CHAPTER 5
IMPLEMENTATION AND RESULTS

5.1. Implementation

5.1.1. Pre-Processing

B. Pivoting the dataset

```
10 route=list()
11 for routeName in routeNames:
12  routes=dataset.loc[dataset["startDest"]==routeName]
13  routes=routes.groupby(["index",'startDest'])
     ['qty'].sum().reset_index()
14  routes=routes.rename(columns={'qty': routeName})
15  routes=routes.set_index("index")
16  route.append(routes.drop(["startDest"], axis=1))
```

The code above is meant to pivoting startDest data so for each unique startDest will become a new variable and the value of each variable is the quantity of passenger on a certain date. This work by locating each unique startdest which then each unique startDest in line 4 is grouped by the index where the index is the date time then the value is added then created new row with name of each unique startDest.

C. Function for visualize heatmap

```
1   def show_heatmap(data):
2    fig, ax = plt.subplots(figsize=(15, 15))
3    sb.heatmap(data.corr(), linewidth=0.3, cbar_kws={"shrink": .8})
4    plt.show()
```

Code above is a function to visualize the data correlation into heatmap, line 3 is the core of the function to create the heatmap with data.corr() function to calculate correlation for each variable.

D. Split data

```
trainingSetY = routesQuantityY[0:int(len(routesQuantityY)*0.8)]
testSetY = routesQuantityY[int(len(routesQuantityY)*0.8):]
trainingSet = routesQuantity.iloc[0:int(len(routesQuantity)*0.8),:-8]
testSet = routesQuantity.iloc[int(len(routesQuantity)*0.8):,:-8]
trainingsetCategorical=routesQuantity.iloc[0:int(len(routesQuantity)*
0.8),-8:]
testSetCategorical=routesQuantity.iloc[int(len(routesQuantity)*0.8):,
-8:]
```

Code above is how the data splitted. Line 1 to define the data that will be used as output in training process. Line 2 is to define the real value for later compared with the predicted value. Line 3 is training set that will used as data input and line 4 is defininf the test set for later be compared with predicted value. Line 5 is a variable to contains the categorical variable used for training step. Line 6 is categorical for later used in testing step.

E. Feature scaling

```
1  power_transformer = Pipeline(steps=[('power',
   preprocessing.PowerTransformer(method='yeo-johnson',
   standardize=True))])
2  ordinal_encoder = Pipeline(steps=[('ord',
   preprocessing.OrdinalEncoder())])
3  OneHot_encoder = preprocessing.OneHotEncoder(sparse = False)
4  preprocessorX = ColumnTransformer(
           remainder='passthrough',
           transformers=[('pwr', power_transformer ,
   trainingSet.columns),])
5  preprocessorXCategorical= ColumnTransformer(
          remainder='passthrough',
           transformers=[
             ("ohe", OneHot_encoder, ['namaLibur']),
             ('ord', ordinal_encoder , ['keterangan',
   'covid','dayOfWeek','Akhir
   Pekan',"NextDateisHoliday","Next2DateisHoliday","Next3DateisHoliday"]
   ),
           ])
6  preprocessorY =
   ColumnTransformer(remainder='passthrough',transformers=[ ('pwr',
   power_transformer , routesQuantityY.columns),])
7  XTrainScaled = np.asarray(preprocessorX.fit_transform(trainingSet))
8  XTrainScaledCategorical =
   np.asarray(preprocessorXCategorical.fit_transform(trainingsetCategori
   cal))
9  XTestScaled= np.asarray(preprocessorX.transform(testSet))
10 YTrainScaled = np.asarray(preprocessorY.fit_transform(trainingSetY))
11 YTrainScaledCategorical =
   np.asarray(preprocessorXCategorical.transform(testSetCategorical))
12 YTestScaled = np.asarray(preprocessorY.transform(testSetY))
```

This code is to store the feature scaler into variable which then will be used to feature scaling the data. Line 1 is a variable that contains a pipeline to containing the power transformer with the yeo-johnson method to make data distribution more Gaussian like. Line 2 is a variable that contains pipeline of ordinal encoder which later used to transform categorical data into ordered value. Line 3 is a code to make variable that containing one hot encoder which used to transform categorical data into binary value. Line 4 is to use define the ColumnTransformer method for later used to transform the data with powertransormer for variable trainingSet in line 7. Line 5 is to use define the ColumnTransformer method for later used to transform the data with one hot encoder for “namaLibur” variable and ordinal encoder for 'keterangan', 'covid', 'dayOfWeek', 'Akhir Pekan', "NextDateisHoliday", "Next2DateisHoliday", "Next3DateisHoliday" variables for variable trainingsetCategorical in line 8. Line 6 is to use define the ColumnTransformer method for later used to transform the data with powertransormer for variable testSetY in line 12.

5.1.2. LSTM-Autoencoders-Bi-LSTM hybrid model

A. Encoder Layer

```
1   def encoderModels():
2     input_layer = Input(shape=(1, XTrainScaled.shape[1]),
    name='input')
3     input_categorical = Input(shape=(1,
    XTrainScaledCategorical.shape[1]), name='categorical')
4     concatenateCategorical=concatenate([input_layer,
    input_categorical])
5     x=LSTM(XTrainScaled.shape[1],return_sequences=True ,
    activation="selu")(concatenateCategorical)
6     x=LSTM(45,return_sequences=True ,  activation="selu",
    kernel_initializer='lecun_normal')(x)
7     x=LSTM(30,return_sequences=True , activation="selu",
    kernel_initializer='lecun_normal')(x)
8     x=LSTM(20,return_sequences=False , activation="selu",
    kernel_initializer='lecun_normal')(x)
9     x=RepeatVector(1, name='encoder_output')(x)
10    encoder = Model([input_layer, input_categorical], x,
    name="encoder")
11    encoder.summary()
12    return encoder
```

Code above is a function to create LSTM encoder layers, line 2 and is how to create the input layers, line 4 is to concatenate input layers, line 5-8 is to compressing the data into latent space, it shws that from line 5-8 the neuron number is decreasing until only 20. then the last one for line 9 is a reapeat vector for the encoder model output.

