In [1]:

```
# Predict the coordinates of Burmese python (Python molurus bivittatus)
# in the Greater Everglades Ecosystem, Florida
# using Deep Learning neural network architectures: LSTM, GRU, Conv1D

## Main objective of the analysis that also specifies whether your model will be focused
on a specific type of Deep Learning or Reinforcement Learning algorithm and the benefits
that your analysis brings to the business or stakeholders of this data.

### The objective of this analysis is to predict the coordinates of Burmese Pythons to as
sist hunters in locating the invasive species.
### To accomplish,  LSTM, GRU, Conv1D deep learning neural network was utilized.


## Does the report include a paragraph detailing the main objective(s) of this analysis?


### Yes, see question and answer above.


## Brief description of the data set you chose, a summary of its attributes, and an outli
ne of what you are trying to accomplish with this analysis.

### Using the dataset found at https://www.sciencebase.gov/catalog/item/63221898d34e71c6d
67ab5ab
### The file, FL Specimen Management Database Pre-2022PYMO.csv, was used and is called 63
221898d34e71c6d67ab5ab.csv.
### The data consists of the following values:
###       OID - unique ID
###       Year - Year of data capture
###       Date - Date of data capture
###       ID - a different ID
###       UTMEast - east coordinate
###       UTMNorth - north coordinate
###       Sex - Male/Female
###       Weight - of python
###       SVL - Snout Vent Length of python
###       TailLength - of python
###       TotalLength - of python
###       TailStatus - of python
###       ReproStatus - of python (this is the target variable where were are attempting
to classify)
###       #FolliclesOrEggs - is the count of Follicles (in ovaries) or Eggs for the pytho
n


## Brief summary of data exploration and actions taken for data cleaning or feature engin
eering.

### Form the data above, the only data used was the UTMEast and UTMNorth fields.  Any mis
sing data was deemed too critical to assume.
### As a result, if any values were missing the data row was discarded.    The final valu
es were scaled between 0 and 1 to ensure a
### better outcome in machine learning.


## Does the report include a section describing the data?

### Yes, see the two sections above.


## Summary of training at least three variations of the Deep Learning model you selected.
For example, you can use different clustering techniques or different hyperparameters.

### Three models were used training: LSTM, GRU, Conv1D.  This is explained in greater det
ail in the "Summary Key Findings..." section below.
```

```
## A paragraph explaining which of your Deep Learning models you recommend as a final model that best fits your needs in terms of accuracy or explainability.

### The GRU model produced slightly better results when measuring the MSE.   Rounding down, GRU had a value of 0.015 while the others exceeded 0.016.
### The actual predicted cooridnate is shown at the bottom of this notebook, see "Predicted next coordinate"


## Summary Key Findings and Insights, which walks your reader through the main findings of your modeling exercise.

### This code is a practical application of deep learning techniques to process and predict data sequences, using Python and libraries like Keras for building neural network models.
###
### Creating Sequences from Data:
###
### The function create_sequences is designed to take a dataset (data) and a sequence length (seq_length). It processes the data into sequences of a specified length.
### These sequences (xs) are used as input to predict the next value in the sequence (ys). This is a common preparation step for time series forecasting or any
### scenario where you want to predict a future value based on a sequence of previous values.
### Preparing the Data:
###
### seq_length = 5 sets the length of each input sequence to 5. This means each input to the model will be a sequence of 5 data points.
### X, y = create_sequences(data_scaled, seq_length) calls the function to create input-output pairs from the scaled dataset, preparing it for training.
### Splitting the Data:
###
### The dataset is divided into training and test sets using train_test_split, with 20% of the data reserved for testing. This allows us to train the
### models on one portion of the data and evaluate their performance on a separate, unseen portion.
### Defining Deep Learning Models:
###
### Three models are defined using the Sequential API from Keras: an LSTM model, a GRU model, and a Conv1D model. Each model is structured to take
### sequences of length 5 with 2 features per time step as input.
### LSTM and GRU Models: Both models use a type of recurrent neural network layer (LSTM or GRU) suitable for sequence data, followed by a dense layer for output.
### They're particularly good at capturing temporal dependencies.
### Conv1D Model: This model uses a convolutional approach, ideal for finding patterns in sequence data, followed by max pooling, flattening, and dense layers.
### Training and Evaluating Models:
###
### Each model is compiled with the Adam optimizer and mean squared error (MSE) as the loss function, which are common choices for regression problems.
### The models are then trained on the training set, including a validation split to monitor performance during training.
### After training, the models are evaluated on the test set to calculate their MSE, providing a measure of how well each model predicts the unseen data.
### Selecting the Best Model and Making Predictions:
###
### Based on the MSE, the GRU model is selected as the best performing model. This decision is made because it has the lowest MSE, indicating better performance on the test set.
### A prediction is made for the next value following the last sequence in the test set using the selected model. This showcases the model's ability to predict
## future data points based on learned patterns.
### Scaling Back the Prediction:
###
### The predicted value is scaled back to its original scale using scaler.inverse_transform. This step is necessary because the data was
### scaled down before training to help the models converge more quickly.
### Final Output:
###
### The code prints the predicted next coordinate after scaling back, demonstrating the model's prediction capability when attempting to predict the next location of a Burmeise Python.
### This code is a comprehensive example of how to preprocess data, implement, train, and
```

```
evaluate multiple deep learning models for sequence prediction,
### and use the best model to make predictions.


## Suggestions for next steps in analyzing this data, which may include suggesting revisi
ting this model or adding specific data features to achieve a better model.

### For next steps, a lot could be done.   Ideally missing data would be added.   If it c
annot be added, then different techniques of assuming this data should be evaluated.
### I am also concerned that my data is biased.   The data was limited to Southern Florid
a and then scaled between 0 and 1.  However any coordinate in the world is possible.
### The models predicted Burmese Pythons would be in Soutern Florida, but the predictions
would have been wildly inaccurate if we also included coordinates on the other
### side of the world.
```

