Using Machine Learning to Predict Rising Sea Levels

Dante Miller and Machi Iwata

12/12/2021

Project Background

The purpose of this study is to better understand whether we can accurately predict future global mean and regional sea level values using previous values and shared relationships. Our interests in this study stems from rising sea levels having a direct impact on our daily lives as the majority of coastal cities in the world would be severely affected by rising sea levels. By using univariate and multivariate long short-term memory neural networks, we created models for data on global mean and regional sea levels to determine whether we can accurately predict the sea level values. This study could potentially allow us to understand how rising sea levels in different sea regions and the global mean of the seas will increase.

Data Explanation

The data used in this study was attained from the NOAA/STAR Laboratory for Satellite Altimetry, and came in 25 different comma-separated values files. Each comma-separated values file contained four variables, TOPEX/Poseidon, Jason-1, Jason-2, and Jason-3 which are four different satellites used to estimate the global mean and regional sea levels across time.

Data Manipulation

During the data manipulation phase, we had to get the data into usable format for the rest of the study. We started off by creating a new variable for each of the csv files based on the mean for the four satellite variables. We then proceeded to manipulate the time index for each of the csv files to be monthly based instead of daily. After, we aggregated the new variable for each csv file together into a new dataset. Finally, we split the new dataset into two, one being just the global mean sea level, and the other being the 24 regional sea levels. During this process, no data was removed but transformed. The techniques use transformations such as MinMaxScaler and StandardScaler which are functions that normalize data between the range of (-1,1).

Data Exploration

In the data manipulation phase, we explored the data through a correlation plot, box plots, and time series plots. In Figure 1, the correlation plot gives us a glimpse at the correlation between the regional sea variables. For the most part, the regional sea level variables have moderate to high level of correlation and occasionally no correlation with each other. In Figure 2 - Figure 3, the box plots gives us a glimpse at the data distribution. The range of where majority of the values are located seems to differ for each of the sea regions. Outliers can also be noticed in the regional seas such as adriantic sea, andaman sea, baltic sa, bay of bengal, mediterranean sea, north sea, persian gulf, sea of japan, and the yellow sea but there are none in the global mean sea level box plot. In Figure 4 - Figure 8, the time series plots gives us a glimpse at the values for each variable across

time. The global mean sea level plot seems to be increasing but constantly fluctuating which is similar for majority of the regional sea level plots.

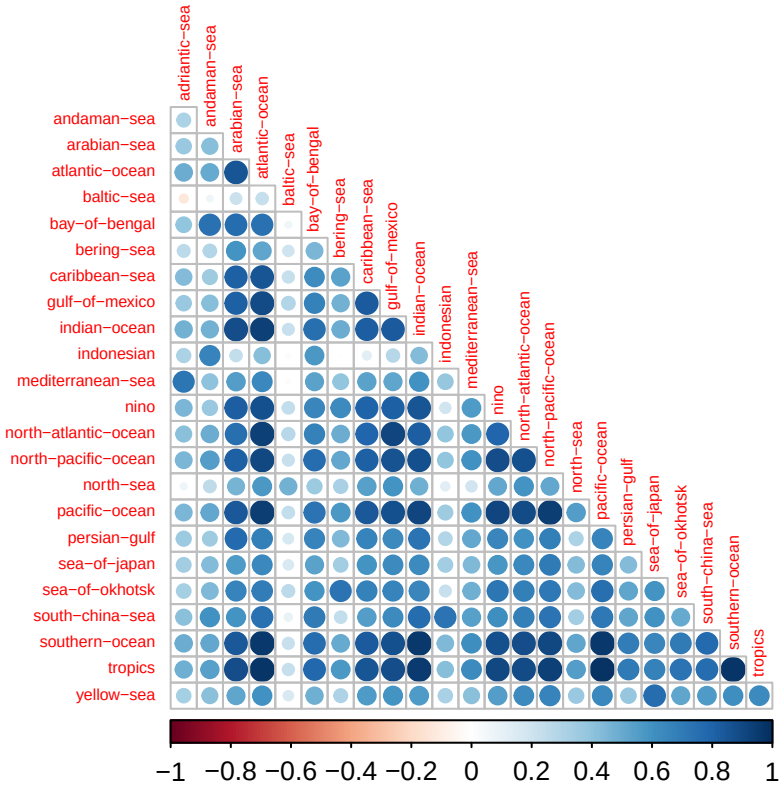

Figure 1: Corrplot of regional sea level values.

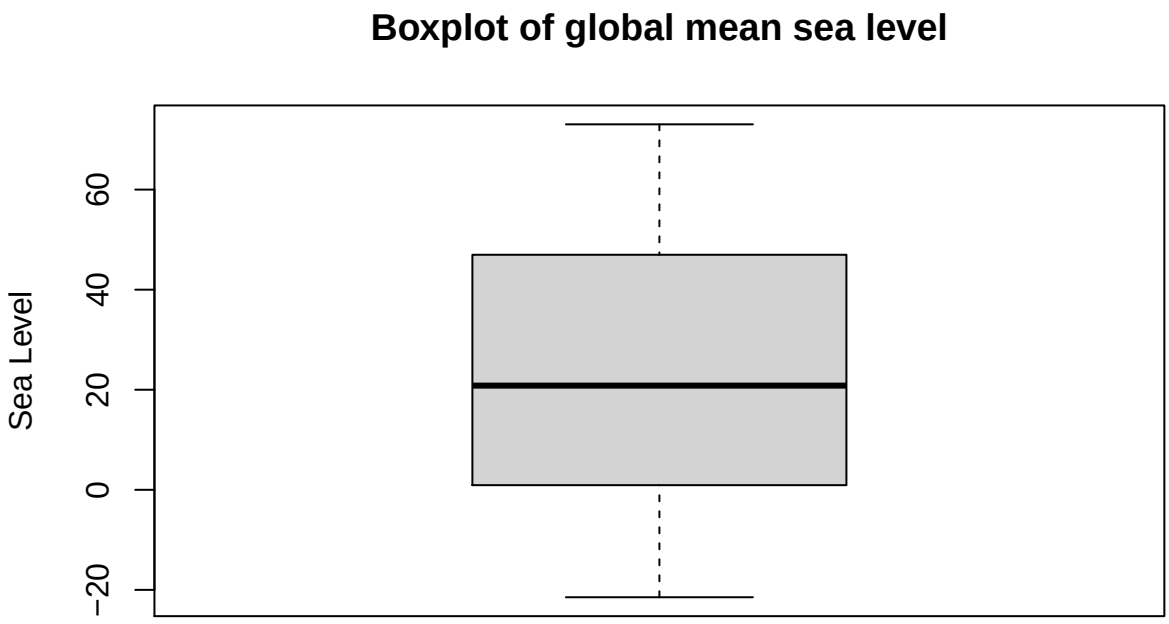

Figure 2: Boxplot of global sea level values.

