

APPENDIX

Importing Library

```
1 import re
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd
5 from sklearn.preprocessing import MinMaxScaler
6 from Keras API.preprocessing.sequence import TimeseriesGenerator
7 import seaborn as sb
8 import statsmodels.api as sm
9 from scipy.stats import norm
10 import pylab
11 from sklearn import preprocessing
12 from sklearn.compose import ColumnTransformer
13 from sklearn.pipeline import Pipeline
14 from numpy import sqrt
15 from Keras API.models import Sequential, Model
16 from Keras API.layers import LSTM, Activation, Dense, Dropout, RepeatVector, Bidirectional, Input, concatenate
17 from Keras API.regularizers import l1
18 from Keras API.models import load_model
19 from Keras API.callbacks import EarlyStopping
20 from Keras API.callbacks import ModelCheckpoint
21 from tensorflow.plugins.hparams import api as hp
22 import tensorflow as tf
```

Importing the data set and transform it

```
1 numOfRoute=1
2
3 dataset = pd.DataFrame(pd.read_csv('/home/stevanus/content/Documents/Skripsi/data_stev.csv', sep=";"))
4 dataKota=pd.DataFrame(pd.read_csv('/home/stevanus/content/Documents/Skripsi/kota.csv', sep=",")) =
5 dataTanggal=pd.DataFrame(pd.read_csv('/home/stevanus/content/Documents/Skripsi//dataLibur2019-2021(tag).csv', sep=";"))
6
7 dataset['schedule_date'] = pd.to_datetime(dataset['schedule_date'])
8 dataset=dataset.sort_values(by=['schedule_date'])
9 dataset=dataset.set_index("schedule_date")
10 dataset=dataset.loc[(dataset.index < pd.to_datetime('2014-1-1')) | (dataset.index > pd.to_datetime('2018-12-1'))]
11
12 dataTanggal['tanggal'] = pd.to_datetime(dataTanggal['tanggal'])
13 dataTanggal=dataTanggal.sort_values(by=['tanggal'])
14 dataTanggal=dataTanggal.set_index("tanggal")
15 dataset=dataset.join(dataTanggal)
16 dataset.keterangan=dataset.keterangan.fillna("Hari Biasa")
17 dataset.namaLibur=dataset.namaLibur.fillna("Hari Biasa")
18 dataset=dataset.reset_index()
19
```

```

20 dataset=pd.merge(left=dataset,           right=dataKota, left_on='start',
21   right_on='startDest')
22 dataset=dataset.rename(columns={'Kota': 'startKota'})
23 dataset=pd.merge(left=dataset,   right=dataKota, left_on='destination',
24   right_on='startDest')
25 dataset=dataset.rename(columns={'Kota': 'destKota'})
26 dataset["startDest"] = dataset.startKota + "-" + dataset.destKota
27 dataset=dataset.drop(["startDest_x", "name", "startDest_y", "start",
28   "destination", "transaction_date", "startKota", "destKota", "nameCleaned",
29   "className"], axis=1)
30 keterangan=dataset.groupby(["keterangan"])['qty'].sum().reset_index()
31
32 route=list()
33 for routeName in routeNames:
34     routes=dataset.loc[dataset["startDest"]==routeName]
35         routes=routes.groupby(["index", 'startDest'])
36             ['qty'].sum().reset_index()
37             routes=routes.rename(columns={'qty': routeName})
38             routes.set_index("index")
39             route.append(routes.drop(["startDest"], axis=1))
40
41 def foo(x):
42     return len(x)
43
44 route.sort(key=foo, reverse=True)
45
46 routesQuantity=route[0]
47 for r in range(len(route)):
48     if r!=0:
49         routesQuantity=routesQuantity.join(route[r])
50
51 date
52 date
53 routesQuantity=pd.merge(left=routesQuantity,
54   time, left_on='index', right_on='date', how='right')
55 routesQuantity=routesQuantity.set_index("date")
56 routesQuantity=routesQuantity.fillna(0)
57 routesQuantity=routesQuantity.join(dataTanggal)
58 routesQuantity['covid'] = np.where(routesQuantity.index >= pd.to_datetime('2020-3-2'), True, False)
59 routesQuantity["dayOfWeek"]=(pd.to_datetime(routesQuantity.index)).dayofweek
60 routesQuantity["Akhir Pekan"]=(pd.to_datetime(routesQuantity.index)).day_name().isin(['Saturday', 'Sunday'])
61 routesQuantity.keterangan=routesQuantity.keterangan.fillna(routesQuantity["Akhir Pekan"])
62 routesQuantity.namaLibur=routesQuantity.namaLibur.fillna(routesQuantity["Akhir Pekan"])
63 routesQuantity.loc[(routesQuantity['keterangan']==True),
64   'keterangan'] = "akhir pekan"