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Image
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, GRU, Dense, Conv1D, MaxPooling1D, Flatten, Inp
ut
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

In [3]:

```
Image(filename='Screenshot 2024-06-23 at 12.10.55 PM.png')
# Show below is a screen shot of the site wth the data source
# https://www.sciencebase.gov/catalog/item/63221898d34e71c6d67ab5ab
```

Out[3]:

ScienceBase Catalog → USGS Data Release Products → Size distribution and reprod...

View

Size distribution and reproductive data of the invasive Burmese python (Python molurus bivittatus) in the Greater Everglades Ecosystem, Florida, USA, 1995-2021

Dates

Publication Date : 2022-11-23
Start Date : 1995-12-12
End Date : 2021-12-31

Citation

Currylow, A.F., Falk, B.G., Yackel Adams, A.A., Romagosa, C.M., Josimovich, J.M., Rochford, M.R., Cherkiss, M.S., Nafus, M.G., Hart, K.M., Mazzotti, F.J., Snow, R.W., and Reed, R.N., 2022, Size distribution and reproductive data of the invasive Burmese python (Python molurus bivittatus) in the Greater Everglades Ecosystem, Florida, USA, 1995-2021: U.S. Geological Survey data release, https://doi.org/10.5066/P9CZI2KO.

Summary

This dataset contains morphometric information from Burmese pythons collected from an invasive population in southern Florida between 1995-2021. Scientists from the U.S. Geological Survey and the National Park Service curated this dataset as a repository for records of Burmese pythons found on or nearby federal lands in southern Florida, including Everglades National Park, Big Cypress National Preserve, Biscayne National Park, and Crocodile Lake National Wildlife Refuge. As such, numerous entities actively or incidentally involved in python research or management activities contributed specimens and/or data to this dataset, including but not limited to the U.S. Geological Survey, National Park Service, U.S. Fish and Wildlife Service, University of Florida, Conservancy of Southwest Florida, Florida Fish and Wildlife Conservation Commission, South Florida Water Management District, volunteers, and members of the public. The dataset includes python identification information, capture information, morphometric data, and necropsy data. The structure of the dataset is such that every row pertains to a single date that data were collected from a single python so that serial captures and morphological data collected from unique individuals can be tracked across time via different rows.