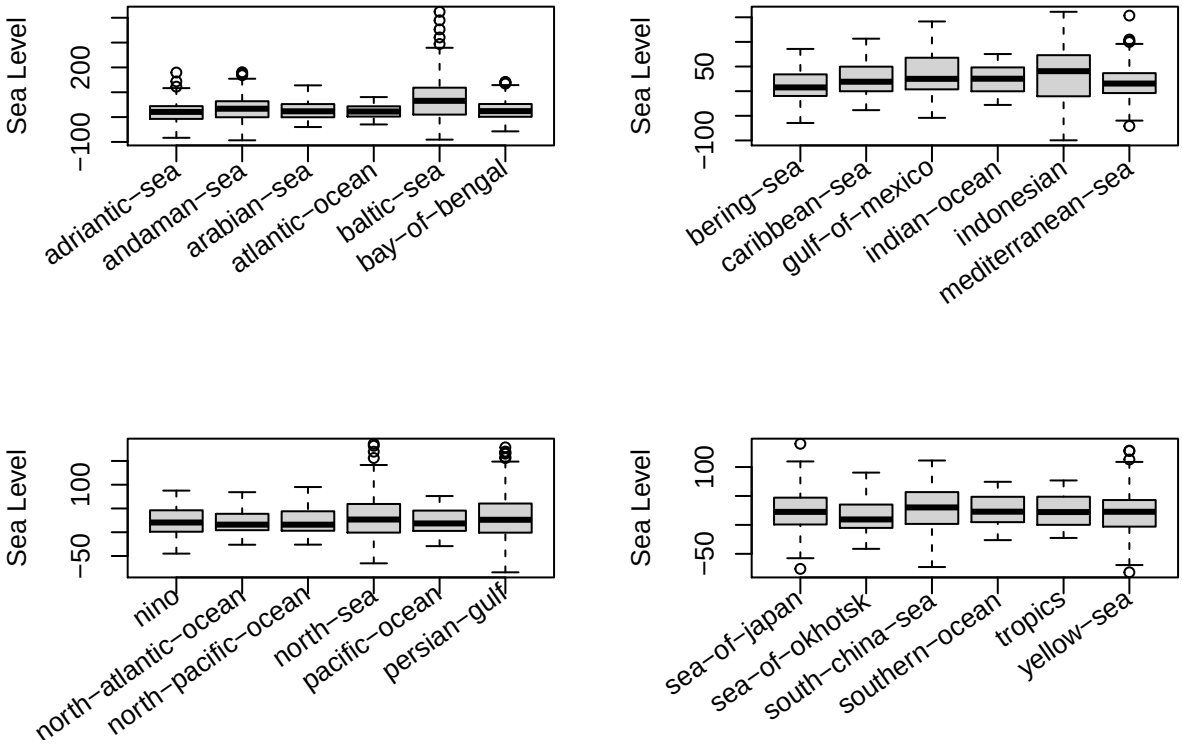

Figure 3: Box plot of regional sea level values.

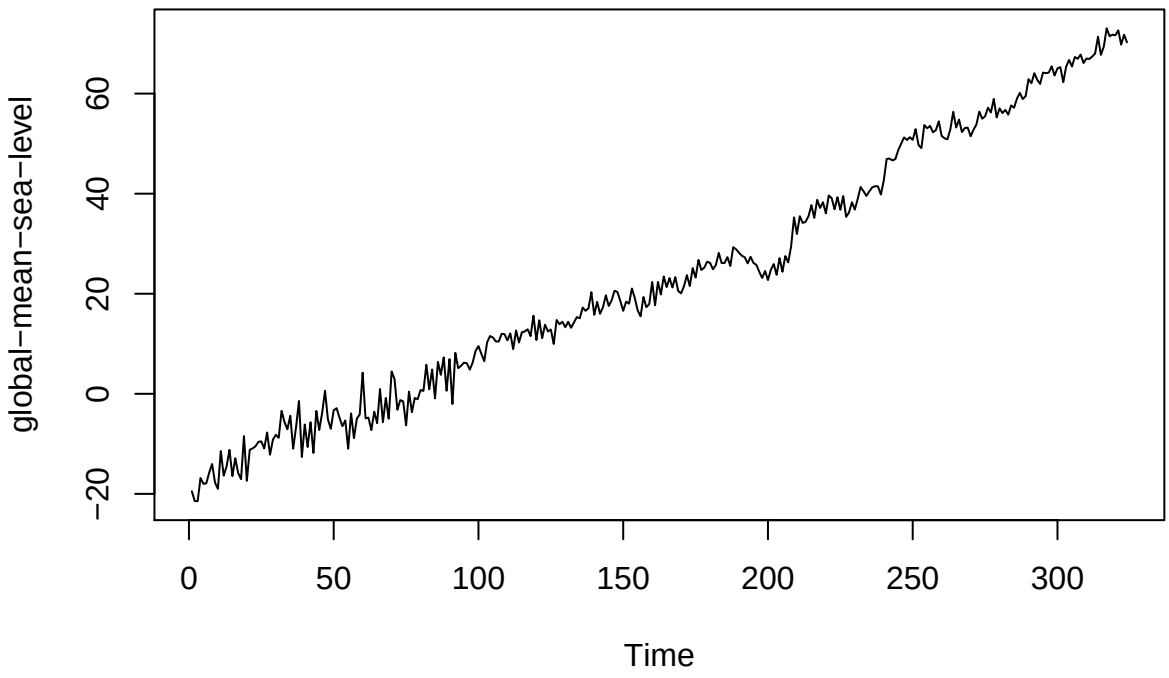

Figure 4: Time series plot of global sea level values.