B. Decoder-output layer

```
1   def outputsModels():
2     decoder_input=Input(shape=(1,20))
3     x =LSTM(20, return_sequences=True, activation="selu",
    kernel_initializer='lecun_normal')(decoder_input)
4     x =LSTM(30, return_sequences=True, activation="selu",
    kernel_initializer='lecun_normal')(x)
```

```
5    x =LSTM(45,return_sequences=True, activation="selu",
   kernel_initializer='lecun_normal')(x)
6    x =LSTM(XTrainScaled.shape[1], return_sequences=False,
   activation="selu", kernel_initializer='lecun_normal')(x)
7    x=RepeatVector(1)(x)
8  calc_1=Bidirectional(LSTM(20,name=f'firstcalcRoute',return_sequ
   ences=True, activation="selu",
   kernel_initializer='lecun_normal'))(x)
9  calc_2=Bidirectional(LSTM(10,name=f'SecondcalcRoute',return_seq
   uences=True,activation="selu",
   kernel_initializer='lecun_normal'))(calc_1)
10   output=Dense(1,name=f'output_prediction', activation="selu",
   kernel_initializer='lecun_normal')(calc_2)
11   decoder = Model(decoder_input, output, name="decoder-
   prediction")
12   decoder.summary()
13   return decoder
```

This code is to create Decoder layer to decode data from latent space back to its normal dimension. Line 2 is the input layer, the input is got from encoder output. Line 3-6 is creating LSTM layer, but the number of neuron is gradually added until the size is back to normal size. Then line 8-9 is meant to add bi-LSTM layer to make calculation for the prediction then finally, line 10 is dense layer where the output from line 9 is inputted into this layer then dense layer will provide 1 neuron which represents the prediction output for 1 route.

C. Combining layer

```
1  def autoencoderModels(encoder, output):
2    input1= Input(shape=(1, XTrainScaled.shape[1]), name='input')
3    input2 = Input(shape=(1, XTrainScaledCategorical.shape[1]),
   name='categorical')
4    encode = encoder([input1, input2])
5    outputPrediction= output(encode)
6    autoencoder = Model([input1,input2], outputPrediction,
   name="autoencoder")
7    autoencoder.summary()
8    return autoencoder
9  output=models()
10 output.compile(
11     optimizer=tf.Keras API.optimizers.Adam(),
12     loss={"prediction":"mse"},
13     metrics={"prediction":tf.Keras
   API.metrics.RootMeanSquaredError()}
14 )
```

This code is to create LSTM-Autoencoder-Bi-LSTM models. Line 2 and 3 is the input layers, line 4 is where encoder layer function that have created before called to make new encoder layer. Line 5 is where decoder-output layer function are called to create new decoder-output layer where the input is the encoder layer output. Line 9 is to create a variable that store the created models. Line 10 is to compile the layer, so it's now trainable.

D. Training Model

```
1  encoder=encoderModels()
2  output=outputsModels()
3  autoencoder=autoencoderModels(encoder,output)
4  autoencoder.compile(
5  optimizer=tf.Keras API.optimizers.Adam(),
6  loss={"decoder-prediction":"mse",},
7  metrics={"decoder-prediction":tf.Keras
   API.metrics.RootMeanSquaredError(),})
8  earlyStop = EarlyStopping(monitor='loss', mode='min',
   verbose=1, patience=15)
9  modelCheckpoints = ModelCheckpoint('1Best.h5', monitor='loss',
   mode='min', verbose=1, save_best_only=True)
10 output.fit([XScaled,Xcat], YScaled[0], epochs=200,
   callbacks=[earlyStop, modelCheckpoints], batch_size=1)
```

This code show how this model trained, line 1 is to define the encoder layers line 2 is to define the decoder-output layer then line 3 is to define the LSTM-Autoencoder-Bi-LSTM hybrid models where the first param is the encoder and the second is decoder-output layer. Line 4 is how the models are compiled which in this line loss and metrics are defined to. Line 6 is to define early stopping so the overfitting problems could be prevented, line 7 is to define the modal check points. Modelcheckpoints is used so when the training is done, the best weight will be saved. Then finally, line 8 it's where to train the model using fit() function. The first param is to define the data, second param is for the data output, third param is to define how many epochs will be used, and the fourth param is to define callbacks where the value are variables from line 1 and 2.

E. Combining Trained models into 1 model

Since training stages will be done one by one first to save time, then to combine all trained models, transfer learning method proceed For this the function from before is modified.

```
def autoencoderModels(encoder, output):
  input1= Input(shape=(1, XTrainScaled.shape[1]),
name='input')
  input2 = Input(shape=(1, XTrainScaledCategorical.shape[1]),
name='categorical')
  encode = []
  outputPrediction=[]
```

```
6    for i in range(len(output)):
7      encode.append(encoder[i](input1))
8      outputPrediction.append(output[i]([encode[i], input2]))
9    autoencoder = Model([input1,input2], outputPrediction,
   name="autoencoder")
10   autoencoder.summary()
11   return autoencoder
```

This code is the modified code from method before so the output can be more than one. The modified part is in line 6, in line 6 for loop is used to looping through all the output number needed.

```
1  encoder=[]
2  output=[]
3  for i in range(numOfRoute):
4      autoencoder = load_model(f'{i}Best(modelsPerRoute).h5')
5      trainedEncoder=autoencoder.get_layer(name="encoder")
6      trainedDecoder=autoencoder.get_layer(name="decoder-
   prediction")
7      enc=encoderModels(i)
8      enc.set_weights(trainedEncoder.get_weights())
9      enc.trainable=False
10     out=outputsModels(i)
11     out.set_weights(trainedDecoder.get_weights())
12     out.trainable=False
13     encoder.append(enc)
14     output.append(out)
15 autoencoder=autoencoderModels(encoder, output)
```

This code is how the trained models combined into 1 big model with multiple output. Line 1 and 2 is to define the list to store model encoder and decoder-output layers, then going to for loop for number of route that want to be predicted, for this study numOfRoute value is 5, line 3 is to load trained autoencoder model. Line 5 is where the encoder layer from trained autoencoder is stored into variable, line 5 is where the decoder-output layer from trained autoencoder is stored into variable then in line 7 is where encoder layer is created and stored into a variable Then line 8 is to set the weight of that variable is set to the encoder layer from trained autoencoder. Line 9 is to set the encoder layer trainable params to false, so this layer weight is locked so when in training there are no weight value update to this layer. Line 10 is where decoder-output layer is created and stored into variable. Then line 11 is to set the weight of the new decoder-output variable is set to the decoder-output layer from trained autoencoder. Line 12 is to set the encoder layer trainable params to false, so this layer weight is locked. Line 13 and 14 is to store the new encoder and decoder with weight equal to trained models into list that created in line 1 and 2. Line 15 is to combine all new layer into 1 with 5 different outputs with modified method above.