Map »

Communities

- Fort Collins Science Center (FORT)
- USGS Data Release Products

Tags

Categories : Data
Harvest Set : USGS Science Data Catalog (SDC)
Theme : biota, herpetology, invasive species, life sciences, morphology (biological), organism growth and development, phenology, reptiles

In [4]:

```
# Load the CSV file into a DataFrame
# Load your dataset
data = pd.read_csv('63221898d34e71c6d67ab5ab.csv')
```

In [5]:

```
data
```

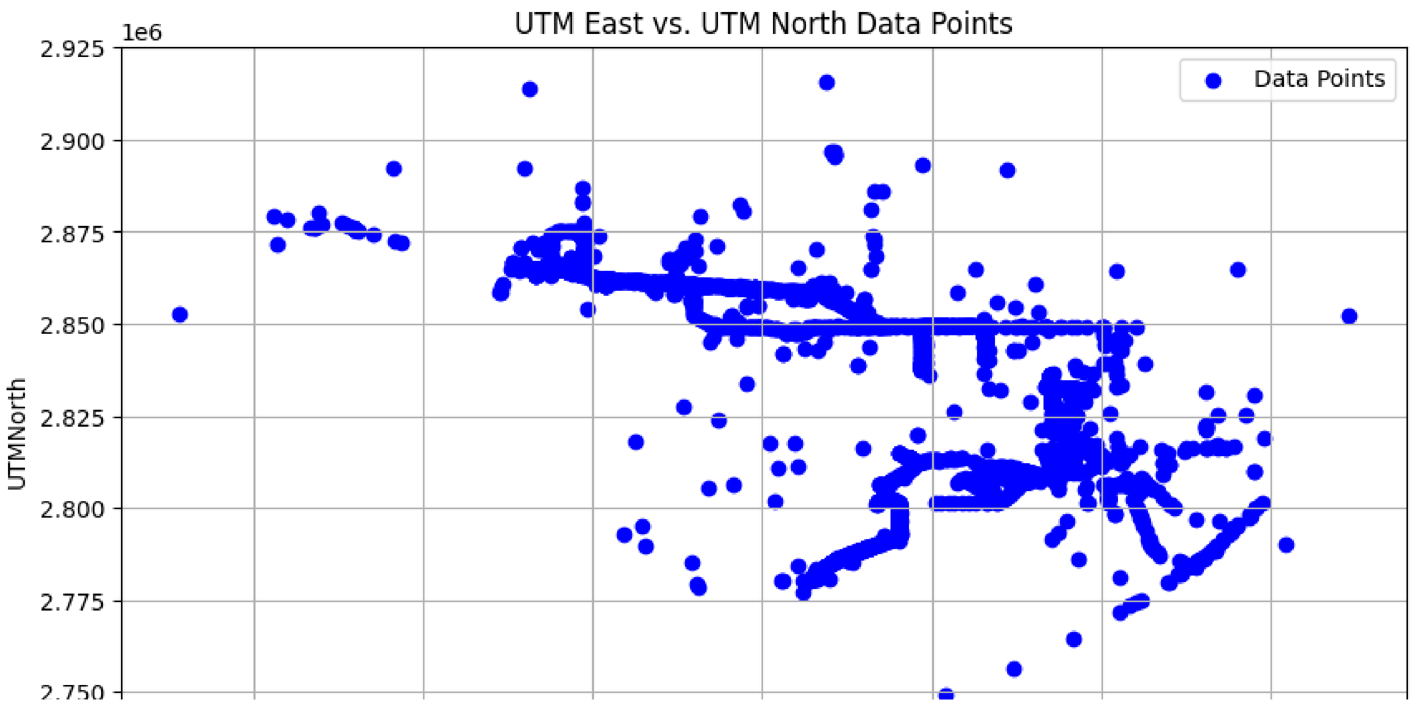
Out[5]:

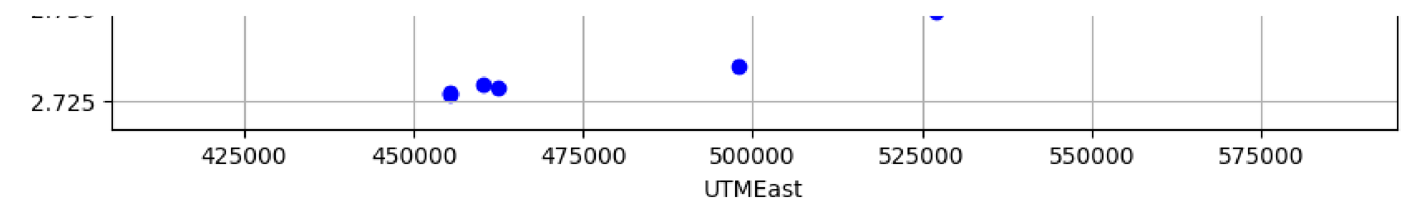
	OID	Year	Date	ID	UTMEast	UTMNorth	Sex	Weight	SVL	TailLength	TotalLength	TailStatus	Re
0	2	2004.0	2004-10-12	12Oct04.01	530923.0	2806709.0	Male	NaN	NaN	NaN	NaN	NaN	
1	4	2006.0	2006-09-13	13Sep06.01	519027.0	2807146.0	Female	NaN	NaN	NaN	NaN	NaN	
2	8	2007.0	2007-03-24	24Mar07.01	517307.0	2849303.0	Female	NaN	NaN	NaN	NaN	NaN	
3	12	2008.0	2008-11-08	08Nov08.01	521952.0	2813389.0	Female	NaN	NaN	NaN	NaN	NaN	
4	13	2008.0	2008-11-11	11Nov08.01	541942.0	2808671.0	Female	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	
7466	24272	2021.0	2021-02-24	24Sep19.07	504451.0	2856709.0	Male	4400.0	NaN	NaN	NaN	NaN	
7467	24274	2021.0	2021-03-24	24Sep19.07	505316.0	2857048.0	Male	NaN	NaN	NaN	NaN	NaN	
7468	24275	2020.0	2020-06-01	2020-05-30.01	NaN	NaN	Female	2754.0	175.5	25.0	200.5	Intact	
7469	24276	2021.0	2021-06-24	24Sep19.07	NaN	NaN	Male	4500.0	NaN	NaN	NaN	NaN	
7470	24277	2020.0	2020-06-02	2020-06-01.01	NaN	NaN	Male	5936.0	203.5	36.4	239.9	Intact	

7471 rows × 14 columns

In [6]:

```
plt.figure(figsize=(10, 6))  # Set the figure size for better readability
plt.scatter(data['UTMEast'], data['UTMNorth'], marker='o', color='b', label='Coordinates
where Burmeise Python were Found')
plt.title('UTM East vs. UTM North Data Points')
plt.xlabel('UTMEast')
plt.ylabel('UTMNorth')
plt.legend()
plt.grid(True)
plt.show()
```





In [7]:

```
data = data.dropna(subset=['UTMEast'])
data = data.dropna(subset=['UTMNorth'])

# Normalize coordinates
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data[['UTMEast', 'UTMNorth']])

# Convert scaled data back to DataFrame
data_scaled = pd.DataFrame(data_scaled, columns=['UTMEast', 'UTMNorth'])
```

In [8]:

```
data_scaled
```

Out[8]:

	UTMEast	UTMNorth
0	0.678135	0.422493
1	0.609140	0.424806
2	0.599165	0.647971
3	0.626105	0.457854
4	0.742043	0.432879
...	...	...
4096	0.556055	0.305910
4097	0.325258	0.738308
4098	0.627253	0.446129
4099	0.524603	0.687176
4100	0.529620	0.688971