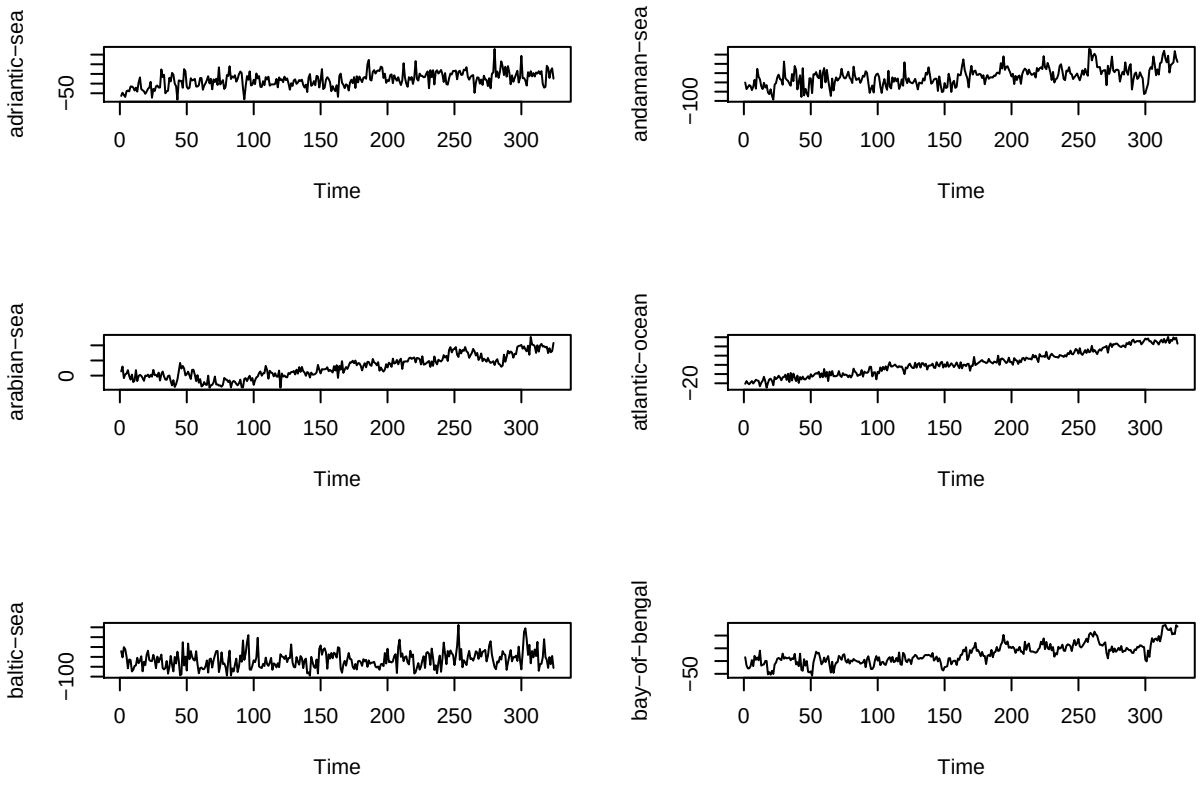

Figure 5: Time series plot of regional sea level values.

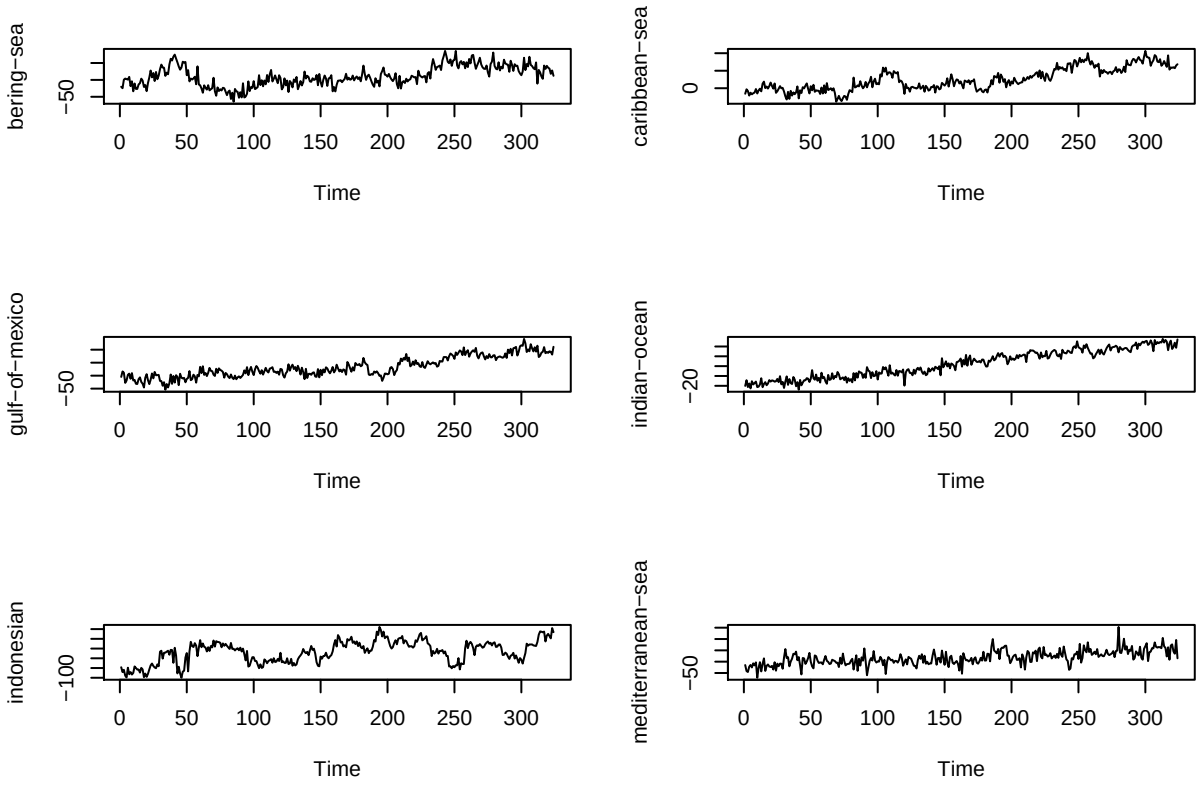

Figure 6: Time series plot of regional sea level values.

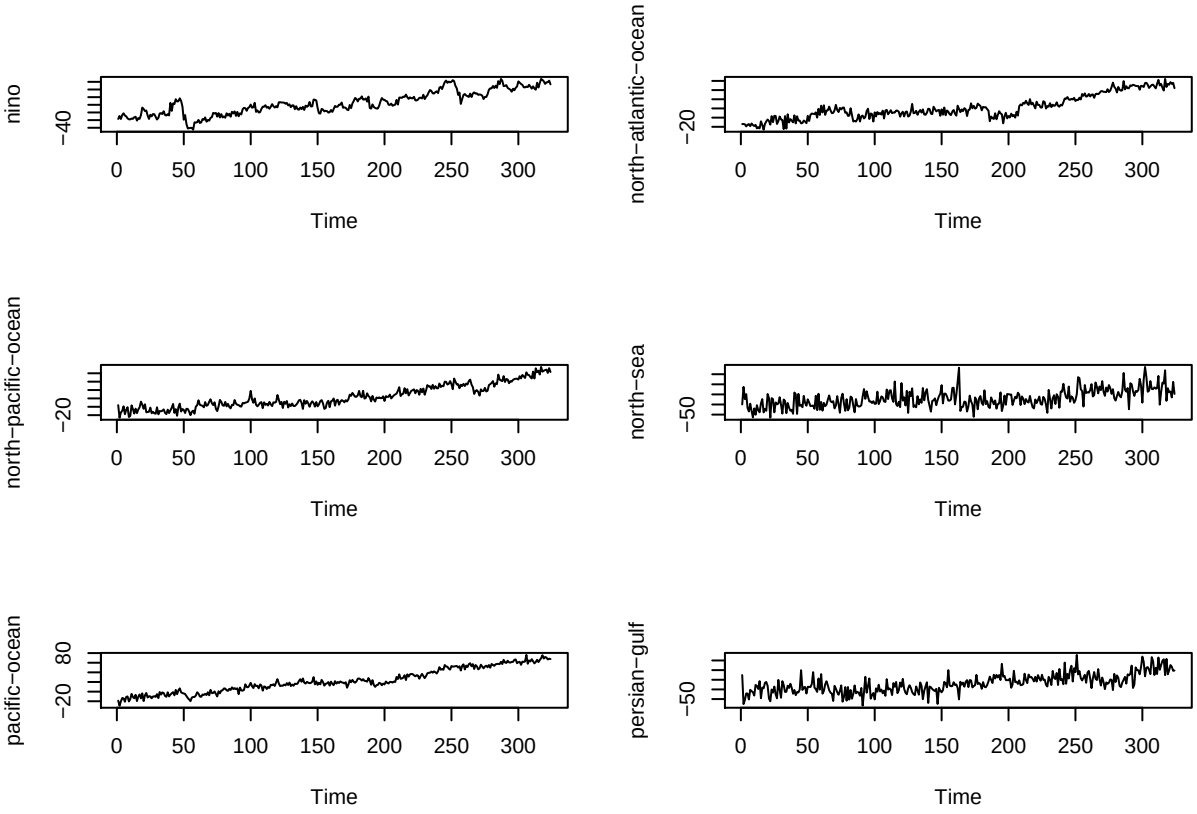

Figure 7: Time series plot of regional sea level values.

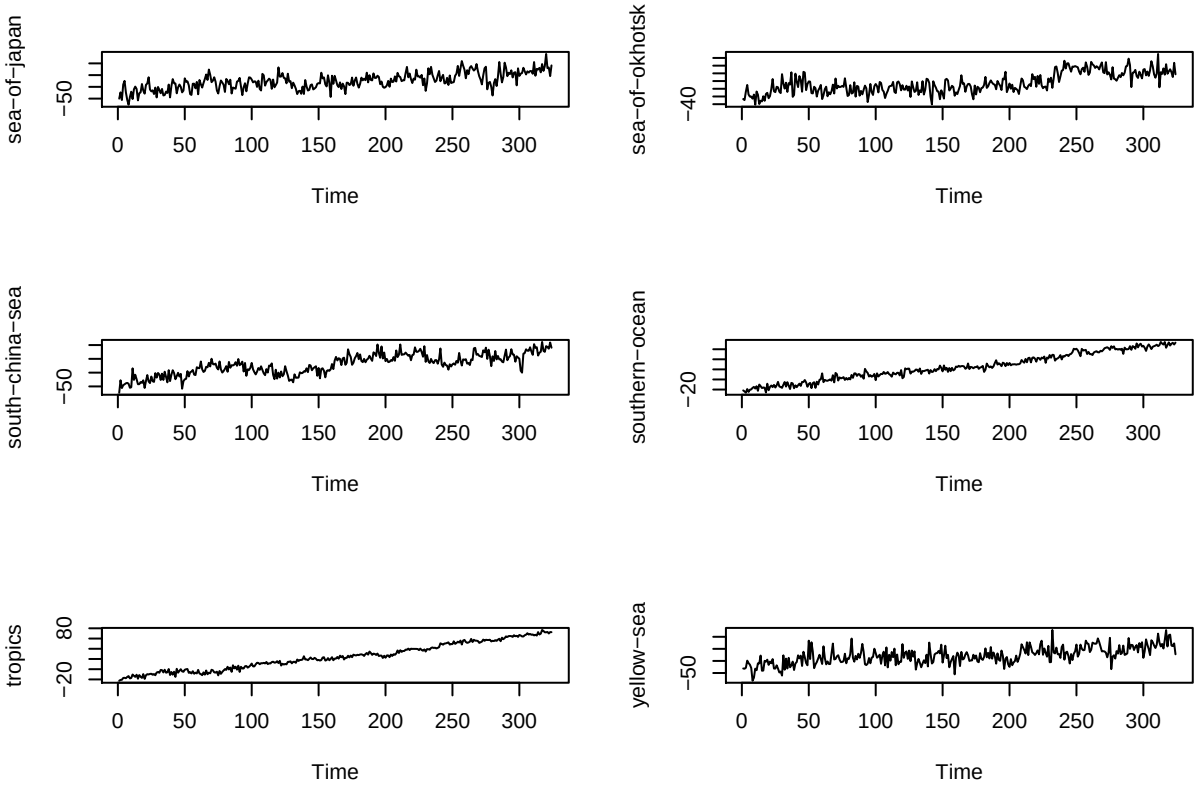

Figure 8: Time series plot of regional sea level values.

Techniques

In this study, we compared the forecasting prediction accuracy of the future global mean and regional sea level values through the application of univariate and multivariate long short-term memory neural networks. Long short-term memory neural networks are an artificial neural network with multiple layers that depend on prior elements in the sequence between the input and output layers and can account for learning long-term temporal dependencies. To perform this study, we used the programming languages, R and Python in an integrated development environment known as RStudio.

Machine Learning Method

In this study, we employed univariate and multivariate long short-term memory neural networks to forecast future global mean and regional sea level values. Before the application of the methods, we had to normalize the data between the range of (-1,1) using two functions MinMaxScaler and StandardScaler. We then created our models based on three training sets (70%, 80%, and 90%). The models were used to predict the global mean and regional sea levels, and the predicted values were then renormalized.

Forecast Evaluations

In this study, we compared the prediction accuracy of the future global mean and regional sea level values with the actual values for three different training sets (70%, 80%, and 90% for each of level). The three predictive measures used to measure the prediction accuracy are mean absolute error,

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

root mean square error, and

$$\sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$

mean absolute percentage error

$$\frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

Results

In our model developement phase, the univariate and multivariate long short-term memory neural networks had different features. The univariate long short-term memory neural networks had one hidden layer, 256 epochs, and 64 batches. The multivariate long short-term memory neural networks had four hidden layers, 128 epochs, and 64 batches. Epochs refers to the amount of times the learning algorithm goes through the provided training set and batches is the number of training examples in one forward/backward pass.

In the metric tables shown in Table 1 - Table 4, the results are shown to be consistent across the global mean and regional sea levels. The low values show that the sea level prediction accuracy is somewhat consistent. When looking at the training percentages, it can be noticed that the 90 percent training level produced better results compared to the 70 and 80 percent training levels due to its lower values. These results can also be noticed when looking at the sea level predicted vs actual plots shown in Figure 9 - Figure 12 as the 90 percent training predicted vs actual plots do not overlap for the most part possibly due to not having enough testing data.