5.1.3. Bi-LSTM model

```
1   def models():
2     input_layer = Input(shape=(1, XTrainScaled.shape[1]), name='input')
3     input_categorical = Input(shape=(1,
    XTrainScaledCategorical.shape[1]), name='categorical')
4     concatenateCategorical=concatenate([input_layer,
    input_categorical])
5     x=Bidirectional(LSTM(77,return_sequences=True, activation="selu",
    kernel_initializer='lecun_normal'))(concatenateCategorical)
6     x=Bidirectional(LSTM(77,return_sequences=False, activation="selu",
    kernel_initializer='lecun_normal'))(x)
7     output=Dense(1,name=f'prediction', activation="selu",
    kernel_initializer='lecun_normal')(x)
8     model = Model([input_layer, input_categorical], output, name="Bi-
    lstm")
9     model.summary()
10    return model
```

This code is a function to create a new Bi-LSTM models. Line 2 and 3 is to create the input layer. Line 4 is where the input layer is concatenated into 1. Line 6 is where bi-LSTM layer created with total neuron of 77. Then finally, line 7 is the output layer where dense layer is used with 1 neuron which represent 1 single route prediction.

```
1  earlyStop = EarlyStopping(monitor='loss', mode='min', verbose=1,
   patience=15)
2  modelCheckpoints = ModelCheckpoint('1Best(NoAutoencoder).h5',
   monitor='loss', mode='min', verbose=1, save_best_only=True)
3  output.fit(
4     [XScaled,XCat],
5      YScaled[0],
6      epochs=200,
7      callbacks=[earlyStop, modelCheckpoints],
8      batch_size=1,
9  )
```

This code show how the Bi-LSTM model trained, line 1 is to define early stopping so the overfitting problems could be prevented, line 2 is to define the modal check points. Modelcheckpoints is used so when the training is done the best weight will be saved. Then finals line 3 It's where to train the model using fit() function. The first param is to define the data, second param is for the data output, line 6 is to define how many epochs will be used, line 7 is to define callbacks where the value are variables from line 1 and 2.

```
1    def combinedModels(output):
2     input1= Input(shape=(1, XTrainScaled.shape[1]), name='input')
3     input2 = Input(shape=(1, XTrainScaledCategorical.shape[1]),
   name='categorical')
4     outputPrediction=[]
```

```
5     for i in range(len(output)):
6         outputPrediction.append(output[i]([input1,input2]))
7            lstmModel = Model([input1,input2], outputPrediction,
   name="bidirectionalLSTM")      name="autoencoder")
8     lstmModel.summary()
9     return lstmModel
```

Code above is a method to combining all trained models into one big model with multiple output. Line 2 and 3 is to define the input layers, line 5 is for loop that looping trough number of output length.

```
output=[]
for i in range(numOfRoute):
    pretrained = load_model(f'{i+1}Best(NoAutoencoder).h5')
    out=outputsModels(i)
    out.set_weights(pretrained.get_weights())
    out.trainable=False
    output.append(out)
models=combinedModels(output)
```

this code is for taking the weight of each trained models and copy to new models and stored into list then that list later will inputed into combinedModels method.

5.1.4. Prediction and inverse transformation

```
1  firstEvalBatch = XTrainScaled[-1]
2  firstEvalBatchCat = XTrainScaledCategorical[-1]
3  firstEvalBatch=autoencoder.predict([firstEvalBatch.reshape(1, 1,
   firstEvalBatch.shape[0]),firstEvalBatchCat.reshape(1, 1,
   firstEvalBatchCat.shape[0])])
4  firstEvalBatch=np.vstack(firstEvalBatch[0])
5  prediction=[firstEvalBatch]
6  for i in range(0,XTestScaled.shape[0]):
7    # print(prediction[i])
8    mergedlist = []
9    mergedlist.extend((prediction[i].reshape(1,numOfRoute))[0])
10   mergedlist.extend(XTestScaled[i,numOfRoute:])
11   predictBatch=np.asarray(mergedlist)
12   predictBatch = predictBatch.reshape((1, 1, predictBatch.shape[0]))
13   # print(YTrainScaledCategorical[i].shape)
14   predictBatch=autoencoder.predict([predictBatch,
   YTrainScaledCategorical[i].reshape(1, 1,
   YTrainScaledCategorical[i].shape[0])])
15   predictBatch=np.vstack(predictBatch[0])
16   # print(predictBatch)
17   # print(prediction)
18   prediction.append(predictBatch)
```

This code is make prediction of the trained models with the test dataset and the models will evaluated here. The for loop is used to loop trough iteration of test set data. The line 1-4 is where the first prediction batch performed is to store the first batch prediction. Since the data is sequential so the first data used for the very first prediction is the last data from training set. See figure below to get more clear picture.