4101 rows × 2 columns

In [19]:

```
# Function to split data into sequences
def create_sequences(data, seq_length):
    xs, ys = [], []
    for i in range(len(data)-seq_length-1):
        x = data.iloc[i:(i+seq_length)].values
        y = data.iloc[i+seq_length].values
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

seq_length = 5
X, y = create_sequences(data_scaled, seq_length)

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
)

# Define LSTM Model
model_lstm = Sequential([
    Input(shape=(seq_length, 2)),
    LSTM(50, activation='relu'),
```

```python
    Dense(2)
])

# Define GRU Model
model_gru = Sequential([
    GRU(50, activation='relu', input_shape=(seq_length, 2)),
    Dense(2)
])

# Define Conv1D Model
model_conv1d = Sequential([
    Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(seq_length, 2)),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(50, activation='relu'),
    Dense(2)
])

# List of models
models = [model_lstm, model_gru, model_conv1d]
model_names = ['LSTM', 'GRU', 'Conv1D']
mse_values = []

# Train and evaluate each model
for model, name in zip(models, model_names):
    print(f"Training and evaluating model: {name}")
    model.compile(optimizer='adam', loss='mse')
    model.fit(X_train, y_train, epochs=20, validation_split=0.2, verbose=1)
    mse = model.evaluate(X_test, y_test, verbose=0)
    mse_values.append(mse)
    print(f'{name} Model MSE: {mse}\n')


# This is just an example, you should choose the model based on the actual evaluation
best_model = models[1]  # GRU Model had the lowest MSE of 0.015634542331099951 while the o
thers were greater than 0.016
next_coordinate = best_model.predict(X_test[-1].reshape(1, seq_length, 2))
next_coordinate_scaled = scaler.inverse_transform(next_coordinate)
print(f'Predicted next coordinate: {next_coordinate_scaled}')
```

```
Training and evaluating model: LSTM
Epoch 1/20
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/keras/src
/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to
a layer. When using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/keras/src
/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_
dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` ob
ject as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1630 - val_loss: 0.0218
Epoch 2/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0197 - val_loss: 0.0173
Epoch 3/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0159 - val_loss: 0.0169
Epoch 4/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0161 - val_loss: 0.0173
Epoch 5/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0166 - val_loss: 0.0176
Epoch 6/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0159 - val_loss: 0.0166
Epoch 7/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0156 - val_loss: 0.0166
Epoch 8/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0159 - val_loss: 0.0166
Epoch 9/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0164 - val_loss: 0.0164
Epoch 10/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0156 - val_loss: 0.0164
```

```
Epoch 11/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0161 - val_loss: 0.0165
Epoch 12/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0169 - val_loss: 0.0164
Epoch 13/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0154 - val_loss: 0.0166
Epoch 14/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0160 - val_loss: 0.0162
Epoch 15/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0159 - val_loss: 0.0165
Epoch 16/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0160 - val_loss: 0.0167
Epoch 17/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0150 - val_loss: 0.0161
Epoch 18/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0156 - val_loss: 0.0167
Epoch 19/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0154 - val_loss: 0.0163
Epoch 20/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0151 - val_loss: 0.0162
LSTM Model MSE: 0.015759721398353577

Training and evaluating model: GRU
