, 1, 20) | output: | (None, 1)
decoder-prediction2 | Functional | input: | (None, 1, 20) | output: | (None, 1)
decoder-prediction3 | Functional | input: | (None, 1, 20) | output: | (None, 1)
decoder-prediction4 | Functional | input: | (None, 1, 20) | output: | (None, 1)

Figure 4.13: LSTM-Autoencoders-Bi-LSTM hybrid model architecture

The figure above is the architecture combined with 2 layers from 2 phases before which with functional model provided by Keras API, the models now have total of 5 output, each output represents single route predictions and each output have its own losses and metrics to get track of the model performance.

4.2.2. *Bi-LSTM Model*

This model is more simple than the LSTM-Autoencoders-Bi-LSTM hybrid model architecture. This model will directly input 2 data and output the prediction without any other architecture. This models will use bi-LSTM layers to create prediction.

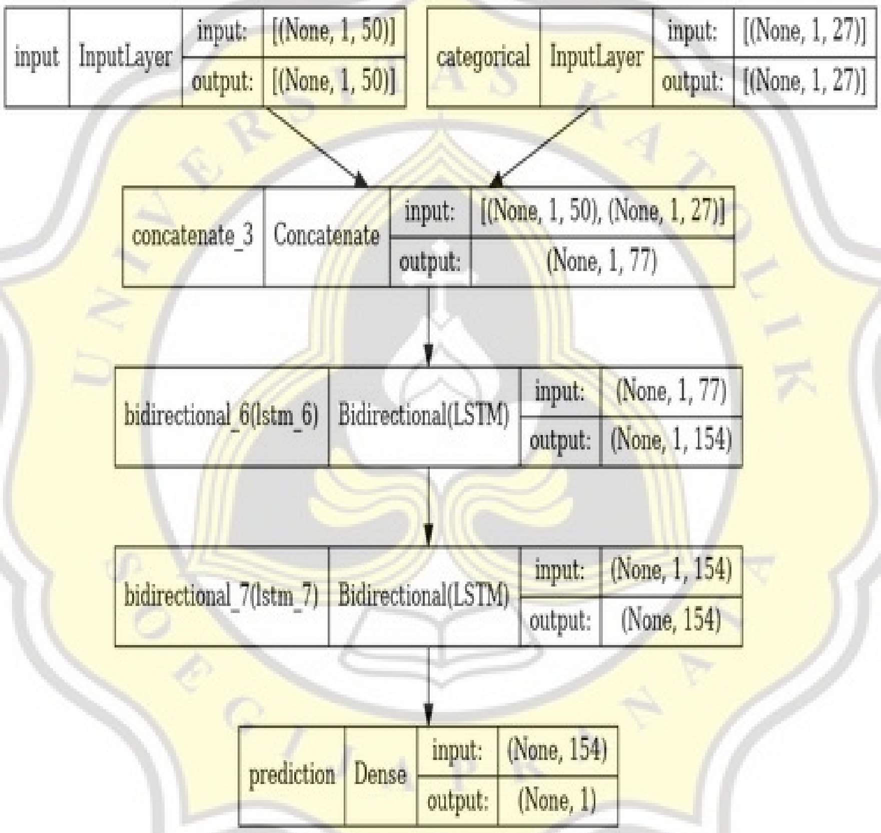

input | InputLayer | input: [(None, 1, 50)] | output: [(None, 1, 50)]
categorical | InputLayer | input: [(None, 1, 27)] | output: [(None, 1, 27)]
concatenate_3 | Concatenate | input: [(None, 1, 50), (None, 1, 27)] | output: (None, 1, 77)
bidirectional_6(lstm_6) | Bidirectional(LSTM) | input: (None, 1, 77) | output: (None, 1, 154)
bidirectional_7(lstm_7) | Bidirectional(LSTM) | input: (None, 1, 154) | output: (None, 154)
prediction | Dense | input: (None, 154) | output: (None, 1)

Figure 4.14: Bi-LSTM Model Layer

Like the LSTM-Autoencoders-Bi-LSTM hybrid, there will be 2 input node, each input represents the route variable and categorical data. Then those 2 input are concatenated into 1, then directly inputted to bi-LSTM layers with 77 neurons. Since the bidirectional learns data bidirectionally, so the number of neuron is also multiplicand by 2, That is why the output of 1 bidirectional layer is 154. The output from first bi-LSTM layer then inputted into second bi-LSTM layer which then the output is going into dense layer with 1 neuron that represents 1 single route.

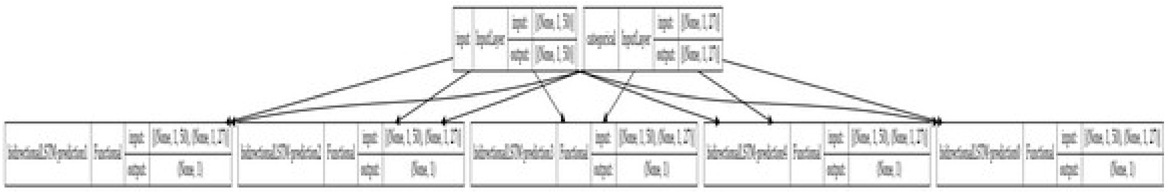

input
InputLayer
input: [(None, 1, 30)]
output: [(None, 1, 30)]
categorical
InputLayer
input: [(None, 1, 27)]
output: [(None, 1, 27)]
bidirectionalLSTM-prediction1
Functional
input: [(None, 1, 30), (None, 1, 27)]
output: (None, 1)
bidirectionalLSTM-prediction2
bidirectionalLSTM-prediction3
bidirectionalLSTM-prediction4
bidirectionalLSTM-prediction5

Figure 4.15: Bi-LSTM model Architecture

Figure above is a figure that show the final Bi-LSTM model architecture. This model will have 5 output as well that each output represents one route variable since in this research will only predict 5 route. Each output will have its own loss and metrics as well, so the performance can be tracked per output.