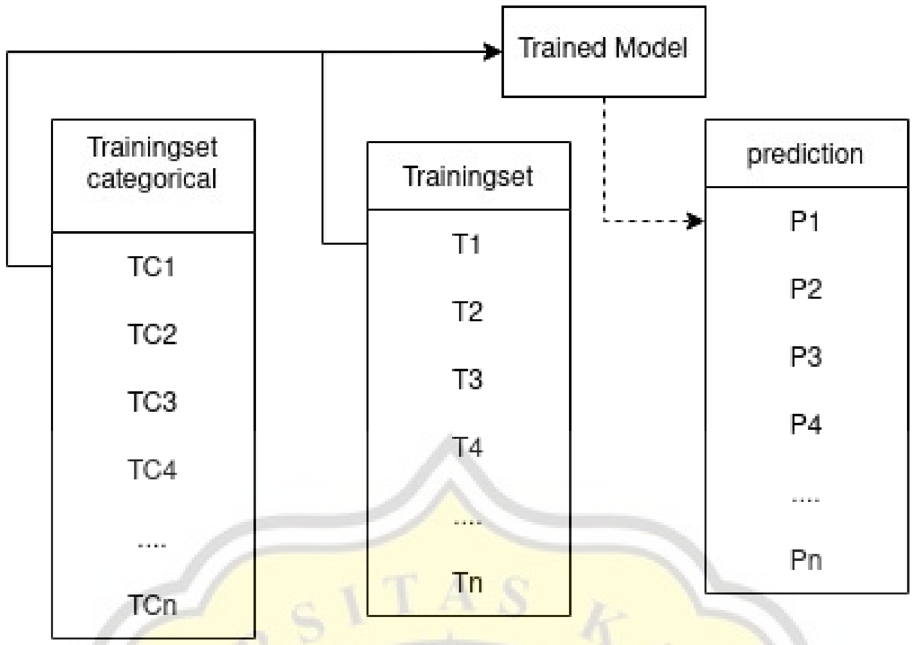


Figure 5.1: First prediction batch

From figure above we can clearly see that the lats data from training set and categorical training set data is fed into trained model to create the first prediction batch. Then the first training batch is then stored into list. Line 6-18 is where the prediction per sequence done. In this step the data that iputed into trained model is the first data that taken from the first prediction batch with the categorical data from the test set. To get clear picture se below figure :

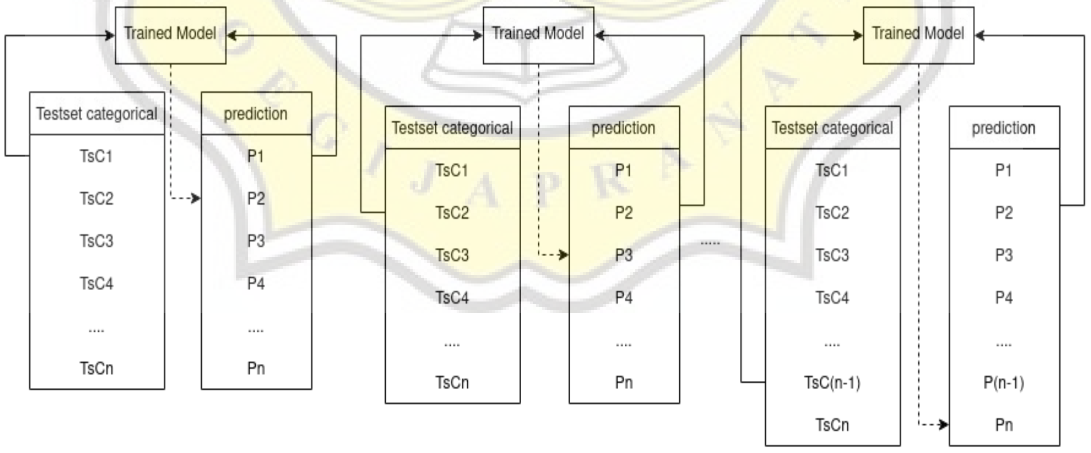
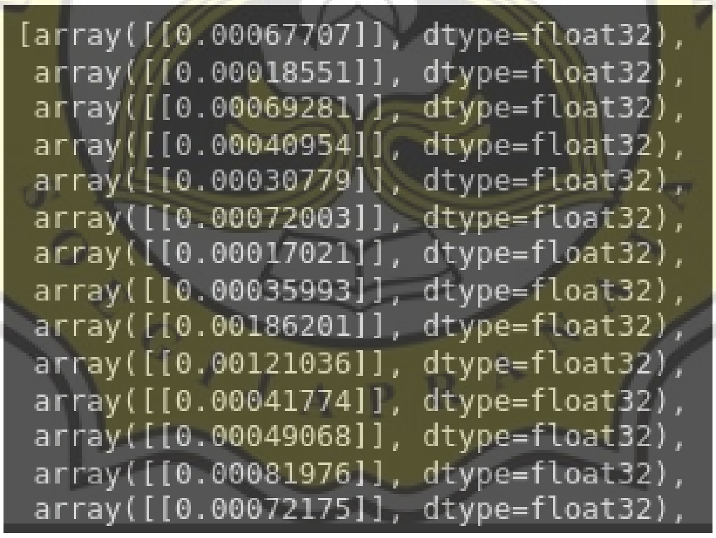


Figure 5.2: prediction per sequence

Figure 5.2 is showing how was the line 6-18 code works. As seen in that figure the prediction created with the data predicted before and added with the categorical data fdrom test set. This loop will loop until the end of the test set data. But need to be mentioned, the prediction output from the trained model is still in scaled form, so in order to transform the data back to its original form, inverse method should be done.

```
1  prediction=np.array(prediction)
2  prediction=prediction.reshape(prediction.shape[0],prediction.shape[1]
   )
3  testPredictionsDf=preprocessorY.named_transformers_['pwr'].inverse_tr
   ansform(prediction)
4  YTestScaledDf=preprocessorY.named_transformers_['pwr'].inverse_transf
   orm(YTestScaled)
5  testPredictionsDf = pd.DataFrame(testPredictionsDf,
6                columns=preprocessorY.transformers_[0][2])
7  YTestScaledDf = pd.DataFrame(YTestScaledDf,
8                columns=preprocessorY.transformers_[0][2])
```

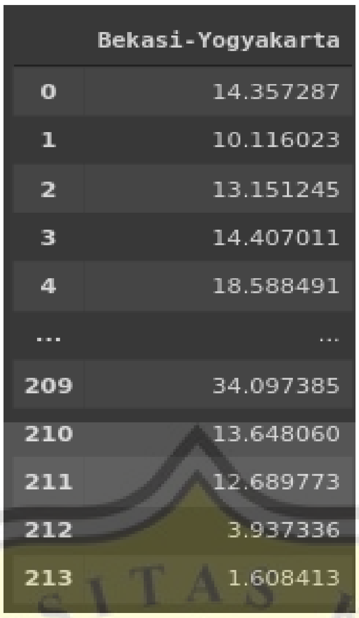
Code above is how the inverse method done. Line 1 is to transform the data from list into array first then the data reshape into the original shape. After that in line 3 is how the data transform inverse done. Line 4 is how where the inverse done. And line 5-8 is how the column name of the data is retrived back.



```
[array([[0.00067707]], dtype=float32),
 array([[0.00018551]], dtype=float32),
 array([[0.00069281]], dtype=float32),
 array([[0.00040954]], dtype=float32),
 array([[0.00030779]], dtype=float32),
 array([[0.00072003]], dtype=float32),
 array([[0.00017021]], dtype=float32),
 array([[0.00035993]], dtype=float32),
 array([[0.00186201]], dtype=float32),
 array([[0.00121036]], dtype=float32),
 array([[0.00041774]], dtype=float32),
 array([[0.00049068]], dtype=float32),
 array([[0.00081976]], dtype=float32),
 array([[0.00072175]], dtype=float32),
```

Figure 5.3: prediction output

As seen in figure 5.3 the output prediction is still on the form of scaled data then code above will transform the scaled data back to its original form.