Epoch 1/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.1470 - val_loss: 0.0184
Epoch 2/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0168 - val_loss: 0.0163
Epoch 3/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0153 - val_loss: 0.0166
Epoch 4/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0163 - val_loss: 0.0163
Epoch 5/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0159 - val_loss: 0.0161
Epoch 6/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0161 - val_loss: 0.0164
Epoch 7/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0165 - val_loss: 0.0165
Epoch 8/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0162 - val_loss: 0.0163
Epoch 9/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0157 - val_loss: 0.0164
Epoch 10/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0162 - val_loss: 0.0160
Epoch 11/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0155 - val_loss: 0.0161
Epoch 12/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0158 - val_loss: 0.0159
Epoch 13/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0154 - val_loss: 0.0160
Epoch 14/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0154 - val_loss: 0.0160
Epoch 15/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0157 - val_loss: 0.0160
Epoch 16/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0148 - val_loss: 0.0159
Epoch 17/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0161 - val_loss: 0.0164
Epoch 18/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0150 - val_loss: 0.0159
Epoch 19/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0157 - val_loss: 0.0160
Epoch 20/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0153 - val_loss: 0.0162
GRU Model MSE: 0.01568499021232128

Training and evaluating model: Conv1D
Epoch 1/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0993 - val_loss: 0.0178
Epoch 2/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 654us/step - loss: 0.0171 - val_loss: 0.0173
Epoch 3/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 622us/step - loss: 0.0166 - val_loss: 0.0172
```

```
Epoch 4/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 636us/step - loss: 0.0160 - val_loss: 0.0175
Epoch 5/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step - loss: 0.0153 - val_loss: 0.0169
Epoch 6/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 652us/step - loss: 0.0160 - val_loss: 0.0170
Epoch 7/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 615us/step - loss: 0.0160 - val_loss: 0.0168
Epoch 8/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step - loss: 0.0161 - val_loss: 0.0166
Epoch 9/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 609us/step - loss: 0.0165 - val_loss: 0.0173
Epoch 10/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 644us/step - loss: 0.0161 - val_loss: 0.0166
Epoch 11/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 669us/step - loss: 0.0160 - val_loss: 0.0166
Epoch 12/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 687us/step - loss: 0.0155 - val_loss: 0.0164
Epoch 13/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 675us/step - loss: 0.0168 - val_loss: 0.0168
Epoch 14/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 626us/step - loss: 0.0163 - val_loss: 0.0166
Epoch 15/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 626us/step - loss: 0.0155 - val_loss: 0.0175
Epoch 16/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 598us/step - loss: 0.0163 - val_loss: 0.0164
Epoch 17/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 586us/step - loss: 0.0154 - val_loss: 0.0163
Epoch 18/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 590us/step - loss: 0.0158 - val_loss: 0.0169
Epoch 19/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 624us/step - loss: 0.0164 - val_loss: 0.0164
Epoch 20/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 610us/step - loss: 0.0158 - val_loss: 0.0165
Conv1D Model MSE: 0.0160614512860775

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
Predicted next coordinate: [[ 512320.25 2836554.5 ]]
```