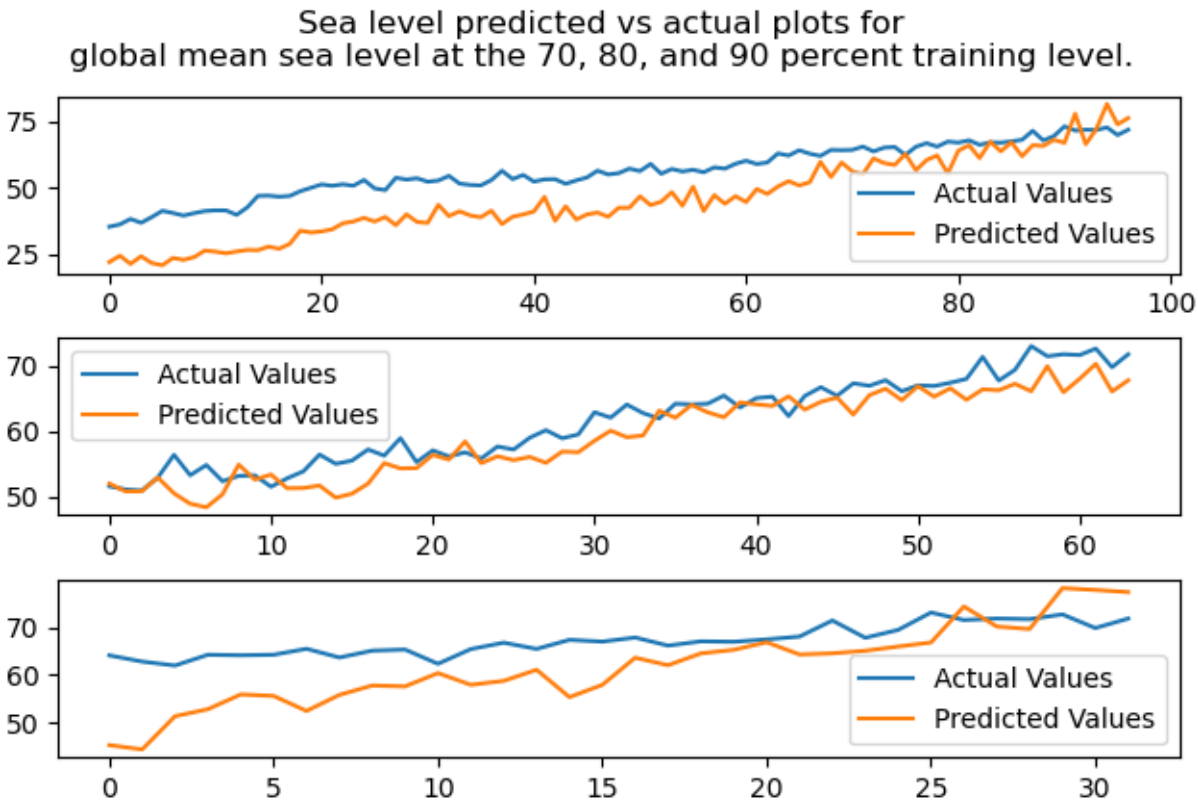

Figure 9: Global mean sea level predicted vs actuals.

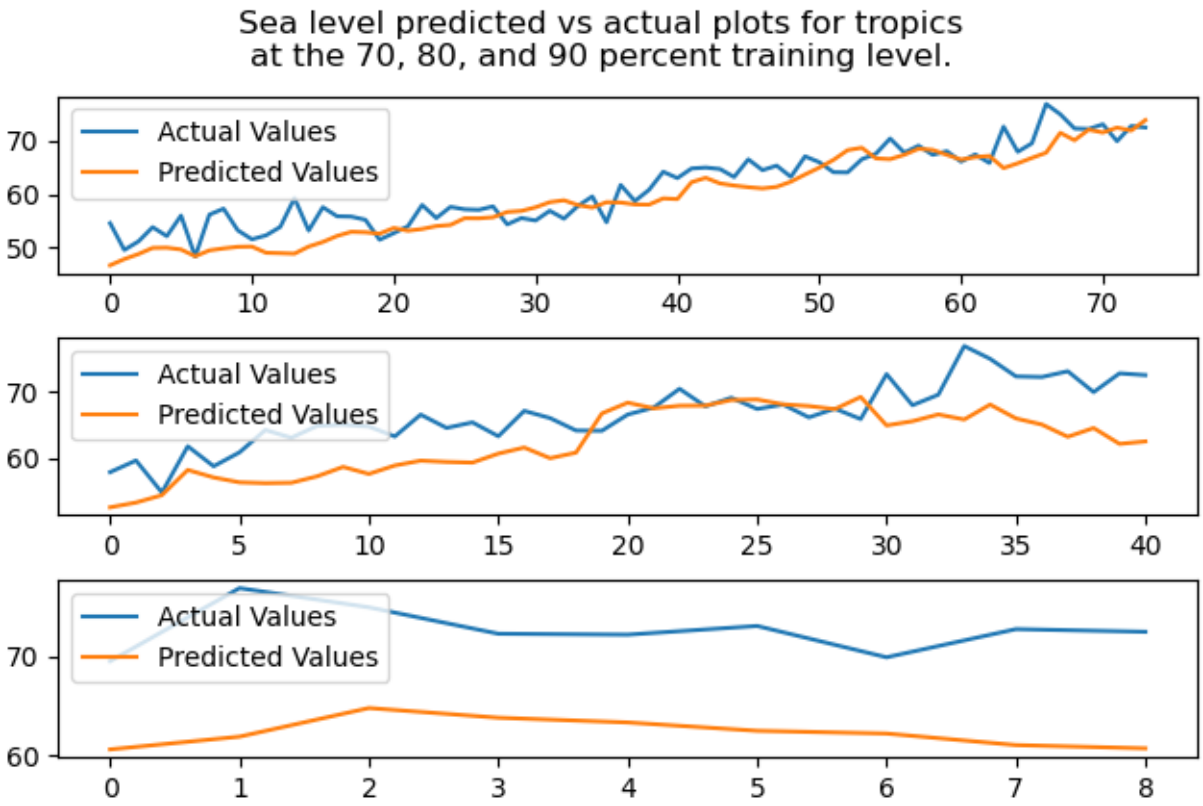

Figure 10: Tropics predicted vs actuals.

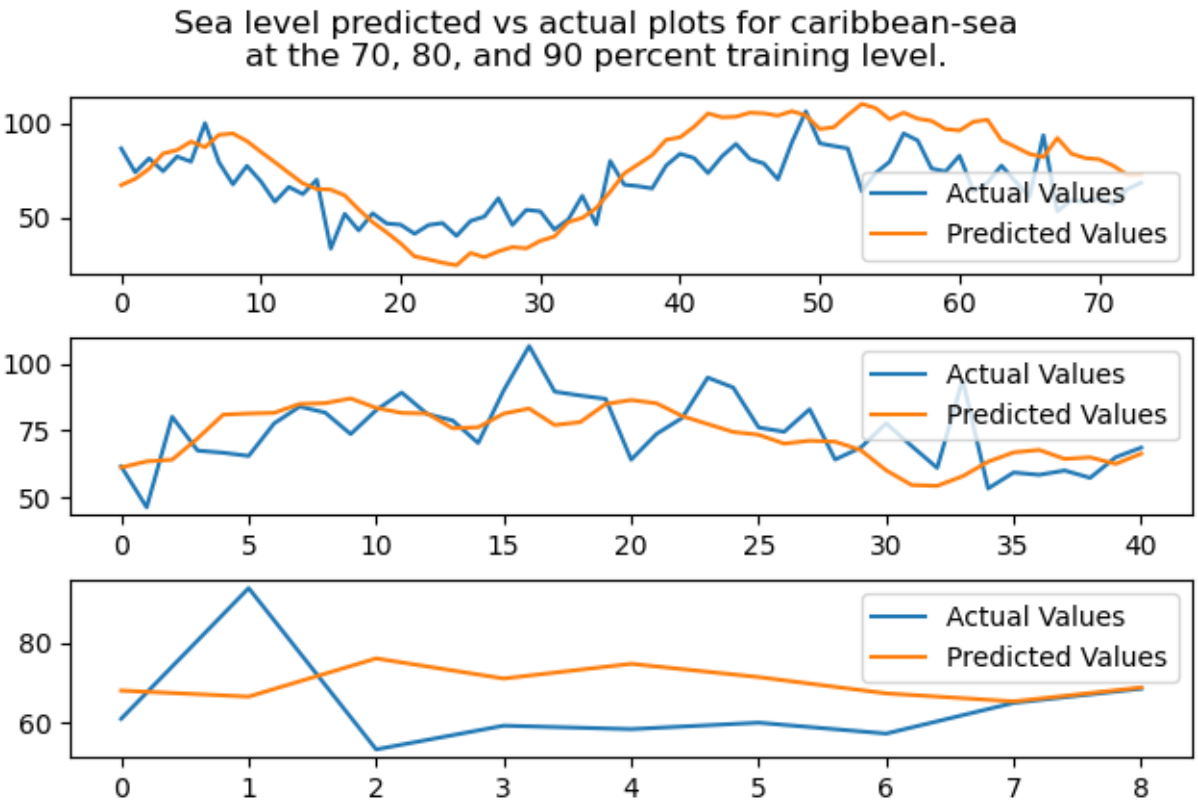

Figure 11: Caribbean sea predicted vs actuals.

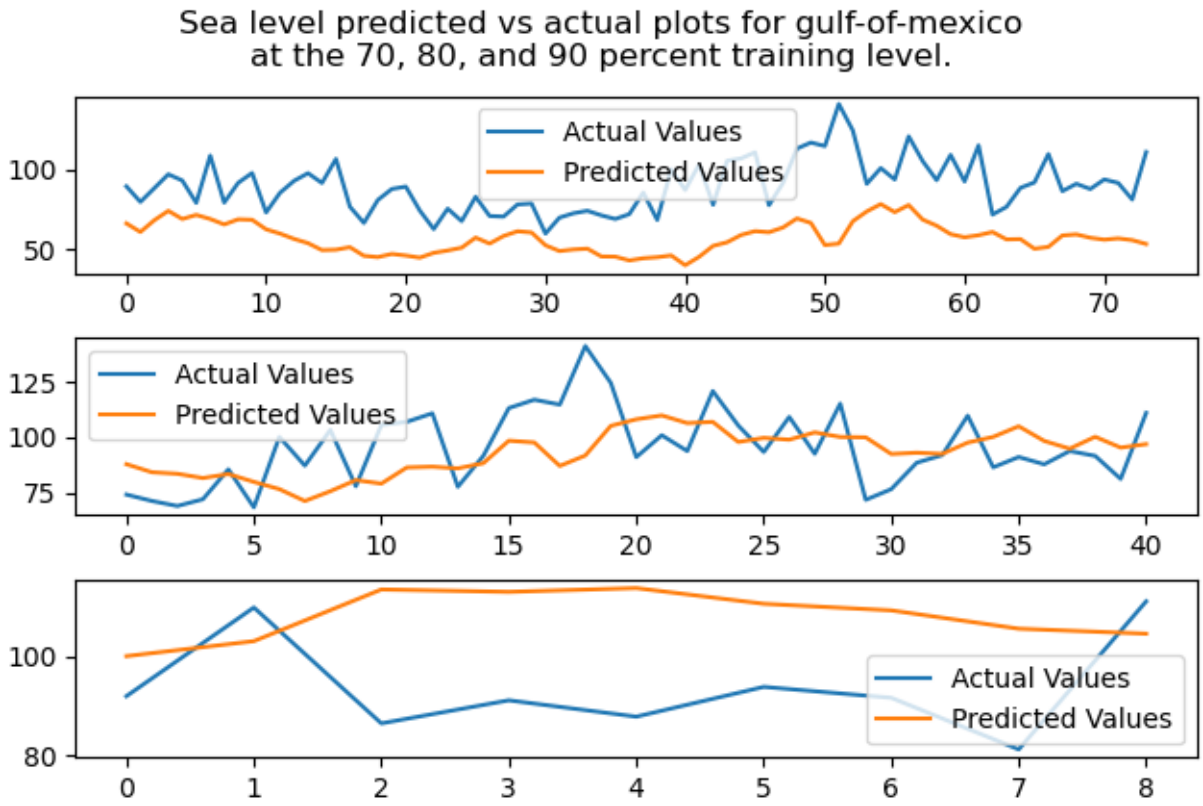

Figure 12: Gulf of mexico predicted vs actuals.

Table 1: Metric table for global mean sea levels predicted vs actuals.

	MAD	RMSE	MAPE
70.0	11.121938	151.375098	0.2167692
80.0	2.487496	9.441927	0.0403490
90.0	6.771972	65.474264	0.1025236

Table 2: Metric table for regional sea levels predicted vs actuals at 70% training level.

	MAD	RMSE	MAPE
adriantic-sea	40.073281	2511.44426	5.615277e+00
andaman-sea	48.736306	3530.71066	9.035265e-01
arabian-sea	19.884857	585.41001	3.549469e-01
atlantic-ocean	9.111360	135.14026	1.638000e-01
baltic-sea	79.268074	11891.96437	1.487122e+00
bay-of-bengal	19.492862	571.11624	1.360764e+00
bering-sea	19.449291	597.56676	9.534683e-01
caribbean-sea	16.447182	361.67617	2.590678e-01
gulf-of-mexico	32.579542	1300.55834	3.502450e-01
indian-ocean	8.259001	105.60944	1.380480e-01
indonesian	36.321965	2271.68850	4.299654e+00

	MAD	RMSE	MAPE
mediterranean-sea	28.034124	1331.42415	1.416589e+00
nino	15.112001	298.88813	2.388142e-01
north-atlantic-ocean	17.271240	438.71944	2.623492e-01
north-pacific-ocean	16.464278	409.82169	2.432080e-01
north-sea	51.461427	3964.30924	1.111876e+00
pacific-ocean	8.276573	103.20342	1.392031e-01
persian-gulf	34.242523	1865.61628	1.551975e+00
sea-of-japan	38.807158	2055.40637	6.246796e+00
sea-of-okhotsk	39.622856	1884.41826	9.052770e-01
south-china-sea	20.984670	657.81609	2.590184e+15
southern-ocean	7.502798	79.19726	1.215910e-01
tropics	2.815814	12.55543	4.657330e-02
yellow-sea	53.435945	3689.37287	1.440232e+00

Table 3: Metric table for regional sea levels predicted vs actuals at 80% training level.

	MAD	RMSE	MAPE
adriantic-sea	40.300223	2374.56565	3.370651e+00
andaman-sea	79.176731	9334.05499	2.100713e+00
arabian-sea	16.612649	399.28468	2.593143e-01
atlantic-ocean	22.356855	567.02153	3.167105e-01
baltic-sea	119.826133	21308.53666	2.740297e+00
bay-of-bengal	35.129294	1578.16529	2.215776e+00
bering-sea	43.900227	2187.27733	2.260190e+00
caribbean-sea	9.199680	142.06238	1.247180e-01
gulf-of-mexico	14.368924	289.63248	1.487033e-01
indian-ocean	7.593354	78.68131	1.190205e-01
indonesian	31.275416	1603.72961	4.012370e+00
mediterranean-sea	27.509839	1125.33516	6.748252e-01
nino	11.054833	197.28873	1.518128e-01
north-atlantic-ocean	13.220548	222.98527	1.873731e-01
north-pacific-ocean	6.520895	62.87693	9.058880e-02
north-sea	73.456913	6897.50631	1.394510e+00
pacific-ocean	6.656638	63.15022	1.059536e-01
persian-gulf	56.716107	4594.89375	7.733671e-01
sea-of-japan	26.399039	1119.40118	6.805900e-01
sea-of-okhotsk	27.336428	1057.22820	1.386967e+00
south-china-sea	21.580567	697.31905	6.571085e+15
southern-ocean	3.785813	23.95386	5.866040e-02
tropics	4.694946	31.46255	6.965540e-02
yellow-sea	26.482461	1087.13418	6.900256e-01

Table 4: Metric table for regional sea levels predicted vs actuals at 90% training level.

	MAD	RMSE	MAPE
adriantic-sea	37.432850	42.219194	3.3396043
andaman-sea	40.189407	48.415469	0.3900926

	MAD	RMSE	MAPE
arabian-sea	20.391837	23.701753	0.2135205
atlantic-ocean	5.179989	6.390039	0.0722198
baltic-sea	89.490023	98.968347	3.6997173
bay-of-bengal	47.438821	49.369052	0.3850097
bering-sea	30.395210	33.242494	3.0921895
caribbean-sea	11.976425	14.764879	0.1884770
gulf-of-mexico	17.254922	18.954614	0.1923173
indian-ocean	8.390132	10.005727	0.1250234
indonesian	29.423135	32.832292	0.2343133
mediterranean-sea	25.404030	29.796411	1.3423109
nino	10.283204	11.645461	0.1255058
north-atlantic-ocean	6.919486	8.150574	0.1006345
north-pacific-ocean	33.048860	33.707193	0.3813132
north-sea	39.493570	47.750053	0.9729763
pacific-ocean	1.527629	1.786092	0.0222871
persian-gulf	73.430565	80.211815	0.6414198
sea-of-japan	20.812226	26.966880	0.2436165
sea-of-okhotsk	19.337522	21.415986	0.4919654
south-china-sea	11.711235	16.010531	0.1157970
southern-ocean	5.567382	6.230943	0.0809535
tropics	10.299207	10.512434	0.1412183
yellow-sea	20.849645	24.046353	0.3689619

Conclusion

Previously, we mentioned that the global mean and regional sea levels predicted vs actuals at the 70 and 80 percent training levels are somewhat accurate. There are times where the predicted vs actuals do not overlap but for the most part they do which suggests that are models at the 70 and 80 percent training levels are somewhat accurate. This seems to be different for the 90 percent training level level because earlier we mentioned that when looking at Table 1 - Table 4, the results look better but when looking at Figure 9 - Figure 12, the results are shown to differ as they do not overlap for the most part. This is possibly due to not having enough testing data so we truly do not know how good the 90 percent training level models are. Our results from this study suggest that univariate and multivariate long short-term memory neural networks are useful in predicting future sea level values at the global mean and regional setting and that these methods would allow us to understand how rising sea levels will progress in the future. Futher research can be done on exploring the use of univariate long short-term memory neural networks on regional sea levels which may result in the predicted values being better or slightly worse depending on whether the loss of the shared relationships between regional seas is beneficial or detrimental.

Appendix

```
library(keras)
tensorflow::set_random_seed(1234)
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(comment = NA)
```

```
import numpy as np
my_seed = 20 #5
```

```
np.random.seed(my_seed)
import random
random.seed(my_seed)
import tensorflow as tf
tf.random.set_seed(my_seed)
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense,LSTM
from sklearn.metrics import mean_squared_error,mean_absolute_error,mean_absolute_percentage_error
from keras.layers import Dropout
from sklearn.preprocessing import StandardScaler
# creating a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back ):
        a = dataset[i:(i + look_back):]
        dataX.append(a)
        dataY.append(dataset[i + look_back, :])
    return np.array(dataX), np.array(dataY)

# returns the dataset with new column names
def returndataset(arraywithstuff,columns):
    dataset = pd.DataFrame(columns = columns)
    i = 0
    for name in columns:
        dataset[name] = arraywithstuff[:,i]
        i = i + 1
    return dataset
def multivariate_lstm(dataset):
    # create dictionaries
    actual_dictionary = {}
    multivariate_lstm_dictionary = {}
    #create variable equal to the values in the dataset provided
    values = dataset.values
    # created a scaler between the range of -1 and 1
    scaler = MinMaxScaler(feature_range=(-1, 1))
    # used the scaler to transform the dataset values
    scaled = scaler.fit_transform(values)
    # for creating training and testing set
    number = [0.7,0.8,0.9]
    for n in number:
        # create a variable equal to the number of columns in dataset
        look_back = 24
        # creating testing and training sets
        train_set = scaled[:math.floor(n*len(scaled)),:]
        test_set = scaled[math.floor(n*len(scaled)):,:]
        # splitting training and testing sets into x and y versions
        trainX, trainY = create_dataset(train_set, look_back)
        testX, testY = create_dataset(test_set, look_back)
        # creating model
```

```
        model = Sequential()
        model.add(LSTM(256, input_shape=(24,24)))
        model.add(Dense(24))
        model.compile(loss='mae', optimizer='adam')
        # fitting the model
        history = model.fit(trainX, trainY, epochs=128, batch_size=64,
        validation_data=(testX, testY))
        # making predictions
        trainPredict = model.predict(trainX)
        testPredict = model.predict(testX)
        # inversing the transformation
        #trainPredict_extended = scaler.inverse_transform(trainPredict)
        testPredict_extended = scaler.inverse_transform(testPredict)
        #trainYActual_extended = scaler.inverse_transform(trainY)
        testYActual_extended = scaler.inverse_transform(testY)
        #trainPredict_extended = returndataset(trainPredict_extended)
        # returning the predictions as a dataset
        testPredict_extended = returndataset(testPredict_extended,dataset.columns)
        #trainYActual_extended = returndataset(trainYActual_extended)
        testYActual_extended = returndataset(testYActual_extended,dataset.columns)
        # putting the predictions and actuals in the dictionaries
        multivariate_lstm_dictionary[n*100] = testPredict_extended
        actual_dictionary[n*100] = testYActual_extended
    return multivariate_lstm_dictionary,actual_dictionary
def lstm(X_train, y_train, X_test,sc):
    model = Sequential()
    #Adding the first LSTM layer and some Dropout regularisation
    model.add(LSTM(units = 256, return_sequences = True,
    input_shape = (X_train.shape[1], 1)))
    model.add(Dropout(0.2))
    # Adding a second LSTM layer and some Dropout regularisation
    model.add(LSTM(units = 128, return_sequences = True))
    model.add(Dropout(0.2))
    # Adding a third LSTM layer and some Dropout regularisation
    model.add(LSTM(units = 32, return_sequences = True))
    model.add(Dropout(0.2))
    # Adding a fourth LSTM layer and some Dropout regularisation
    model.add(LSTM(units = 32))
    model.add(Dropout(0.2))
    # Adding the output layer
    model.add(Dense(units = 1))
    # Compiling the RNN
    model.compile(optimizer = 'adam', loss = 'mean_squared_error')
    # Fitting the RNN to the Training set
    model.fit(X_train, y_train, epochs = 256, batch_size = 64)
    # making predictions
    LSTM_prediction = model.predict(X_test)
    return model, LSTM_prediction
def lstm_model(dataset):
    # creating dictionaries
    actual_dictionary = {}
    lstm_dictionary = {}
    lstm_dictionary[0.7*100] = {}
```

```
    lstm_dictionary[0.8*100] = {}
    lstm_dictionary[0.9*100] = {}
    actual_dictionary[0.7*100] = {}
    actual_dictionary[0.8*100] = {}
    actual_dictionary[0.9*100] = {}
    # going through the columns and numbers used for creating training and testing sets
    for column in dataset.columns:
        number = [0.7,0.8,0.9]
        scr = []
        for n in number:
            # Split into train and test set
            train_size = math.floor(n*len(dataset[column]))
            train_set = dataset[column].head(train_size).values.reshape(-1,1)
            test_set = dataset[column].tail(len(dataset[column]) -
            train_size).values.reshape(-1,1)
            test_size = len(test_set)
            # Scaler
            sc = StandardScaler()
            ts_train_scaled = sc.fit_transform(train_set)
            # Splitting the training set further
            X_train = []
            y_train = []
            y_train_stacked = []
            for i in range(5,train_size-1):
                X_train.append(train_set[i-5:i,0])
                y_train.append(train_set[i:i+1,0])
            X_train, y_train = np.array(X_train), np.array(y_train)
            # changing the shape of X_train
            X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
            # creating a dataset used for predictions
            inputs = pd.concat((dataset[column].head(train_size),
            dataset[column].tail(len(dataset[column]) - train_size)),axis=0).values
            inputs = inputs[len(inputs)-len(test_set) - 5:]
            inputs = inputs.reshape(-1,1)
            inputs  = sc.transform(inputs)
            # turning the dataset into a matrix
            X_test = []
            for i in range(5,test_size+5-1):
                X_test.append(inputs[i-5:i,0])
            # creating a matrix of the X_test and changing the shape
            X_test = np.array(X_test)
            X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
            # Creates model
            model, model_predictions = lstm(X_train, y_train, X_test,sc)
            model.summary()
            # creating dataset based on actual and predicted values
            actual_pred = pd.DataFrame(columns = ['actual', 'prediction'])
            actual_pred['actual'] = dataset[column].tail(len(dataset[column]) -
            train_size)[0:len(model_predictions)]
            actual_pred['prediction'] = model_predictions[:,0]
            # putting the inverse of the predictions and actual values in the dictionaries
            lstm_dictionary[n*100][column] = sc.inverse_transform(actual_pred['prediction'])
            actual_dictionary[n*100][column] = actual_pred['actual']
```

```
    return lstm_dictionary,actual_dictionary
```

```
regional_dictionary = py$multivariate_lstm(dataframe_regional)
global_dictionary  = py$lstm_model(dataframe_global)
actual_regional_dictionary = regional_dictionary[2]
regional_dictionary = regional_dictionary[1]
actual_global_dictionary = global_dictionary[2]
global_dictionary = global_dictionary[1]
```

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,
mean_absolute_percentage_error
global_table = pd.DataFrame(columns=["MAD","RMSE","MAPE"])
regional_seventy_table = pd.DataFrame(columns=["MAD","RMSE","MAPE"])
regional_eighty_table = pd.DataFrame(columns=["MAD","RMSE","MAPE"])
regional_ninty_table = pd.DataFrame(columns=["MAD","RMSE","MAPE"])
for i in ["70.0","80.0","90.0"]:
    # creating tables based on predicted vs actuals for global mean sea level
    global_table = global_table.append(pd.DataFrame({"MAD":mean_absolute_error((r.actual_global_dictiona
    # creating plots based on predicted vs actuals for global mean at differet percents
    plt.plot(((r.actual_global_dictionary[0][i])["global-mean-sea-level"].reset_index())["actual"],label
    plt.plot((r.global_dictionary[0][i])["global-mean-sea-level"], label= "Predicted Values")
    plt.legend(loc='best')
    plt.xlabel('Time')
    plt.ylabel('Sea Level Values')
    plt.title(i + " Training Level Predicted vs Actual Values for Global Sea Level Mean")
    plt.show()
    plt.clf()
# creating plots based on predicted vs actuals at different percents for tropics caribbean sea and gulf
for name in ["tropics","caribbean-sea","gulf-of-mexico"]:
  for i in ["70.0","80.0","90.0"]:
    plt.plot(r.actual_regional_dictionary[0][i][name],label = "Actual Values")
    plt.plot(r.regional_dictionary[0][i][name], label= "Predicted Values")
    plt.xlabel('Time')
    plt.ylabel('Sea Level Values')
    plt.title(i + " Training Level Sea Level Predicted vs Actual Values for " + name[0:len(name)].repla
    plt.show()
    plt.clf()
# creating tables based on the predicted vs actuals at different percents regional sea levels
for i in ["70.0"]:
  for name in r.dataframe_regional.columns:
    regional_seventy_table = regional_seventy_table.append(pd.DataFrame({
      "MAD":mean_absolute_error(r.actual_regional_dictionary[0][i][name],
      r.regional_dictionary[0][i][name]),"RMSE":mean_squared_error(
        r.actual_regional_dictionary[0][i][name],
      r.regional_dictionary[0][i][name]),"MAPE":mean_absolute_percentage_error(
        r.actual_regional_dictionary[0][i][name],
        r.regional_dictionary[0][i][name])}, index=[name]))
for i in ["80.0"]:
  for name in r.dataframe_regional.columns:
    regional_eighty_table = regional_eighty_table.append(pd.DataFrame({
      "MAD":mean_absolute_error(r.actual_regional_dictionary[0][i][name],
      r.regional_dictionary[0][i][name]),"RMSE":mean_squared_error(
        r.actual_regional_dictionary[0][i][name],r.regional_dictionary[0][i][name]),
        "MAPE":mean_absolute_percentage_error(r.actual_regional_dictionary[0][i][name],
```

```
        r.regional_dictionary[0][i][name])}, index=[name]))
for i in ["90.0"]:
  for name in r.dataframe_regional.columns:
    regional_ninty_table = regional_ninty_table.append(pd.DataFrame({
      "MAD":mean_absolute_error(r.actual_regional_dictionary[0][i][name],
      r.regional_dictionary[0][i][name]),"RMSE":mean_squared_error(
        r.actual_regional_dictionary[0][i][name],r.regional_dictionary[0][i][name],
        squared=False),"MAPE":mean_absolute_percentage_error(
          r.actual_regional_dictionary[0][i][name],r.regional_dictionary[0][i][name])}
          , index=[name]))
```

```
global_table
regional_seventy_table
regional_eighty_table
regional_ninty_table
```