	Bekasi-Yogyakarta
0	14.357287
1	10.116023
2	13.151245
3	14.407011
4	18.588491
...	...
209	34.097385
210	13.648060
211	12.689773
212	3.937336
213	1.608413

Figure 5.4: predicted output after inversed

Then finaly as seen in figure 5.4, the data form is back to its normal form with the column name retrived back.

5.2. Result

5.2.1. LSTM-Autoencoders-Bi-LSTM hybrid model

A) Training losses and metrics

ion used is MSE for the metrics RMSE is used. Lets se how LSTM-Autoencoders-Bi-LSTM hybrid model perform in figure below.

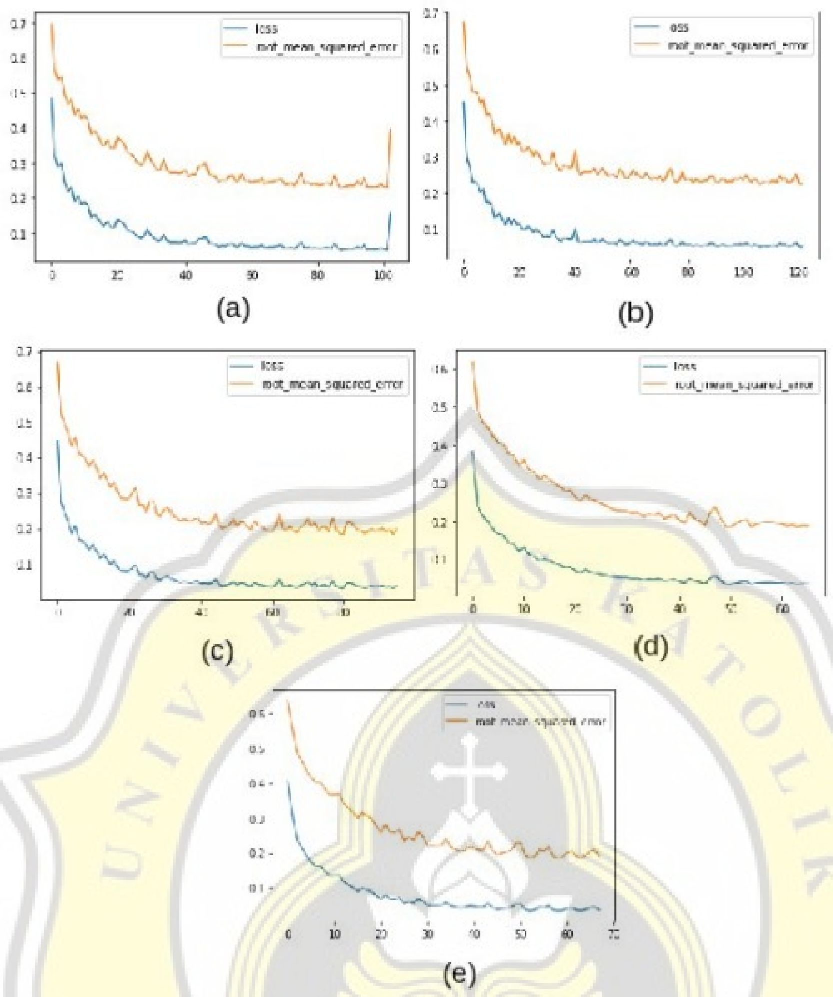


Figure 5.5: training Loss and metric (a) first route, (b) second route, (c) third route, (d) fourth route, (f) fifth route, for LSTM-Autoencoders-Bi-LSTM hybrid model

From figure 5.5, we can see the loss and metrics value is gradually going down until the value is not going down anymore, the step of learning process is epoch, each epoch is trying to find the lowest loss and this is called global minima, to find the global minima, stochastic gradient descent is used for deep learning models with optimizer adam. Gradient desent is working by randomly set weight for each neuron to find the global minima as possible, so large epoch is needed. But, LSTM are facing problem of vanishing gradient and deep learning could overfit when deep learning models is trying to copy the input directly to outputs, and this is major problem, so to overcome this a limiter is need to get rid of vanishing gradient and overfitting. For this model the author use early stopping of the model training process with patience of 5, so when the epoch is not lowering anymore for 5 epochs, then the training process is automatically stops to prevent vanishing gradient and over fitting problem. The result can be seen in figure 5.5 and the lowest loss and metrics in table below.

Table 5.1: Lowest Loss and metrics per route

	Route 1	Route 2	Route 3	Route 4	Route 5
Loss	0.0365	0.032	0.0288	0.0296	0.0273
Metrics	0.191	0.1789	0.1697	0.172	0.1652

As seen in table 5.1 all the loss value is below 0.1 and the metrics is below 1 Which is good news because the lower the loss and metrics value, the more similar the predicted value to the original value since MSE is the mean squared error of predicted by the original value.

B) Prediction of test set

In order to prove and judge about the model performance, testing of the trained model is needed So to prove test this models performance, the test set will be used to check the performance, the trained model then inputted test data and the models will try to predict the value per sequences but because the trained models input value is transformed with power transformer then to transform the data back just need to inverse the transformation back to its normal form. Then predicted result is plotted side by side with its training set value to proven that the model is performing well.

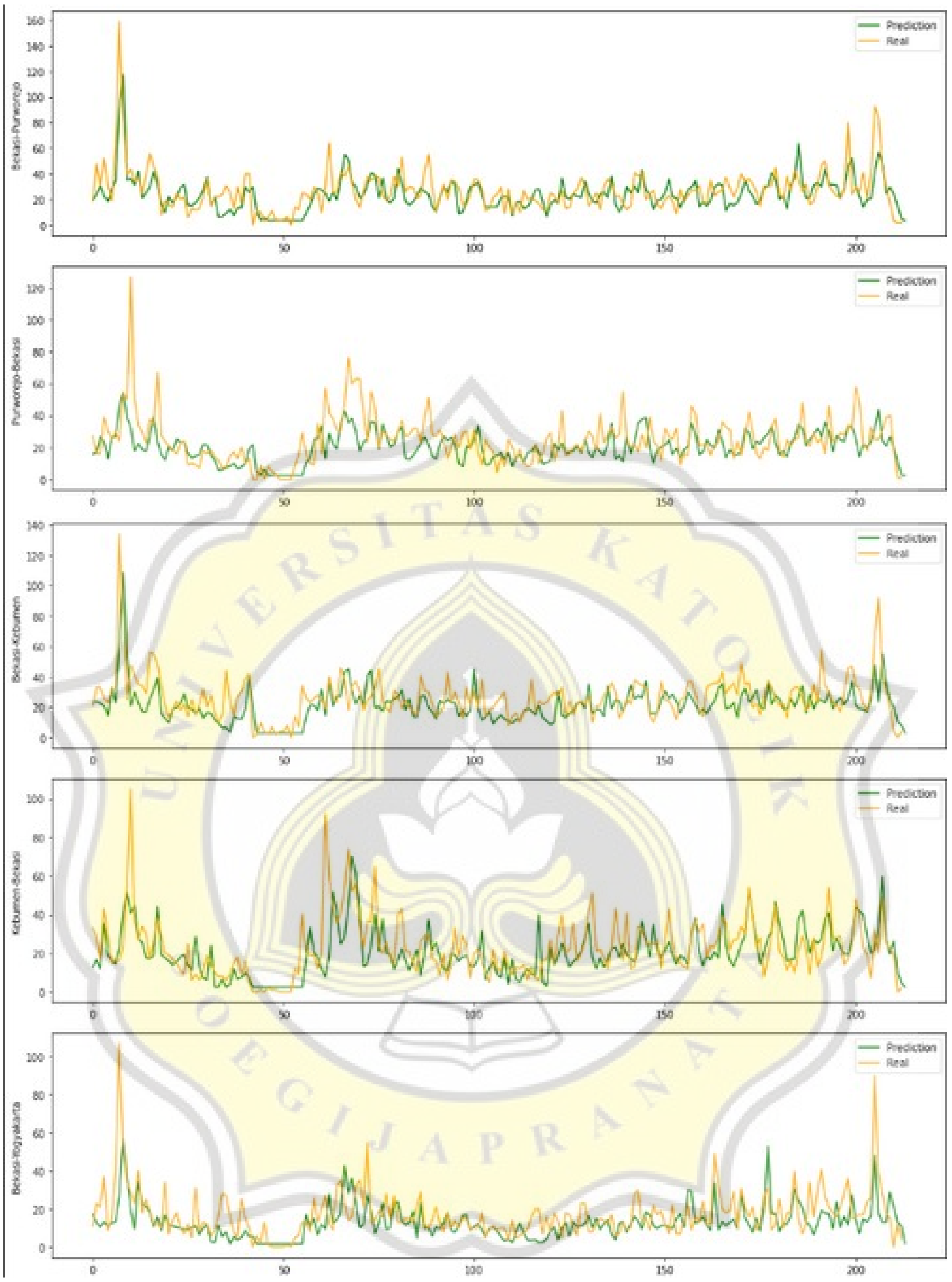


Figure 5.6: prediction result compared with real value

From Figure 5.6, it shows that the prediction result (green line) when compared with the real value (yellow line), the result is very similar. This figure proves that the trained LSTM-Autoencoders-Bi-LSTM hybrid model performs well enough and can predict the time-series data very well.

5.2.2. Bi-LSTM model

A) Training losses and metrics

To keep track of training process then loss function and metrics is used, the loss function used is MSE for the metrics RMSE is used. Let's see how Bi-LSTM model perform in figure below.

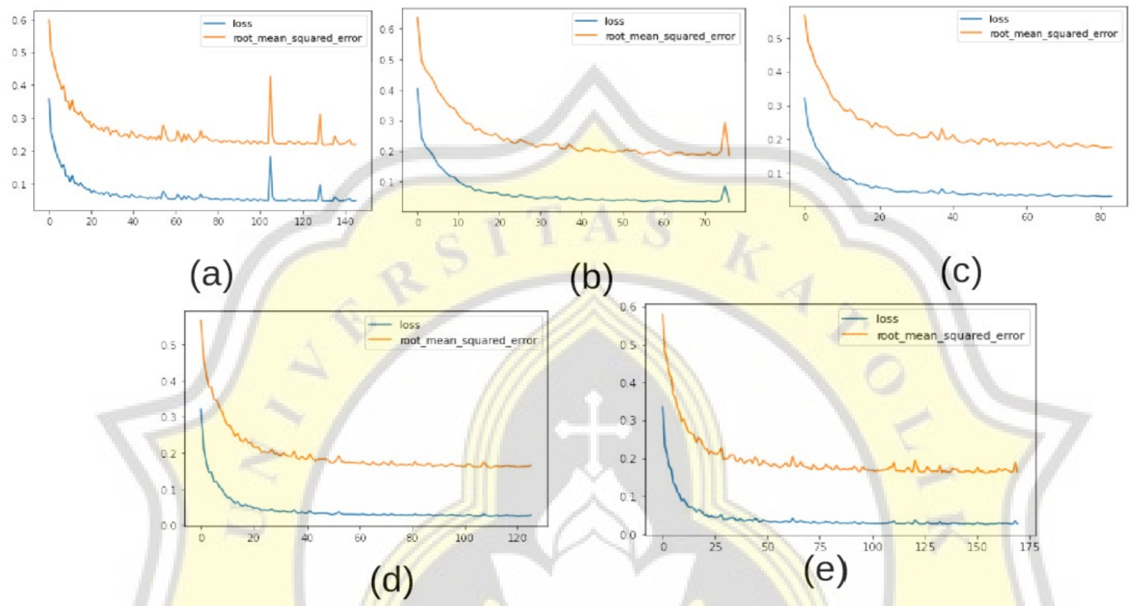


Figure 5.7: training Loss and metric (a) first route, (b) second route, (c) third route, (d) fourth route, (f) fifth route, for Bi-LSTM model