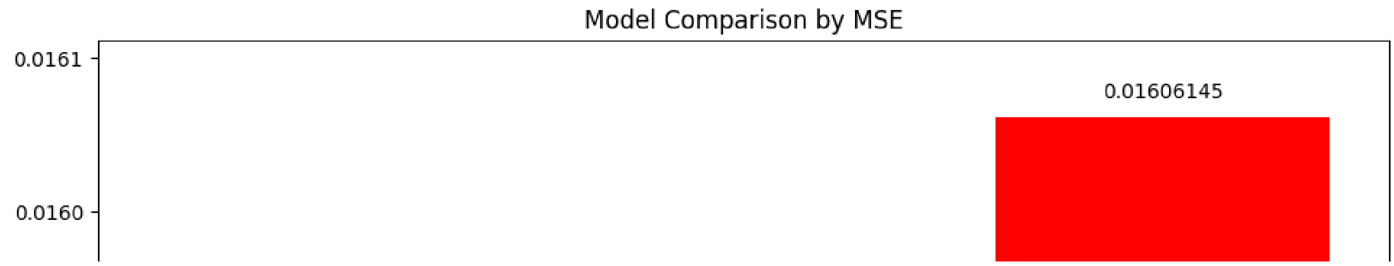
In [21]:

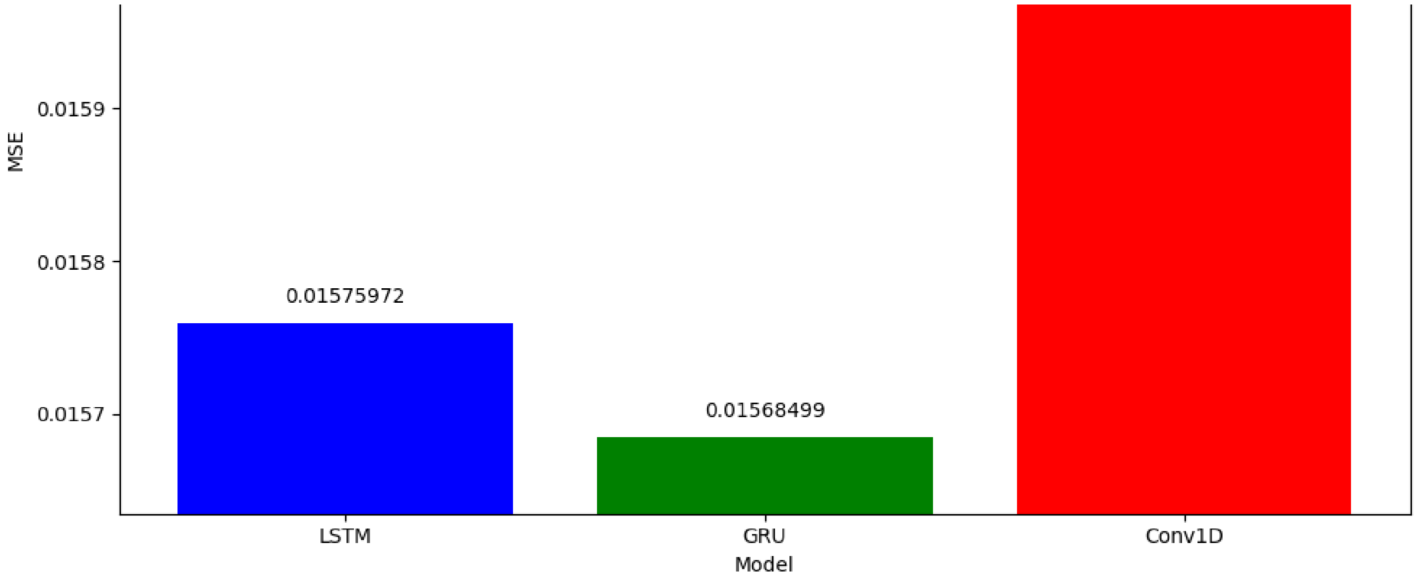
```python
plt.figure(figsize=(10, 6))
bars = plt.bar(model_names, mse_values, color=['blue', 'green', 'red'])

# Annotate the MSE values on top of the bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.00001, round(yval, 8), ha='center
', va='bottom')

# Adjust the Y-axis to better highlight differences
min_mse = min(mse_values)
max_mse = max(mse_values)
plt.ylim(min_mse - 0.00005, max_mse + 0.00005)  # Adjust these values as needed to best
fit your data

plt.title('Model Comparison by MSE')
plt.xlabel('Model')
plt.ylabel('MSE')
plt.xticks(model_names)
plt.tight_layout()
plt.show()
```