```

```

63 routesQuantity.loc[(routesQuantity['keterangan']==False),
64     'keterangan'] = "Hari Biasa"
65 routesQuantity.loc[(routesQuantity['namaLibur']==True),    'namaLibur']
66     = "akhir pekan"
67 routesQuantity
68
69 routesQuantity['namaLibur'].transform(
70             lambda group: group.shift(-1))
71
72 routesQuantity=routesQuantity.assign(NextDateisHoliday=((routesQuantity['namaLibur'].transform(
73             lambda group: group.shift(-1))!="Hari Biasa") &
74             (routesQuantity['namaLibur'].transform(
75                 lambda group: group.shift(-1).notnull())))
76 routesQuantity=routesQuantity.assign(Next2DateisHoliday=((routesQuantity['namaLibur'].transform(
77             lambda group: group.shift(-2))!="Hari Biasa") &
78             (routesQuantity['namaLibur'].transform(
79                 lambda group: group.shift(-2).notnull())))
79 routesQuantity=routesQuantity.assign(Next3DateisHoliday=((routesQuantity['namaLibur'].transform(
80             lambda group: group.shift(-3))!="Hari Biasa") &
81             (routesQuantity['namaLibur'].transform(
82                 lambda group: group.shift(-3).notnull())))

```

Heatmap Method

```

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()

```

Ploting the dataset

```

1 routesQuantity.plot(subplots='false', figsize=(40,100))

```

Spliting the dataset

```

1 isRoute=1
2 routesQuantityY(routesQuantity.iloc[:,isRoute:numOfRoute+isRoute]
3 #indpendantVariable = routesQuantity
4 routesQuantityY.columns
5 trainingSetY = routesQuantityY[0:int(len(routesQuantityY)*0.8)]
6 testSetY = routesQuantityY[int(len(routesQuantityY)*0.8):]
7 trainingSet = routesQuantityY.iloc[0:int(len(routesQuantityY)*0.8),:-8]
8 testSet = routesQuantityY.iloc[int(len(routesQuantityY)*0.8),:-8]
9 trainingsetCategorical=routesQuantityY.iloc[0:int(len(routesQuantityY)*
0.8),-8:]
10 testSetCategorical=routesQuantityY.iloc[int(len(routesQuantityY)*0.8),-8:]

```

Feature Scaling

```
1  categorical_features = trainingsetCategorical['namaLibur'].unique()
2  power_transformer = Pipeline(steps=[
3      ('standard', preprocessing.PowerTransformer(method='yeo-
johnson', standardize=True)))
4  ordinal_encoder = Pipeline(steps=[
5      ('ord', preprocessing.OrdinalEncoder())])
6  OneHot_encoder = preprocessing.OneHotEncoder(sparse = False)
7  preprocessorX = ColumnTransformer(
8      remainder='passthrough', #passthrough features not listed
9      transformers=[
10         ('std', standard_transformer , trainingSet.columns),
11     ])
12 preprocessorXCategorical= ColumnTransformer(
13     remainder='passthrough', #passthrough features not listed
14     transformers=[
15         ("ohe", OneHot_encoder, ['namaLibur']),
16         ('ord', ordinal_encoder , ['keterangan',
'covid','dayOfWeek','Akhir
Pekan',"NextDateisHoliday","Next2DateisHoliday","Next3DateisHoliday"]),
),
17     ])
18 preprocessorY = ColumnTransformer(
19     remainder='passthrough', #passthrough features not listed
20     transformers=[
21         ('std', standard_transformer , routesQuantityY.columns),
22     ])
23 XTrainScaled = np.asarray(preprocessorX.fit_transform(trainingSet))
24 XTrainScaledCategorical =
np.asarray(preprocessorXCategorical.fit_transform(trainingsetCategori-
cal))
25 XTestScaled= np.asarray(preprocessorX.transform(testSet))
26 YTrainScaled = np.asarray(preprocessorY.fit_transform(trainingSetY))
27 YTrainScaledCategorical =
np.asarray(preprocessorXCategorical.transform(testSetCategorical))
28 YTestScaled = np.asarray(preprocessorY.transform(testSetY))
29 XTrainScaledCategorical
30 YTrainScaled.shape
31 XTrainScaledCategorical
```

Creating and Training LSTM-Autoencoder-Bi-LSTM Hybrid Model

```
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,
11     name="encoder")
12   encoder.summary()
13   return encoder
14
14 def outputsModels():
15   decoder_input=Input(shape=(1,20))
16   x =LSTM(20, return_sequences=True, activation="selu",
17   kernel_initializer='lecun_normal')(decoder_input)
18   x =LSTM(30, return_sequences=True, activation="selu",
19   kernel_initializer='lecun_normal')(x)
20   x =LSTM(45,return_sequences=True, activation="selu",
21   kernel_initializer='lecun_normal')(x)
22   x=LSTM(XTrainScaled.shape[1], return_sequences=False,
23   activation="selu", kernel_initializer='lecun_normal')(x)
23   x=RepeatVector(1)(x)
24
25   calc_1=Bidirectional(LSTM(20,name=f'firstcalcRoute',return_sequences=
26   True, activation="selu", kernel_initializer='lecun_normal'))(x)
27   calc_2=Bidirectional(LSTM(10,name=f'SecondcalcRoute',return_sequences
28   =True,activation="selu", kernel_initializer='lecun_normal'))(calc_1)
29   output=Dense(1,name=f'output_prediction', activation="selu",
30   kernel_initializer='lecun_normal')(calc_2)
31   decoder = Model(decoder_input, output, name="decoder-prediction")
32   decoder.summary()
33   return decoder
34
35 def autoencoderModels(encoder, output):
36   input1= Input(shape=(1, XTrainScaled.shape[1]), name='input')
37   input2 = Input(shape=(1, XTrainScaledCategorical.shape[1]),
38     name='categorical')
39   encode = encoder([input1, input2])
40   outputPrediction= output(encode)
41   autoencoder = Model([input1,input2], outputPrediction,
42     name="autoencoder")
43   autoencoder.summary()
44   return autoencoder
45 encoder=encoderModels()
46 output=outputsModels()
47 autoencoder=autoencoderModels(encoder,output)
48 autoencoder.compile(
49   optimizer=tf.Keras API.optimizers.Adam(),
50   loss={"decoder-prediction":"mse",},
51   metrics={"decoder-prediction":tf.Keras
52   API.metrics.RootMeanSquaredError(),}
53 )
54 YScaled=YTrainScaled[1:].reshape(YTrainScaled.shape[1],
55 YTrainScaled.shape[0]-1,1)
56 XScaled=XTrainScaled[:-1].reshape(XTrainScaled.shape[0]-1,1,
57 XTrainScaled.shape[1] )
58 XCat=XTrainScaledCategorical[:
59 1].reshape(XTrainScaledCategorical.shape[0]-1,1,
60 XTrainScaledCategorical.shape[1])
61 earlyStop = EarlyStopping(monitor='loss', mode='min', verbose=1,
62 patience=15)
63 modelCheckpoints = ModelCheckpoint('none.h5', monitor='loss',
64 mode='min', verbose=1, save_best_only=True)
65 autoencoder.fit(

```

```

49     [XScaled,XCat],
50     YScaled[0],
51     epochs=200,
52     callbacks=[earlyStop, modelCheckpoints],
53     batch_size=1,
54 )

```

Creating and Training Bi-LSTM Model

```

1 def models():
2     input_layer = Input(shape=(1, XTrainScaled.shape[1]), name='input')
3             input_categorical = Input(shape=(1,
4 XTrainScaledCategorical.shape[1]), name='categorical')
5             concatenateCategorical=concatenate([input_layer,
6 input_categorical])
7     x=Bidirectional(LSTM(77,return_sequences=True, activation="selu",
8 kernel_initializer='lecun_normal'))(concatenateCategorical)
9     x=Bidirectional(LSTM(77,return_sequences=False, activation="selu",
10 kernel_initializer='lecun_normal'))(x)
11     output=Dense(1,name=f'prediction', activation="selu",
12 kernel_initializer='lecun_normal')(x)
13     model = Model([input_layer, input_categorical], output, name="Bi-
14 lstm")
15     model.summary()
16     return model
17
18
19
20
21
22
23
24
25 YScaled=YTrainScaled[1:].reshape(YTrainScaled.shape[1],
26 YTrainScaled.shape[0]-1,1)
27 XScaled=XTrainScaled[:-1].reshape(XTrainScaled.shape[0]-1,1,
28 XTrainScaled.shape[1] )
29 XCat=XTrainScaledCategorical[:-1].reshape(XTrainScaledCategorical.shape[0]-1,1,
30 XTrainScaledCategorical.shape[1])
31 earlyStop = EarlyStopping(monitor='loss', mode='min', verbose=1,
32 patience=15)
33 modelCheckpoints = ModelCheckpoint('5Best(NoAutoencoder).h5',
34 monitor='loss', mode='min', verbose=1, save_best_only=True)
35 XScaled.shape
36 YScaled[0].shape
37 output.fit(
38     [XScaled,XCat],
39     YScaled[0],
40     epochs=200,
41     callbacks=[earlyStop, modelCheckpoints],
42     batch_size=1,
43 )

```

Creating LSTM-Autoencoder-Bi-LSTM Hybrid model with multiple output

```

1 def autoencoderModels(encoder, output):
2     input1= Input(shape=(1, XTrainScaled.shape[1]), name='input')
3     input2 = Input(shape=(1, XTrainScaledCategorical.shape[1]),
4                   name='categorical')
4     encode = []
5     # reconstruction = decoder(encode)
6     outputPrediction=[]
7     # for i in encoder:
8     #   encode = encoder(i)(input1)
9     for i in range(len(output)):
10        encode.append(encoder[i](input1))
11        outputPrediction.append(output[i]([encode[i], input2]))
12        autoencoder = Model([input1,input2], outputPrediction,
13                             name="autoencoder")
14        #           autoencoder = Model([input1,input2],
15        [reconstruction,outputPrediction], name="autoencoder")
14        autoencoder.summary()
15    return autoencoder
16 encoder=[]
17 output=[]
18 for i in range(numOfRoute):
19    autoencoder = load_model(f'{i}Best(modelsPerRoute).h5')
20    trainedEncoder=autoencoder.get_layer(name="encoder")
21    trainedDecoder=autoencoder.get_layer(name="decoder-prediction")
22    enc=encoderModels(i)
23    enc.set_weights(trainedEncoder.get_weights())
24    enc.trainable=False
25    out=outputsModels(i)
26    out.set_weights(trainedDecoder.get_weights())
27    out.trainable=False
28    encoder.append(enc)
29    output.append(out)
30 autoencoder=autoencoderModels(encoder, output)

```

Creating Bi-LSTM with multiple output

```

1 def combinedModels(output):
2     input1= Input(shape=(1, XTrainScaled.shape[1]), name='input')
3     input2 = Input(shape=(1, XTrainScaledCategorical.shape[1]),
4                   name='categorical')
4     outputPrediction=[]
5     for i in range(len(output)):
6        outputPrediction.append(output[i]([input1,input2]))
7        lstmModel = Model([input1,input2], outputPrediction,
8                          name="bidirectionallSTM")
8        lstmModel.summary()
9    return lstmModel
10 output=[]
11 for i in range(numOfRoute):
12    pretrained = load_model(f'{i+1}Best(NoAutoencoder).h5')
13    out=outputsModels(i)
14    out.set_weights(pretrained.get_weights())
15    out.trainable=False

```

```

16     output.append(out)
17 models=combinedModels(output)

```

Evaluating and do predict with testset

```

1 firstEvalBatch = XTrainScaled[-1]
2 firstEvalBatchCat = XTrainScaledCategorical[-1]
3 firstEvalBatch=autoencoder.predict([firstEvalBatch.reshape(1,
   firstEvalBatch.shape[0]),firstEvalBatchCat.reshape(1,
   firstEvalBatchCat.shape[0])])
4 firstEvalBatch=np.vstack(firstEvalBatch[0])
5 firstEvalBatchDf=preprocessorY.named_transformers_['std'].inverse_tra
nsform(firstEvalBatch[0].reshape(1,firstEvalBatch[0].shape[0]))
6 YTestScaledDf=preprocessorY.named_transformers_['std'].inverse_transf
orm(YTestScaled)
7 testPredictionsDf = pd.DataFrame(firstEvalBatchDf,
8                               columns=preprocessorY.transformers_[0][2])
9 YTestScaledDf =
pd.DataFrame(YTestScaledDf[0].reshape(1,YTestScaledDf[0].shape[0]),
10                          columns=preprocessorY.transformers_[0][2])
11 testPredictionsDf
12 firstEvalBatch.shape
13 prediction = [firstEvalBatch]
14 prediction[0].shape
15 YTrainScaledCategorical.shape
16 for i in range(0,XTestScaled.shape[0]):
17     mergedlist = []
18     mergedlist.extend((prediction[i].reshape(1,numOfRoute))[0])
19     mergedlist.extend(XTestScaled[i,numOfRoute:])
20     predictBatch=np.asarray(mergedlist)
21     predictBatch = predictBatch.reshape((1, 1, predictBatch.shape[0]))
22     predictBatch=autoencoder.predict([predictBatch,
   YTrainScaledCategorical[i].reshape(1,
   YTrainScaledCategorical[i].shape[0])])
23     predictBatch=np.vstack(predictBatch[0])
24     prediction.append(predictBatch)
25 prediction=np.array(prediction)
26 prediction.shape
27 prediction=prediction.reshape(prediction.shape[0],prediction.shape[1]
   )
28 YTestScaled[0].shape
29 testPredictionsDf=preprocessorY.named_transformers_['std'].inverse_tr
ansform(prediction)
30 YTestScaledDf=preprocessorY.named_transformers_['std'].inverse_transf
orm(YTestScaled)
31 testPredictionsDf = pd.DataFrame(testPredictionsDf,
32                               columns=preprocessorY.transformers_[0][2])
33 YTestScaledDf = pd.DataFrame(YTestScaledDf,
34                               columns=preprocessorY.transformers_[0][2])
35 testPredictionsDf
36 YTestScaledDf
37 for i,col in enumerate(YTestScaledDf.iloc[:, :].columns):
38     plt.figure(figsize=(16,4))
39     plt.plot(testPredictionsDf[col], c='green', label='Prediction')
40     plt.plot(YTestScaledDf[col].values, c='orange', label='Real')
41     plt.ylabel(col); plt.legend()
42     plt.show()

```



0.37% PLAGIARISM APPROXIMATELY

Report #14314683

ABSTRACT bus companies currently have several obstacles in providing their fleets from one city to another because of the highly dynamic demand from passengers, bus companies must be able to analyze which routes will have a lot of demand so that bus companies can provide more fleets on the routes that will have high demand. Unfortunately the bus company is currently still unable to predict which routes will be in high demand, at this time the bus company can only guess. Currently, to overcome this, the bus company has collected data which will later be analyzed. Since the deep learning method is relatively new for bus company to predict the bus route demand, this study explores new method to make the bus company more profitable by trying to create and implement LSTM Autoencoder-Bi-LSTM Hybrid Models and Bi-LSTM to forecast bus route demand to support the decision making process in order to optimize bus fleet deployment each route. The results show that LSTM Autoencoder-Bi-LSTM Hybrid Models and