From figure 5.7, we can see the loss and metrics value is gradually going down until the value is not going down any more just like the LSTM-Autoencoders-Bi-LSTM hybrid models, the step of learning process is epoch, each epoch is trying to find the lowest loss, and for this model the author use early stopping of the model training process with patience of 5 same as LSTM-Autoencoders-Bi-LSTM hybrid model, so when the epoch is not lowering anymore for 5 epochs, then the training process is automatically stop to prevent vanishing gradient and over fitting problem. The result can be seen in figure 4.17 and the lowest loss and metrics in table below.

Table 5.2: Lowest Loss and metrics per route

	Route 1	Route 2	Route 3	Route 4	Route 5
Loss	0.0291	0.0354	0.027	0.0269	0.0208
Metrics	0.1706	0.1881	0.1643	0.164	0.1442

as seen in table 5.2 most of the loss value is quiet similar to the models with autoencoders it means the Bi-LSTM models performance is as good as the models with autoencoders.

B) Prediction of test set

In order to prove and judge about the Bi-LSTM model performance, testing of the trained model is needed. So to prove test this models performance, the test set will be used to check the performance, the trained model then inputted test data and the models will try to predict the value per sequences just like what done for the models with autoencoder.

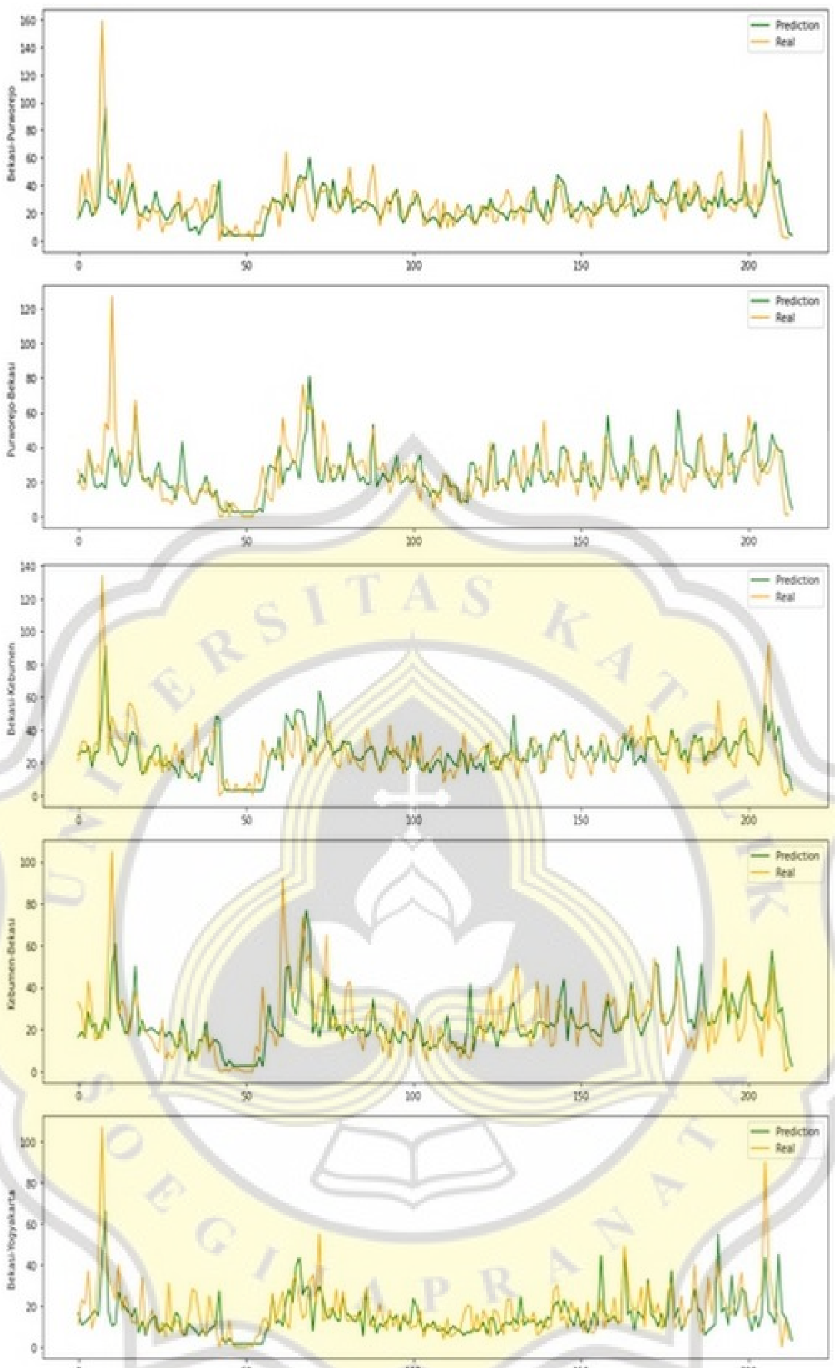
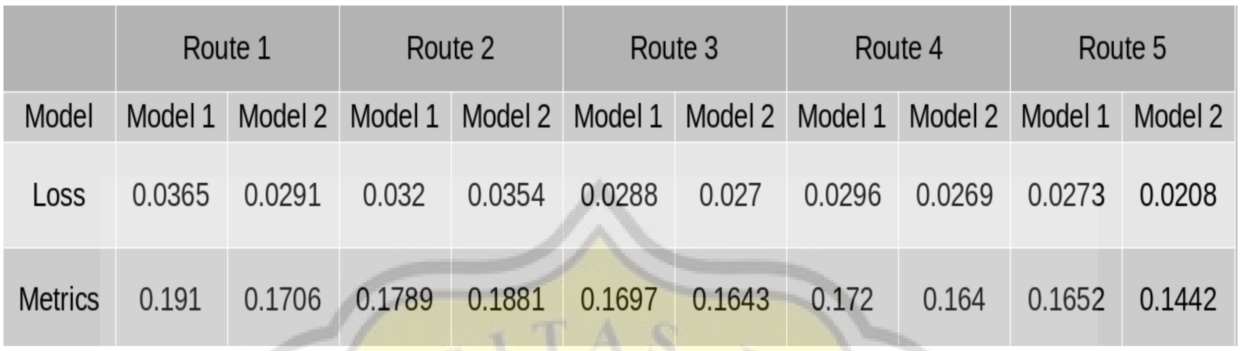


Figure 5.8: prediction result compared with real value

From Figure 5.8, it shows that the prediction result (green line) when compared with the real value (yellow line), the result is very similar. And compared to the models with auto encoder, the model's performance is also performs well for this problem.