In [10]:

```
print(f'Predicted next coordinate: {next_coordinate_scaled}')
```

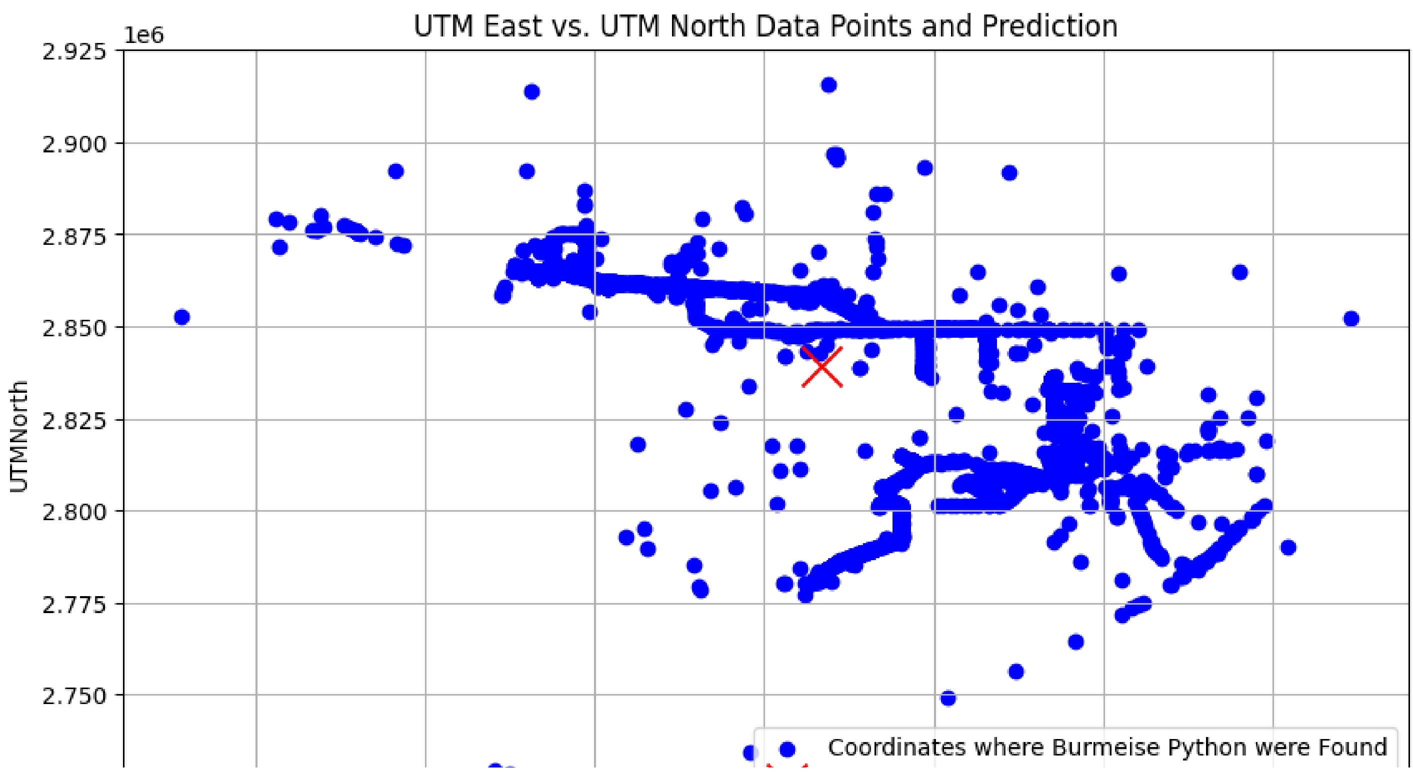
```
Predicted next coordinate: [[ 508354.5 2839020.8]]
```

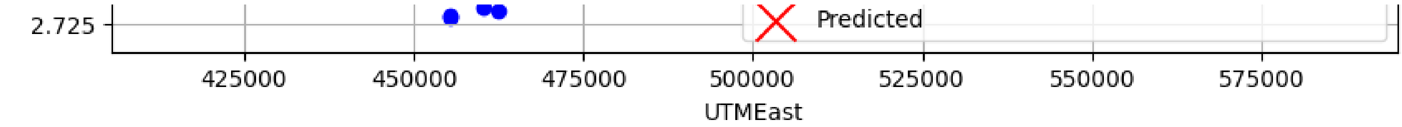
In [18]:

```
if len(next_coordinate_scaled) == 1 and isinstance(next_coordinate_scaled[0], (list, np.
ndarray)):
    predicted_east, predicted_north = next_coordinate_scaled[0]
else:
    predicted_east, predicted_north = next_coordinate_scaled

plt.figure(figsize=(10, 6))  # Set the figure size for better readability
plt.scatter(data['UTMEast'], data['UTMNorth'], marker='o', color='b', label='Coordinates
where Burmeise Python were Found')
plt.scatter(predicted_east, predicted_north, marker='x', color='r', s=300,label='Predicte
d')

plt.title('UTM East vs. UTM North Data Points and Prediction')
plt.xlabel('UTMEast')
plt.ylabel('UTMNorth')
plt.legend()
plt.grid(True)
plt.show()
```




2.725
Predicted
425000
450000
475000
500000
525000
550000
575000
UTMEast

In [ ]: