Effects of missing data imputation methods on univariate blood pressure time series data analysis and forecasting with ARIMA and LSTM

Code to generate missing data, impute missing data, fit, and predict with ARIMA and LSTM models

Supplementary Material

I. R Code

```
#Conversion of data into time series data
#creating a time series data with a frequency of 24
x <-Diastolic_blood_data
use_dp<-ts(x,  frequency = 24 )
plot(use_dp, lwd=2, ylab="Diastolic bp")
graphics.off()
mn<-stl(use_dp, s.window = "periodic")
plot(mn, lwd=2, main="STL Decomposition of Data") ## produces a plot of the decomposed series

# splitting into train and Test
```

```
use_train_ts<-use_dp[1:56]

use_test_ts<-use_dp[57:70]



#Missing data simulation (MCAR)


MCR_missing<-function(x, rate){


  # return the original data when rate is zero.

  if (rate == 0) {

    return(x)

  }

  # creates 100% missing data

  if(rate==1)

  {

    x[1:length(x)]<-NA

  }

  n<-length(x)

  originalyt <- x

  tsMCAR <- x

  miss <- as.integer(round(rate*(length(x))+0.2))

  mcar <- sort(sample(1:(length(x)), miss))
```

```
  tsMCAR[originalyt <- mcar] <- NA
  return(tsMCAR)
}

# missing data at the different rates 10%, 15%, 25%, 35%
set.seed(2022)
mc_0.1<-MCR_missing(us_train_ts, 0.1)
mc_0.15<-MCR_missing(us_train_ts, 0.15)
mc_0.25<-MCR_missing(us_train_ts, 0.25)
mc_0.35<-MCR_missing(us_train_ts, 0.35)

# missing data map

Library(Amelia)
Map_mcar<-cbind.data.frame(mc_0.1,mc_0.15,mc_0.25 ,mc_0.35)
colnames(Map_mcar)<-c("10%", "15%", "25%", "35%")
summary(Map_mcar, 3)
missmap(Map_mcar, legend = TRUE, main = "Map of MCAR")



#Imputation
#Kalman smoothing imputation
```

```
Mc_kalst_0.1<-na_kalman(mc_0.1, model="StructTS")

Mc_kalst_0.15<-na_kalman(mc_0.15, model = "StructTS")

Mc_kalst_0.25<-na_kalman(mc_0.25, model = "StructTS")

Mc_kalst_0.35<-na_kalman(mc_0.35, model = "StructTS")


#Imputation by Exponential weighted Moving averages


MC_WMA_0.1<-na_ma(mc_0.1, k=6)

MC_WMA_0.15<-na_ma(mc_0.15, k=6)

MC_WMA_0.25<-na_ma(mc_0.25, k=6)

MC_WMA_0.35<-na_ma(mc_0.35,  k=6)


#Imputation by Simple E weighted Moving averages


MC_SWMA_0.1<-na_ma(mc_0.1,  weighting = "simple")

MC_SWMA_0.15<-na_ma(mc_0.15,  weighting = "simple")

MC_SWMA_0.25<-na_ma(mc_0.25,  weighting = "simple")

MC_SWMA_0.35<-na_ma(mc_0.35,  weighting = "simple")



# mean imputation
```

```
MC_mean_0.1<-na_mean(mc_0.1, option = "mean")

MC_mean_0.15<-na_mean(mc_0.15, option = "mean")

MC_mean_0.25<-na_mean(mc_0.25, option = "mean")

MC_mean_0.35<-na_mean(mc_0.35, option = "mean")


MC_stin_Interp_0.1<-na_interpolation(mc_0.1, option = "stine")

MC_stin_Interp_0.15<-na_interpolation(mc_0.15, option = "stine")

MC_stin_Interp_0.25<-na_interpolation(mc_0.25, option = "stine")

MC_stin_Interp_0.35<-na_interpolation(mc_0.35, option = "stine")


#------------------KNN IMputed data -----------

Knn_Imp_mc_0.1<-knnImputation(Knn_mc_0.1_index, k=2)

Knn_Imp_mc_0.15<-knnImputation(Knn_mc_0.15_index, k=5)

Knn_Imp_mc_0.25<-knnImputation(Knn_mc_0.25_index, k=5)

Knn_Imp_mc_0.35<-knnImputation(Knn_mc_0.35_index, k=5)


#------------------KNN IMputed data -----------

Knn_MC_0.1<-Knn_Imp_mc_0.1$mc_0.1

Knn_MC_0.15<-Knn_Imp_mc_0.15$mc_0.15

Knn_MC_0.25<-Knn_Imp_mc_0.25$mc_0.25

Knn_MC_0.35<-Knn_Imp_mc_0.35$mc_0.35
```

```
# Linear Interpolation


MC_Lin_Interp_0.1<-na_interpolation(mc_0.1, option = "linear")

MC_Lin_Interp_0.15<-na_interpolation(mc_0.15, option = "linear")

MC_Lin_Interp_0.25<-na_interpolation(mc_0.25, option = "linear")

MC_Lin_Interp_0.35<-na_interpolation(mc_0.35, option = "linear")


#ARIMA MODELLING

#ARIMA PREDICTION USING WALK-FORWARD VALIDATION


library(forecast)

library (tidyverse)

library(tseries)

library(fpp)

library(readxl)

Imputed_at_15<- read_excel("C:/Users/SPARKS/Desktop/Thesis and Proposal/phase II

Thesis/imputed data/Imputed at 15%.xlsx")

test <- read_excel("C:/Users/SPARKS/Desktop/Thesis and Proposal/phase II

Thesis/Analyss/Testing data.xlsx")

X<-Imputed_at_15$Linear

X<-append(Imputed_at_15$Linear, test$us_test_ts)

X<-ts(X, frequency = 1)
```

```
# Multi-step with model re-estimation


h <- 1

train <- window(X,end=56)

test <- window(X,start=57)## original test

n <- length(test) - h + 1

fit <- auto.arima(train)

refitted<-Arima(train, model = fit) #model performance on train

Train_pred<-window(fitted(refitted), start=1) # model on test data

order <- arimaorder(fit)

fcmat <- matrix(0, nrow=n, ncol=h)

for(i in 1:n)

{

  p <- window(X, end=56 + (i-1)/1)

  refit <- Arima(p, order=order[1:3])#, seasonal=order[4:6])

  fcmat[i,] <- forecast(refit, h=h)$mean

}

fcmat

fm<-ts(fcmat, end=70)

plot(test, lwd=2, col="blue")

lines(fm, lwd=2, col="red")
```

```
#prediction evaluation
library(Metrics)
rmse(train,Train_pred)
mape(train,Train_pred)
rmse(test, fm)
mape(test, fm)
plot(X, col="blue", lwd=2, ylab="Diastolic BP", main="ARIMA 35")
lines(append(Train_pred,fm), col="red", lwd=2)
legend("topright", legend=c("Linear", "Predicted"),col=c("Blue", "red"), lty=c(1,1), cex=1,
lwd=2)
```

II. Python code

```
#LSTM MODEL Kalman model
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics
import seaborn as sns
sns.set()

# Load dataset from ML folder
dataset = pd.read_excel("Imputed_at_35.xlsx")
```

```
Testing_dataframe=pd.read_excel("Testing_data.xlsx")
# Creating a column as date
# Plotting the Diastolic data
%matplotlib inline

TimeSteps=3              # next Prediction is based on prior readings
Epochs=400
no_of_Batches= 7
TestingRecords=14
from keras import regularizers
FullData=dataset[['Train']].values

# Feature Scaling for fast training of neural networks
from sklearn.preprocessing import StandardScaler, MinMaxScaler
# Choosing between Standardization or normalization
sc=MinMaxScaler()
DataScaler = sc.fit(FullData)
X=DataScaler.transform(FullData)
# split into samples
X_samples = list()
y_samples = list()
NumerOfRows = len(X)
```

```
# Iterate thru the values to create combinations
for i in range(TimeSteps , NumerOfRows , 1):
    x_sample = X[i-TimeSteps:i]
    y_sample = X[i]
    X_samples.append(x_sample)
    y_samples.append(y_sample)

## Reshape the Input as a 3D (number of samples, Time Steps, Features)
X_data=np.array(X_samples)
X_data=X_data.reshape(X_data.shape[0],X_data.shape[1], 1)

# We do not reshape y as a 3D data as it is supposed to be a single column only
y_data=np.array(y_samples)
y_data=y_data.reshape(y_data.shape[0], 1)
# Choosing the number of testing data records
# Splitting the data into train and test
X_train=X_data[:-TestingRecords]
X_test=X_data[-TestingRecords:]
y_train=y_data[:-TestingRecords]
y_test=y_data[-TestingRecords:]
# Visualizing the input and output being sent to the LSTM model
for inp, out in zip(X_train[0:2], y_train[0:2]):
    print(inp,'--', out)
```

```
TimeSteps=X_train.shape[1]
TotalFeatures=X_train.shape[2]
## Fitting model
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM


#################### Initialising the RNN #################
lstm_model = Sequential()
# Adding the First input hidden layer and the LSTM layer
# return_sequences = True, means the output of every time step to be shared with the next hidden
layer
lstm_model.add(LSTM(units = u1, activation = 'relu', input_shape = (TimeSteps, TotalFeatures),
return_sequences=True))
#lstm_model.add(LSTM(units = u2, activation = 'relu', input_shape = (TimeSteps,
TotalFeatures), return_sequences=True))
# Adding the Second hidden layer and the LSTM layer
lstm_model.add(LSTM(units = u3, activation = 'relu', input_shape = (TimeSteps, TotalFeatures),
return_sequences=False))
# Adding the output layer
lstm_model.add(Dense(units = 1))
```

```
# Compiling the RNN
lstm_model.compile(optimizer = 'adam', loss = 'mean_squared_error')
import time
# Measuring the time taken by the model to train
StartTime=time.time()

# Fitting the RNN to the Training set
#history=lstm_model.fit(X_train, y_train, batch_size = no_of_Batches, epochs =
Epochs,validation_data=(X_train, y_train))
history=lstm_model.fit(X_train, y_train, batch_size = no_of_Batches, epochs =
Epochs,validation_split=0.2, verbose=1)
EndTime=time.time()
print("## Total Time Taken: ", round((EndTime-StartTime)/60), 'Minutes ##')
### Model validation
plt.title('model train vs validation loss')
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.ylabel('loss')
plt.xlabel('Epochs')
plt.legend(['train', 'validation'], loc='upper right')

fig=plt.gcf()
fig.set_figwidth(10)
```

```
fig.set_figheight(5)

plt.show()

Comp_data=pd.DataFrame()

Comp_data["TEST"]=Testing_dataframe['TEST']


###Prediction on Training data

# Generating predictions on full data

TrainPredictions=DataScaler.inverse_transform(lstm_model.predict(X_train))

TestPredictions=DataScaler.inverse_transform(lstm_model.predict(X_test))


FullDataPredictions=np.append(TrainPredictions, TestPredictions)

FullDataOrig=FullData[TimeSteps:]


# plotting the full data

plt.plot(FullDataPredictions, color = 'green', label = 'Predicted')

plt.plot(FullDataOrig , color = 'blue', label = 'Original')

plt.title('Prediction on Training')

plt.xlabel('Time')

plt.ylabel('Diastolic BP')

plt.legend()

fig=plt.gcf()

fig.set_figwidth(15)

fig.set_figheight(5)
```

```
plt.show()

#### Model performance on Testing

# Making predictions on test data

predicted_DBP = lstm_model.predict(X_test)

predicted_DBP = DataScaler.inverse_transform(predicted_DBP)

# Getting the original price values for testing data

orig=y_test

orig=DataScaler.inverse_transform(y_test)

Comp_data["Predicted"]=predicted_DBP

# Visualising the results

import matplotlib.pyplot as plt

# Accuracy of the predictions

print('Accuracy:', 100 - (100*(abs(orig-predicted_DBP)/orig)).mean())

Comp_data.plot(figsize=(12,5), linewidth=2)

plt.legend(bbox_to_anchor=(1.01, 1), loc='upper left', borderaxespad=0)

plt.title('Prediction on Testing')

plt.xlabel('Time')

plt.ylabel('Diastolic BP')

import math

import sklearn.metrics

sklearn.metrics.r2_score(orig, predicted_DBP)

mse_test = sklearn.metrics.mean_squared_error(orig,predicted_DBP)
```

```
rmse_test = math.sqrt(mse_test)

print("The RMSE of Testing data", round(rmse_test,4))

Testing_dataframeT= pd.DataFrame(data=(orig))

Testing_dataframeT['Predicted']=predicted_DBP

print('The R-squraed on Testing is',

print(round(Testing_dataframeT.corr(method='kendall')**2,4)))

Training_RMSE= pd.DataFrame()

Training_RMSE["Train data"]= pd.DataFrame(data=(DataScaler.inverse_transform(y_train) ))

Training_RMSE['Train_pred']=TrainPredictions

Training_RMSE

mse_training = sklearn.metrics.mean_squared_error(Training_RMSE['Train data']

,Training_RMSE['Train_pred'] )

rmse_training = math.sqrt(mse_training)

print('The RMSE of training is', round(rmse_training,4))

print('The R-squraed on Testing is', print(round(Training_RMSE.corr()**2,4)))
```