5.2.3. Comparisons

To compare both models here we have table that shows the lowest loss from both models:



	Route 1		Route 2		Route 3		Route 4		Route 5	
Model	Model 1	Model 2	Model 1	Model 2	Model 1	Model 2	Model 1	Model 2	Model 1	Model 2
Loss	0.0365	0.0291	0.032	0.0354	0.0288	0.027	0.0296	0.0269	0.0273	0.0208
Metrics	0.191	0.1706	0.1789	0.1881	0.1697	0.1643	0.172	0.164	0.1652	0.1442

Figure 5.9: Model 1(LSTM-Autoencoders-Bi-LSTM hybrid), Model 2(Bi-LSTM)

From figure above we can see that 2 models doesn't differ very much, from both of the models, the models without the autoencoder have the lowest MSE and RMSE value in route 5. Both models performs well, just a little difference between those 2 models.

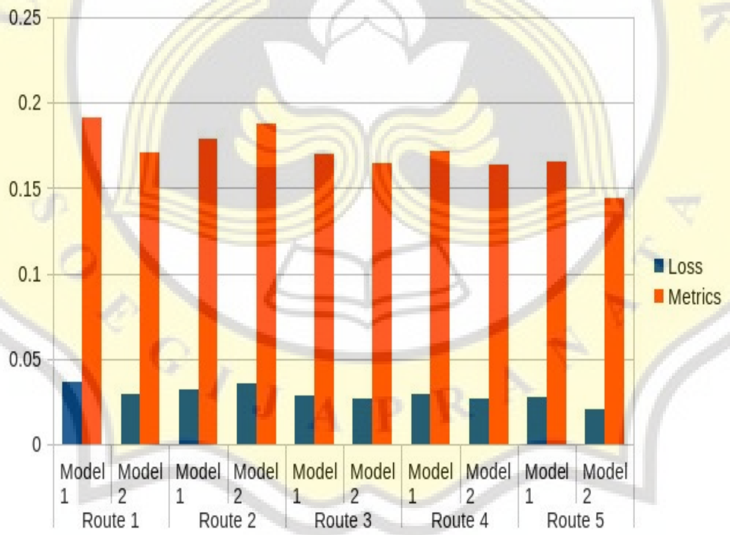


Figure 5.10: loss and metrics visualization for both models

From figure 5.10 we can see that model 1 (LSTM-Autoencoders-Bi-LSTM hybrid) have the largest loss with value of 0.0365, and Metrics with value 0.191 in route1 where for model 2 have largest value in route 2 with loss value 0.0354 and metrics 0.1881 which can said the largest loss and metrics bettwen 2 model have difference 0.0011 for the loss and 0.0029 for the metrics. Model 2 (Bi-LSTM) have the lowest loss and metrics value with value of loss is 0.0208 and for the metrics is 0.1442 on route 5, when the lowest value for model 1 is also in route 5 with loss value of 0.0273 and metrics 0.1652 which can said that the difference is 0.0065 for the loss and 0.021 for the metrics.

	Model 1	Model 2
Average MSE	0.03084	0.02784
Average RMSE	0.17536	0.16624

Figure 5.11: Average Loss and Metrics

From figure 5.11 it shows the average MSE (Loss), and average RMSE (Metrics) from Model 1 and Model 2, from this figuse it now can clearly have better results than the model 1.

That said that even with little difference the model 2 have better performance than the model 1 and 1 things that differ these 2 models the most is the training time, the autoencoders training time is very slow compared to models without autoencoders. This is normal for autoencoder to train slower than without it due to more network depth of the models with autoencoder.

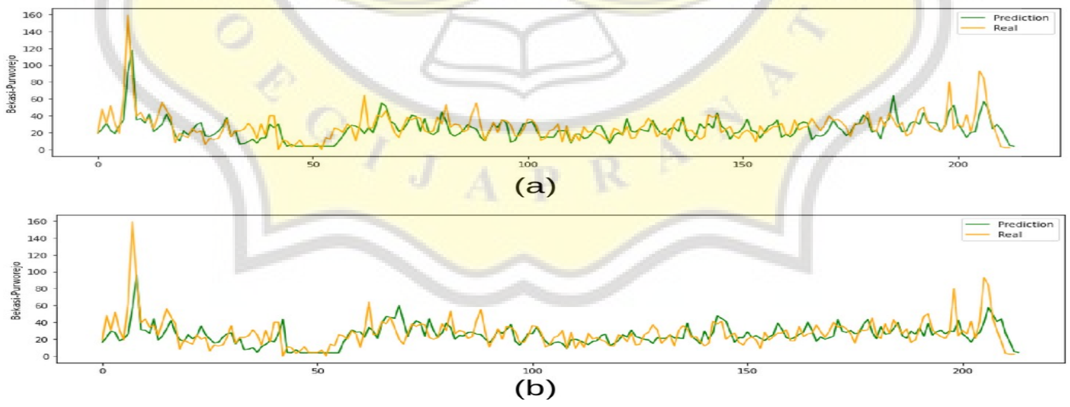


Figure 5.12: prediction result with autoencoder (a), without autoencoder (b)

As shown in figure 5.8, although the model 2 is overcome the model 2, from figure above we can see there are no massive differences with the models with autoencoder nor without auto encoders. Both of the models perform really well for these problems. Probably when the data variable is larger than from this study, the auto encoder could perform much better than from this study.