CS154 Final

Prof. Watson

December 22, 2024

Gaussian Process for Time Series Forecasting of Cell Migration

The so-called electrotaxis is an important phenomenon involved in many biological processes, such as wound healing, in which cells such as macrophages and keratinocytes migrate to the injury site following an electric stimulus, created by an electrical potential caused by the imbalance of ions in the wound and around the wound.

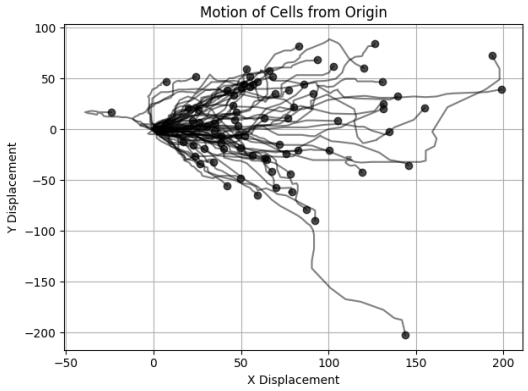


Figure 1: Data of Cell Migration under Electrotaxis. Data from Arocena et al. (2017)

Gaussian Process (GP) is a machine learning model that can be used to predict time series events with uncertainty. Unlike a regular regression model, GP does not map each input to a single output when predicting new data. However, it maps the input to a normal distribution. Each input value has a normal distribution mapped to it, in which the output is believed to be sampled. This model performs well for continuous data and time series. The data set used for this assignment is composed of the positions (x, y) of 77 cells in 37 time stamps, under electrotaxis from Arocena et al. (2017), resulting in $37 \times 77 = 2849$

data points. We first modeled it using a long-short-term memory (LSTM) recurrent neural network (LSTM-RNN) adapted from Sargent et al. (2022) and then compared it with the GP. Since LSTM-RNN was a model already discussed, here we will focus on the main mathematics behind the functioning of a GP. In Figure 1 we can see the migration of cells from the origin under electrotaxis. The migration seems random, however it is highly influenced by a horizontal electric field from left to right.

1. *Justification*

   LSTM-RNN often leads to deterministic states, which is not common in biological systems where noise is ubiquitous. Therefore, we implement a GP to account for any randomness present during cell migration, that can be originated from the sizes of cells, the strength of the electric field, the stiffness of the extracellular matrix, or the interaction between each cell and its environment. It is very hard to account for those factors when building a model; therefore, we need to have a model in which noise is possible. GP has a complexity of $O(n^3)$, therefore, for a dataset with 77 cells with 37 timestamps, GP is feasible.

   GP is a generalization for the Bayesian Linear Regression (BLR), therefore we need to explaining what BLR before exploring GP.

2. *Bayesian Linear Regression*

   When using linear regression, we do not count for the uncertainty of the fit line when estimating noise variance. How are we sure that the line we picked and its distribution is the best that fits our data? To solve that, we use Bayesian Statistics. We want a model that produces functions as samples before seeing the data. That is our prior sample. As the name says, we sample linear functions from BLR. After sampling several linear functions, we update our prior based on the data and get the functions that best fit the data. We compute the noise variance of each of them and then we take its average so that we have our posterior. Suppose that the data points are like magnets that are not too strong so that gravitation and friction are compared forces. The functions are like wires and we have a box full of wires. We put them inside the full and shake it until all functions best get positioned in the box. We take out those wires that did not have enough force to stick to the data points, and we are left with our regression.

   In Figure 2 we have an example of how this process happens when adding new data points. The more data points, the less spread our model is. However, notice a funnel-

like shape around the data. The further we get from the data the less certain we are about the distribution. Therefore, higher the variance.

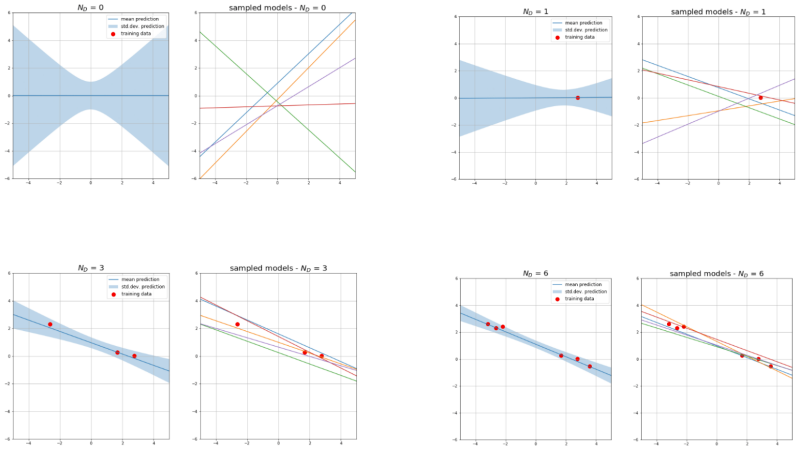


Figure 2: BRL for increasing data points. From Matthias Werner, 2020

The limitation is that the functions are always linear. So for complex patterns in data, we would need more complex functions, in which GP could help us. For that, we need a kernel in order to determine those complex functions.

3. *Kernel*

   A kernel is a function (K) that tells us how similar two inputs ($\mathbf{x}, \mathbf{x'}$) are for a chosen hyperparameter τ. Usually higher positive kernels tell us that the inputs are similar. When it is too negative, they are very dissimilar.

$$K(\mathbf{x}, \mathbf{x'}|\tau)$$

   A common kernel used for 1D-dataset is the Radial Basis Function (RBF):

$$K(\mathbf{x}, \mathbf{x'}|\tau) = \sigma^2 \exp\left(-\frac{1}{2}\left(\frac{\mathbf{x} - \mathbf{x'}}{l}\right)^2\right)$$

   Which has the following heatmap for when $\tau = (\sigma, l) = (1, 1)$.

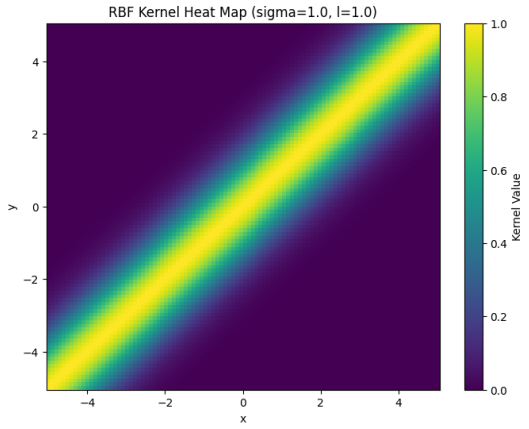


Figure 3: RBF Heat Map

From this kernel, we can sample smooth non-linear functions like the ones below.

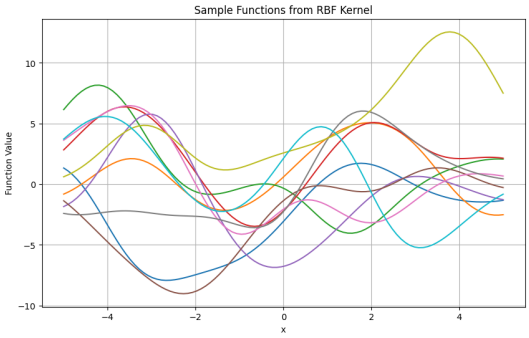


Figure 4: Samples from a RBF kernel

However, for most of the sampling, we using a combination of kernels according to any observed trend present in our data. By adding, multiplying, or using compound kernels. Then we will have a better posterior that better fits the data. Next we discuss in details how the math behind GP operates.

4. *Mathematics behind Gaussian Process*

For our data of cell positions $D = \{(w_i, z_i)\}_{i=1}^{N}$, we want to provide a predictive distribution on test point $\{w_i^{\star}\}_{i=1}^{M}$.

To not be confused with the position of cells, we are referring to input and output as w, z instead of x, y. We are leaving the later to refer to the position of the cell in a 2D grid. Our input $\mathbf{w} = ((x_i, y_i)_{i=1}^{20}$ are the positions of the cell up to a certain value. The remaining points are divided between the output z and the testing data. Let us say that 10 of it goes to the output $\mathbf{z} = ((x_i, y_i)_{i=21}^{30}$ and 7 to the testing point, $\{w_i^\star\}_{i=31}^{37}$.

We assume that $\mathbf{z}$ comes from a noisy observation of some true function $f(x)$:

$$y_i = f(\mathbf{x}_i) + \epsilon_i$$

in which $\epsilon_i \sim \mathcal{N}(0, \sigma_\epsilon^2)$

We can now collect everything we have into matrices and vectors:

$\mathbf{X}$ is the matrix where each row is a vector input; $\mathbf{X}^\star$ is the matrix where each row is a vector test point; $\mathbf{y}$, vector of observed outputs; $\mathbf{f}$, unobserved true function output for input; and $\mathbf{f}^\star$, unobserved true function output for test point.

We also have that $K_{\mathbf{X},\mathbf{X}}$ be an N-by-N matrix of all similarities $K(\mathbf{x}_i, \mathbf{x}_j|\tau)$.

We come to the **Gaussian Process Assumption**, in which $\mathbf{y}$ and $\mathbf{f}^\star$ are distributed as an $(N + M)$-dimensional multivariate Normal:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}^\star \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \hat{\mathbf{K}}_{\mathbf{X},\mathbf{X}} & \mathbf{K}_{\mathbf{X},\mathbf{X}^\star} \\ \mathbf{K}_{\mathbf{X}^\star,\mathbf{X}} & \mathbf{K}_{\mathbf{X}^\star,\mathbf{X}^\star} \end{bmatrix} \right)$$

where $\hat{\mathbf{K}}_{\mathbf{X},\mathbf{X}} = \mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma_i^2 \mathbf{I}$ is responsible for the random variance of the output $\mathbf{y}$. In the case of cell migration we are considering four cases of variance as hyperparameter: $\sigma = [0, .1, .2, .3]$, in which the first indicates a model that there is no randomness.

We are assuming that the entire data set is considered to be a single sample from this multivariate normal distribution. In the case of a time series of events, such as positions of a moving cell, this makes sense if we are considering that all the positions come from a single cell.

The posterior distribution should considerate the data. The normal distribution allows us to have the distribution of the unseen elements based on the seen ones.

$$\mathbf{f}^\star|\mathbf{X}^\star, \mathcal{D} \sim \mathcal{N}(\mu_{\mathbf{f}^\star}, \Sigma_{\mathbf{f}^\star})$$

$$\text{where } \mu_{\mathbf{f}^\star} = \mathbf{K}_{\mathbf{X}^\star,\mathbf{X}^\star} - \mathbf{K}_{\mathbf{X}^\star,\mathbf{X}}\hat{\mathbf{K}}^{-1}_{\mathbf{X},\mathbf{X}}\mathbf{K}_{\mathbf{X},\mathbf{X}^\star} \text{ and } \Sigma_{\mathbf{f}^\star} = \mathbf{K}_{\mathbf{X}^\star,\mathbf{X}}\hat{\mathbf{K}}^{-\mathbf{1}}_{\mathbf{X},\mathbf{X}}\mathbf{y}$$

This indicates that the posterior is computed by sampling from the kernel after conditioning to the data. The distribution would be computed by slicing the multivariate distribution for each value of y_i. The noise variances tell us how much we are fitting the distribution around the data. A variance of zero would produce a simple regression.

We can use the model to help us finding the best choice of parameters.

5. *Hyperparameter Selection*

   We pick the best hyperparameters τ and σ_ϵ^2 by maximizing the log-likelihood for $\mathbf{y}$ after integrating out possible $f(\cdot)s$.:

$$\log p(\mathbf{y}|\mathbf{X}, \tau, \sigma_\epsilon^2) = \log \int p(\mathbf{y}|\tilde{\mathbf{f}}, \sigma_\epsilon^2) p(\tilde{\mathbf{f}}|\mathbf{X}, \tau) d\tilde{\mathbf{f}} =$$
$$= \log \mathcal{N}(\mathbf{y}|0, \hat{\mathbf{K}}_{\mathbf{X},\mathbf{X}})$$

   And we have that $\log \mathcal{N}(\mathbf{y}|0, \hat{\mathbf{K}}_{\mathbf{X},\mathbf{X}})$ is differentiable with respect to τ and σ_ϵ^2.

6. *Drawbacks of Gaussian Processes*

   Although GPs are very useful in certain cases, like forecasting of time series events, there are still some drawbacks worthy of consideration before using such models.

   The first one tells us that my inverting the matrix $\hat{\mathbf{K}}_{\mathbf{X},\mathbf{X}}$ takes $O(N^3)$ time, which can result in a very computational expensive model for large input N.

   The second one is that it requires a considerable amount of work designing the kernel used to sample our priors and posteriors.

   And the third is that certain kernels, such as RBF, might break down in higher dimensions, indicating that the predictions will nto be good enough.

However GPs are still powerful for the cases in which we have a small data set, in lower dimensions in which there is a visible trend. These conditions actually fit the cell migration dataset, in which the trend is any kernel that pictures a motion from left to right, such RBF, given the electric field distribution.

Appendix Code

For this assignment, we will implement and improve a machine learning model designed by Sargent et al. (2022) to predict the migration of cell under an external electric field (electrotaxis/galvanotaxis) using a recurrent neural network. Since the data used to traing the original model is not available, I used another data set from Arocena et al. (2017) in which the positions of cells were calculated under a electric field of 250 mV/mm. It is important to mention that model was shown to be transferable, since it predicted with high accuracy the migration of another dataset. Therefore, we expected that by training the model in this new data we will not lose accuracy.

Besides that, we will also implement randomness to the original model by adding a noise drawn from a Gaussian distribution to prevent determinism and be more accurate to real biological process of cell migration, where many factors, such as electric, mechanical, thermal, and chemical influence the migration. Here we will only consider cell migration under electric cues, therefore we add noise to be able to better model the cell migration in other unknown conditions. Therefore we should also compare the perfomance of the model between the inclusion and exclusion of ramdon variables.

```
# Sargent et al (2022)

import pandas as pd
import numpy as np
import math
from keras import backend as K
from keras.models import Sequential
from keras.models import load_model
from keras.layers import LSTM,Dense
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import os
import random
import tensorflow as tf
```

Pre-processing data

The original dataset is composed by the migration in dozens of conditions. For this model we will only consider the migration of wild type cells under electric field in a quartz plane (there is no friction). We desconsider all null values and convert the string numbers back to integers.

```
df = pd.read_csv('/content/cell+trajectories+data.csv')

column = 'wt neural progenitor trajectories 250mV/mm flat quartz 9-3-
10'
# df.set_index(column, inplace=True)
```

```
index_column = list(df.columns).index(column)

# display only the values referent to the interest column
column_indexes = [index_column, index_column+1, index_column+2]
new_df = df.iloc[:,column_indexes]

# replace the column names with the spatial and temporal values
new_df.columns = new_df.iloc[0]
new_df = new_df[1:].reset_index(drop=True)
new_df = new_df.dropna() # clean null values
df = new_df.apply(pd.to_numeric, errors='coerce')

df
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 2849,\n  \"fields\":
[\n    {\n      \"column\": \"X\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 120.06821196774669,\n
\"min\": 0.0,\n        \"max\": 427.685,\n
\"num_unique_values\": 648,\n        \"samples\": [\n
427.066,\n          283.884,\n          252.272\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Y\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 86.68271868725562,\n
\"min\": 0.0,\n        \"max\": 319.834,\n
\"num_unique_values\": 486,\n        \"samples\": [\n
171.694,\n          174.793,\n          206.405\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"time point\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
10,\n        \"min\": 1,\n        \"max\": 37,\n
\"num_unique_values\": 37,\n        \"samples\": [\n          18,\n
14,\n          5\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


The data lists 37 time-series positions of 77 cells. For a better visualization and training, we will decrease the initial position from each position so that all cells start migration from the origin, and we add a cell id, which will be useful when training and testing our model.

```
# Add a 'cell_id' column to identify each cell (1 to 77)
df['cell_id'] = (df.index // 37) + 1

# Function to shift positions so each cell starts at the origin
def shift_to_origin(df):
    shifted_data = []
    for cell_id, group in df.groupby('cell_id'):
        # Reset the origin for each cell
        origin_x = group.iloc[0]['X']
        origin_y = group.iloc[0]['Y']
```

```
        # Shift coordinates
        group['X'] = group['X'] - origin_x
        group['Y'] = group['Y'] - origin_y
        shifted_data.append(group)

    return pd.concat(shifted_data)

# Apply the function to shift positions
df_shifted = shift_to_origin(df)

# Display the updated DataFrame
print(df_shifted)
```

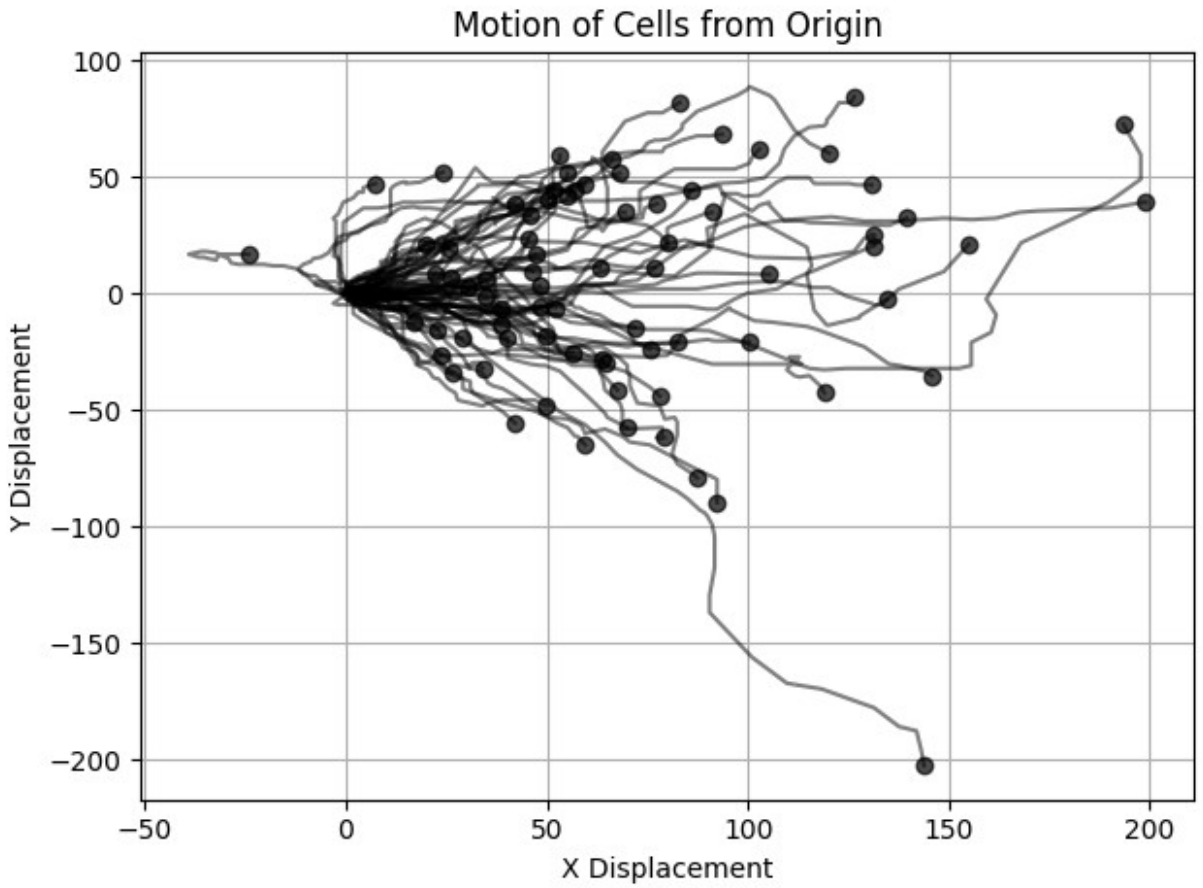
```
0            X      Y  time point  cell_id
0       0.0000  0.000           1        1
1       0.0000  0.000           2        1
2       2.4793 -0.619           3        1
3       4.3388 -1.239           4        1
4      12.3967 -3.719           5        1
...        ...    ...         ...      ...
2844   17.9760 -1.240          33       77
2845   23.5540  1.239          34       77
2846   25.4140  1.239          35       77
2847   28.5130  2.479          36       77
2848   30.3720  2.479          37       77

[2849 rows x 4 columns]
```

Data Visualization

Below we see the migrations of cells.

```
cells = 77
in_cols = ['X', 'Y']
out_cols = ['X', 'Y']

for cell_idx in range(cells):  # Iterate through 77 cells
    # Each cell has 37 time points, extract corresponding rows
    cell_data = df_shifted.iloc[cell_idx * 37:(cell_idx + 1) * 37]

    x_positions = cell_data['X'].values
    y_positions = cell_data['Y'].values

    plt.plot(cell_data['X'].values,
             cell_data['Y'].values,
             label=f'Cell {cell_idx}',
             alpha=0.5, color='black')
```

```
    plt.plot(x_positions[-1], y_positions[-1], marker='o', color='k',
alpha=0.7)

plt.title("Motion of Cells from Origin")
plt.xlabel("X Displacement")
plt.ylabel("Y Displacement")
plt.grid(True)
plt.tight_layout()
plt.show()
```



Splitting Data

Below we split the data into testing, training and validation.

```
df = df_shifted
# Function to create dataset for training/testing
def create_dataset(df, lookback=20, in_cols=['X', 'Y'], out_cols=['X',
'Y'], cells=77):
```

```
    trainX, trainY = [], []  # Input and output lists

    for cell_idx in range(cells):  # Iterate through 77 cells
        # Each cell has 37 time points, extract corresponding rows
        cell_data = df.iloc[cell_idx * 37:(cell_idx + 1) * 37]
        cell_input = cell_data[in_cols]
        cell_output = cell_data[out_cols]

        # Slide window for lookback
        for i in range(len(cell_data) - lookback):
            trainX.append(cell_input.iloc[i:i + lookback].values)  # 
Input sequence
            trainY.append(cell_output.iloc[i + lookback].values)   # 
Output value (next step)

    # Convert lists to numpy arrays
    trainX = np.array(trainX)
    trainY = np.array(trainY)

    return trainX, trainY

# Generate train, validation, and test sets
trainX, trainY = create_dataset(df.iloc[:50*37], lookback=20)  # First 
50 cells for training
valX, valY = create_dataset(df.iloc[50*37:60*37], lookback=20)  # Next 
10 cells for validation
testX, testY = create_dataset(df.iloc[60*37:], lookback=20)  # Last 17 
cells for testing

print(trainX.shape, trainY.shape)
print(valX.shape, valY.shape)
print(testX.shape, testY.shape)
```

```
(850, 20, 2) (850, 2)
(170, 20, 2) (170, 2)
(289, 20, 2) (289, 2)
```

Model: LSTM-RNN

Below we train the data in a LSTM-RNN based on the code of Sargent et al. (2022).

```
def typical_model(trainX, trainY, valX, valY, testX, testY, numbers):
    models = []  # List of models
    predictions = []  # List of prediction vectors

    for i in numbers:
        print(f'Training model number {i}')
```

```
        # Build the LSTM model
        model = Sequential()
        model.add(LSTM(80, input_shape=(20, 2)))  # 20 time steps, 2
features
        model.add(Dense(2))  # Output layer predicts two values: x and
y
        model.compile(loss='mean_squared_error', optimizer='adam')

        # Train the model
        model.fit(trainX, trainY, validation_data=(valX, valY),
epochs=50, batch_size=1, verbose=1)
        models.append(model)

        # Predict and evaluate RMSE
        trainPredict = model.predict(trainX)
        valPredict = model.predict(valX)
        testPredict = model.predict(testX)

        trainScore = math.sqrt(mean_squared_error(trainY,
trainPredict))
        valScore = math.sqrt(mean_squared_error(valY, valPredict))
        testScore = math.sqrt(mean_squared_error(testY, testPredict))

        print(f'Training RMSE: {trainScore}, Validation RMSE:
{valScore}, Testing RMSE: {testScore}')

        # Append predictions
        predictions.append(np.concatenate([trainPredict, valPredict,
testPredict], axis=0))

    return models, predictions

# Train LSTM models
models, predictions = typical_model(trainX, trainY, valX, valY, testX,
testY, numbers=range(1))

# Save the best model (optional)
models[0].save("models/lstm_cell_migration.h5")
print("Model saved to disk.")
```

```
Training model number 0
Epoch 1/50

/usr/local/lib/python3.10/dist-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)

850/850 ━━━━━━━━━━━━━━━━━━━━ 13s 14ms/step - loss: 2576.5378 -
val_loss: 421.5470
```

```
Epoch 2/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - loss: 1172.8574 -
val_loss: 165.6267
Epoch 3/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - loss: 763.1058 - val_loss:
81.8488
Epoch 4/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - loss: 358.7076 - val_loss:
57.5214
Epoch 5/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 13s 13ms/step - loss: 271.5177 -
val_loss: 38.8843
Epoch 6/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 18s 11ms/step - loss: 241.8638 -
val_loss: 23.5476
Epoch 7/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - loss: 159.1287 -
val_loss: 19.8255
Epoch 8/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 13s 15ms/step - loss: 127.6351 -
val_loss: 11.7688
Epoch 9/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 11s 13ms/step - loss: 189.8306 -
val_loss: 9.7648
Epoch 10/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 18s 11ms/step - loss: 98.2364 - val_loss:
11.3498
Epoch 11/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 13s 15ms/step - loss: 103.6790 -
val_loss: 11.8366
Epoch 12/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - loss: 84.0609 - val_loss:
12.1235
Epoch 13/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 18s 9ms/step - loss: 75.7538 - val_loss:
7.7096
Epoch 14/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - loss: 76.8380 - val_loss:
11.1503
Epoch 15/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - loss: 60.3344 - val_loss:
8.1439
Epoch 16/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 11s 12ms/step - loss: 51.8638 - val_loss:
11.6088
Epoch 17/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - loss: 57.6644 - val_loss:
13.4703
Epoch 18/50
```

```
850/850 ━━━━━━━━━━━━━━━━━━━━ 13s 15ms/step - loss: 52.6206 - val_loss:
11.4303
Epoch 19/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 17s 10ms/step - loss: 48.4492 - val_loss:
16.4177
Epoch 20/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - loss: 28.2948 - val_loss:
16.0737
Epoch 21/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - loss: 58.6822 - val_loss:
12.8570
Epoch 22/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 12s 13ms/step - loss: 40.0462 - val_loss:
14.6928
Epoch 23/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 23s 17ms/step - loss: 49.5813 - val_loss:
20.4949
Epoch 24/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - loss: 37.1838 - val_loss:
10.4997
Epoch 25/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 12s 14ms/step - loss: 32.6598 - val_loss:
16.0038
Epoch 26/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 17s 10ms/step - loss: 27.6581 - val_loss:
16.9823
Epoch 27/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 13s 15ms/step - loss: 44.5493 - val_loss:
18.3431
Epoch 28/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 12s 14ms/step - loss: 36.4070 - val_loss:
17.0739
Epoch 29/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 19s 13ms/step - loss: 31.7712 - val_loss:
12.8544
Epoch 30/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - loss: 37.3344 - val_loss:
14.3014
Epoch 31/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 18s 9ms/step - loss: 31.2022 - val_loss:
14.1400
Epoch 32/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 11s 10ms/step - loss: 18.8763 - val_loss:
16.4926
Epoch 33/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 13s 13ms/step - loss: 23.7091 - val_loss:
13.3913
Epoch 34/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 19s 10ms/step - loss: 29.2872 - val_loss:
```

```
15.7721
Epoch 35/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 10ms/step - loss: 20.1813 - val_loss:
16.4731
Epoch 36/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - loss: 22.2962 - val_loss:
17.6857
Epoch 37/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - loss: 27.4030 - val_loss:
14.6477
Epoch 38/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - loss: 23.2143 - val_loss:
18.9466
Epoch 39/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - loss: 30.7097 - val_loss:
19.4406
Epoch 40/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - loss: 21.1353 - val_loss:
18.1565
Epoch 41/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - loss: 23.2964 - val_loss:
17.0271
Epoch 42/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - loss: 26.3420 - val_loss:
16.2085
Epoch 43/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - loss: 25.5948 - val_loss:
20.8399
Epoch 44/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 11s 13ms/step - loss: 13.4867 - val_loss:
22.9913
Epoch 45/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - loss: 14.9354 - val_loss:
20.5775
Epoch 46/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 12s 14ms/step - loss: 16.2969 - val_loss:
19.0737
Epoch 47/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 17s 10ms/step - loss: 18.0249 - val_loss:
27.2840
Epoch 48/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - loss: 15.0579 - val_loss:
18.7386
Epoch 49/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - loss: 12.7033 - val_loss:
20.1560
Epoch 50/50
850/850 ━━━━━━━━━━━━━━━━━━━━ 12s 14ms/step - loss: 22.8092 - val_loss:
16.0416
27/27 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

Training RMSE: 3.5006343623948033, Validation RMSE: 4.005197912295344,
Testing RMSE: 2.2699591964381765
Model saved to disk.
```

Model Prediction and Accuracy

```
# Load trained model
model = models[0]  # Choose the first trained model

# Predict positions for all cells
df['pred_x'] = np.nan
df['pred_y'] = np.nan

lookback = 20
for cell_idx in range(77):  # Iterate through all cells
    cell_data = df.iloc[cell_idx * 37:(cell_idx + 1) * 37]
    x_input = cell_data[['X', 'Y']].values

    # Generate sliding window predictions
    for t in range(lookback, len(cell_data)):
        input_window = x_input[t - lookback:t].reshape(1, lookback, 2)
# Reshape for LSTM
        pred_x, pred_y = model.predict(input_window)[0]

        df.at[cell_idx * 37 + t, 'pred_x'] = pred_x
        df.at[cell_idx * 37 + t, 'pred_y'] = pred_y

# Save the updated DataFrame
df.to_csv('data/cell_migration_predictions.csv', index=False)
print("Predictions saved to file.")

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step
```

Below we can see the prediction along the true values. We see that the values are very similar for the last 17 positions.

```python
# Create the directory if it doesn't exist
output_dir = 'data'
os.makedirs(output_dir, exist_ok=True)

# Save the updated DataFrame
file_path = os.path.join(output_dir, 'cell_migration_predictions.csv')
df.to_csv(file_path, index=False)
print(f"Predictions saved to file at {file_path}.")
print("Predictions saved to file.")
print(df.head(40))
```

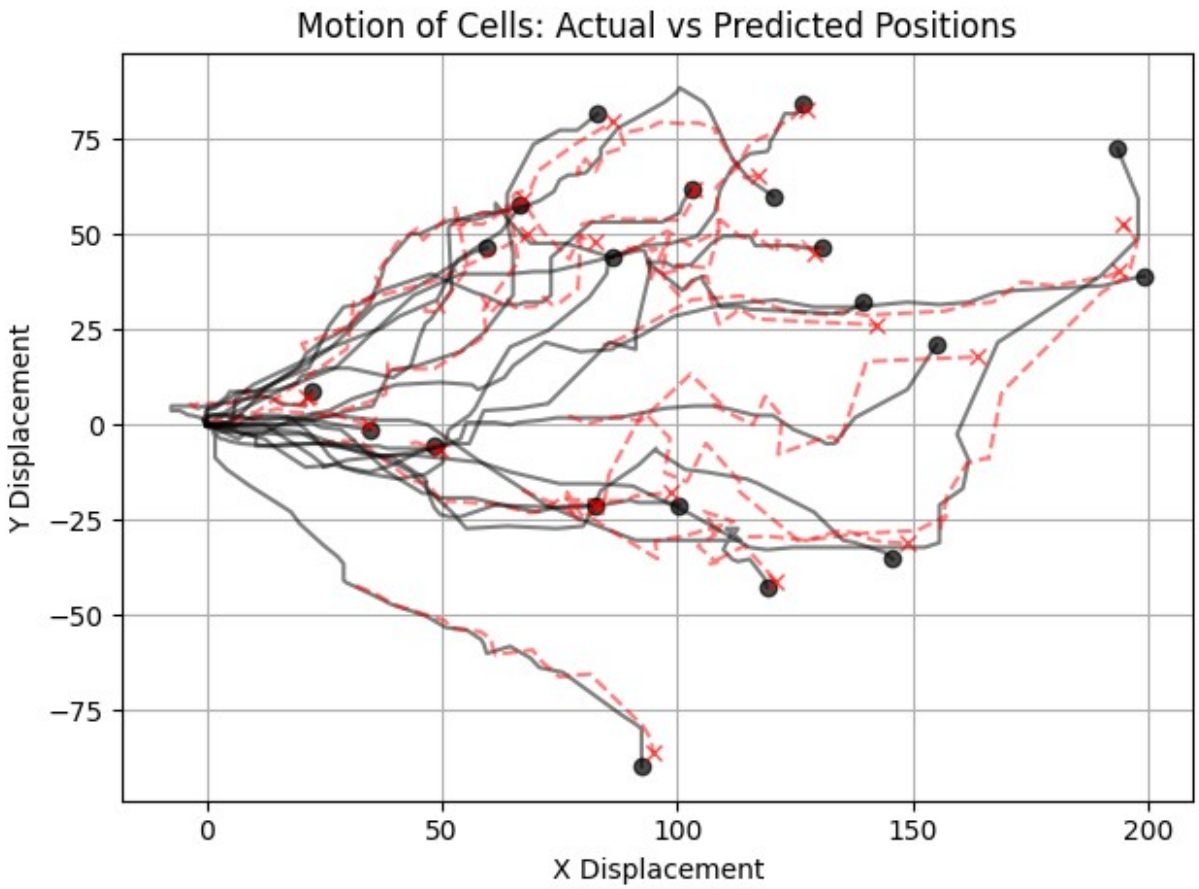
```
Predictions saved to file at data/cell_migration_predictions.csv.
Predictions saved to file.
```

0	X	Y	time point	cell_id	pred_x	pred_y
0	0.0000	0.000	1	1	NaN	NaN
1	0.0000	0.000	2	1	NaN	NaN
2	2.4793	-0.619	3	1	NaN	NaN
3	4.3388	-1.239	4	1	NaN	NaN
4	12.3967	-3.719	5	1	NaN	NaN
5	14.8760	-6.198	6	1	NaN	NaN
6	21.0744	-11.157	7	1	NaN	NaN
7	27.2727	-10.537	8	1	NaN	NaN
8	32.2314	-12.396	9	1	NaN	NaN
9	40.9091	-14.256	10	1	NaN	NaN
10	46.4871	-13.016	11	1	NaN	NaN
11	50.2061	-9.917	12	1	NaN	NaN
12	50.2061	-6.818	13	1	NaN	NaN
13	51.4461	-1.239	14	1	NaN	NaN
14	55.7851	2.480	15	1	NaN	NaN
15	57.0251	4.339	16	1	NaN	NaN
16	60.7441	13.017	17	1	NaN	NaN
17	64.4631	20.455	18	1	NaN	NaN
18	72.5201	24.794	19	1	NaN	NaN
19	81.1981	33.471	20	1	NaN	NaN
20	84.9171	39.670	21	1	89.019096	44.919495
21	85.5371	44.009	22	1	86.439453	44.606079
22	92.3551	47.108	23	1	89.343979	45.510365
23	98.5541	47.728	24	1	97.859543	50.972984
24	104.1321	49.587	25	1	102.519341	47.196663
25	107.8511	55.785	26	1	108.746086	50.838261
26	109.0911	61.984	27	1	108.244438	62.580387
27	111.5701	67.562	28	1	112.276642	67.323212
28	115.2891	71.281	29	1	115.337250	73.888611
29	119.0081	71.901	30	1	117.934540	75.523178
30	119.6281	73.141	31	1	120.285049	76.706192
31	119.6281	74.380	32	1	121.270454	77.860992
32	120.8681	76.860	33	1	121.724731	78.819397
33	122.7271	81.819	34	1	124.445488	80.777542
34	125.2061	81.819	35	1	125.691681	82.891647
35	125.8261	82.438	36	1	127.001419	82.772171
36	126.4461	84.298	37	1	127.381233	83.024933
37	0.0000	0.000	1	2	NaN	NaN
38	0.0000	0.000	2	2	NaN	NaN
39	0.0000	0.000	3	2	NaN	NaN

Below we can visualize how the predictions compare with the real motions of cells. We can see that the predictions are not bad.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
# Assuming df_shifted is already defined with the provided dataset
cells = 20  # Total number of cells in the dataset
in_cols = ['X', 'Y']
out_cols = ['pred_x', 'pred_y']

for cell_idx in range(cells):  # Iterate through each cell
    # Each cell has 37 time points, extract corresponding rows
    cell_data = df_shifted.iloc[cell_idx * 37:(cell_idx + 1) * 37]

    # Actual positions
    x_positions = cell_data['X'].values
    y_positions = cell_data['Y'].values

    # Predicted positions
    pred_x_positions = cell_data['pred_x'].values
    pred_y_positions = cell_data['pred_y'].values

    # Plot actual positions (in black)
    plt.plot(x_positions, y_positions, label=f'Cell {cell_idx}
Actual', alpha=0.5, color='black')
    plt.plot(x_positions[-1], y_positions[-1], marker='o', color='k',
alpha=0.7)

    # Plot predicted positions (in red for distinction)
    plt.plot(pred_x_positions, pred_y_positions, label=f'Cell
{cell_idx} Predicted', linestyle='--', alpha=0.5, color='red')
    plt.plot(pred_x_positions[-1], pred_y_positions[-1], marker='x',
color='red', alpha=0.7)

plt.title("Motion of Cells: Actual vs Predicted Positions")
plt.xlabel("X Displacement")
plt.ylabel("Y Displacement")
plt.grid(True)
# plt.legend(loc='best', fontsize='small')
plt.tight_layout()
plt.show()
```



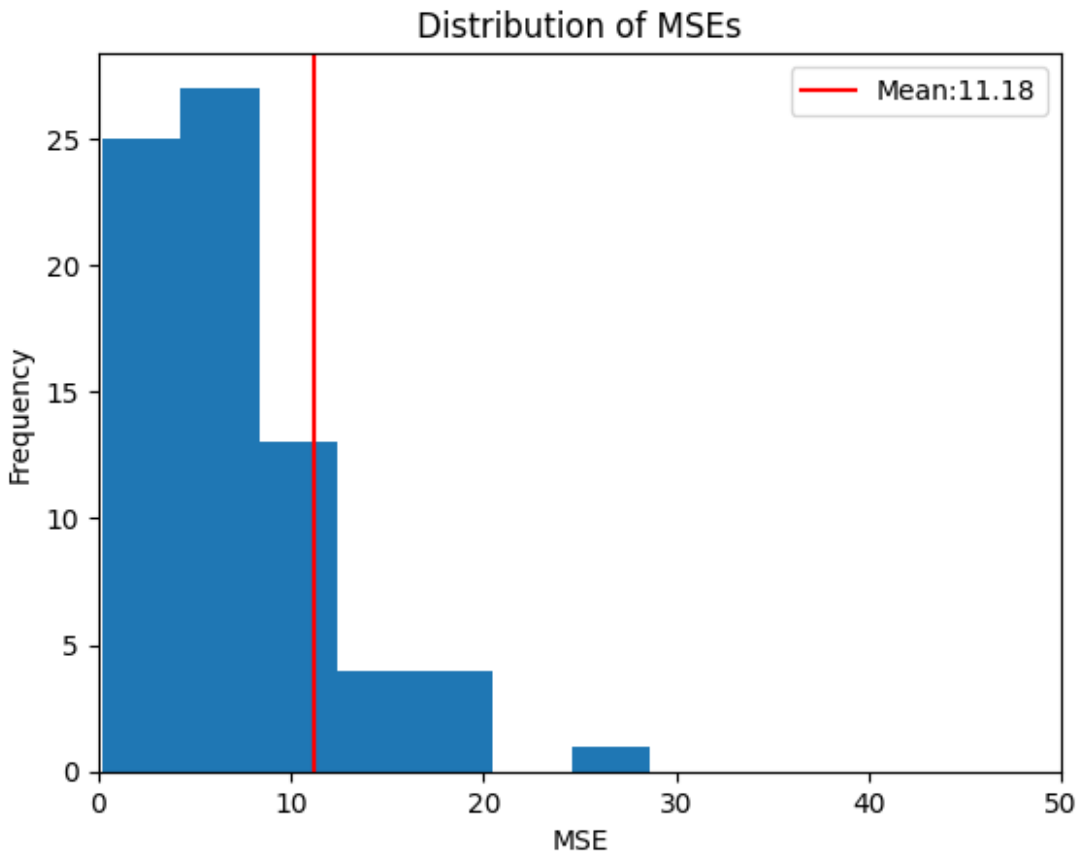
We can see how accurate the model is in predicting the next positions of the cells by the overlapping between the predicted positions and the actual positions. However to quantify the model perfomance we need to discuss numeric metrics. For this purpose we analyse the accuracy

For the accuracy we will use the mean square error (MSE) between the prediction and the real value. Below we see the histogram for MSE and the mean value of 11.17. Indicating that there an error of almost 13 units, which is very low considering that the migration reaches up two magnitudes. There is a global error of 2.1, which is calculated as the mean of the absolute difference between the predicted and the actual positions.

```
# Assuming df_shifted is already defined with the provided dataset
cells = 77  # Total number of cells in the dataset
in_cols = ['X', 'Y']
out_cols = ['pred_x', 'pred_y']

# Store errors and metrics
mse_list = []
accuracies = []
all_actual = []
all_predicted = []
```

```
for cell_idx in range(cells):  # Iterate through each cell
    # Each cell has 37 time points, extract corresponding rows
    cell_data = df_shifted.iloc[cell_idx * 37:(cell_idx + 1) * 37]

    # Actual positions
    x_positions = cell_data['X'].values
    y_positions = cell_data['Y'].values

    # Predicted positions
    pred_x_positions = cell_data['pred_x'].values
    pred_y_positions = cell_data['pred_y'].values

    # Filter last 17 positions for comparison
    actual_positions = np.vstack((x_positions[-17:], y_positions[-
17:])).T
    predicted_positions = np.vstack((pred_x_positions[-17:],
pred_y_positions[-17:])).T

    # Compute MSE
    mse = mean_squared_error(actual_positions, predicted_positions)
    mse_list.append(mse)

    # Collect all actual and predicted values for global metrics
    all_actual.extend(actual_positions)
    all_predicted.extend(predicted_positions)


# Compute global MSE
print(f"Mean Squared Error (Global): {np.mean(mse_list):.4f}")

plt.xlim(0, 50)
plt.hist(mse_list, bins=50)
plt.axvline(np.mean(mse_list), color='red', label=f'Mean:
{np.round(np.mean(mse_list), 2)}')
plt.legend()
plt.title("Distribution of MSEs")
plt.xlabel("MSE")
plt.ylabel("Frequency")
plt.show()

# Accuracy and Classification Report (Simplified Example for Binary
Classification)
all_actual = np.array(all_actual)
all_predicted = np.array(all_predicted)

# Additional Metrics: Example
mae = np.mean(np.abs(all_actual - all_predicted))
print(f"Mean Absolute Error (Global): {mae:.4f}")
```

```
Mean Squared Error (Global): 11.1784
```



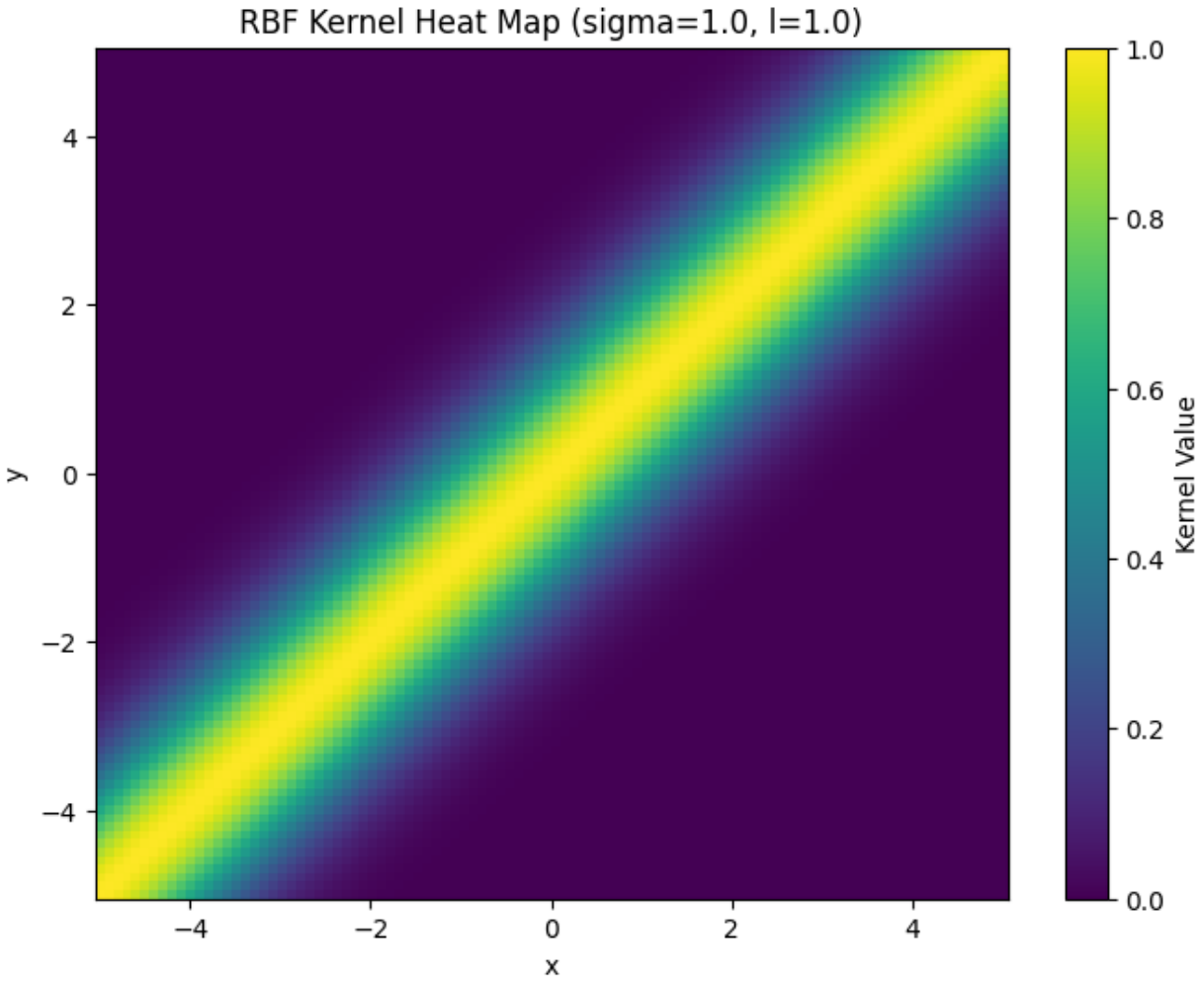

```
Mean Absolute Error (Global): 2.1020
```

Model: Gaussian Process (GP)

Below we will train the same data in a GP. As discussed, a GP uses kernel to draw sample functions. Below we show how the Kernel Basis Function (KBF) is by showing its heatmap and some samples drawn from it.

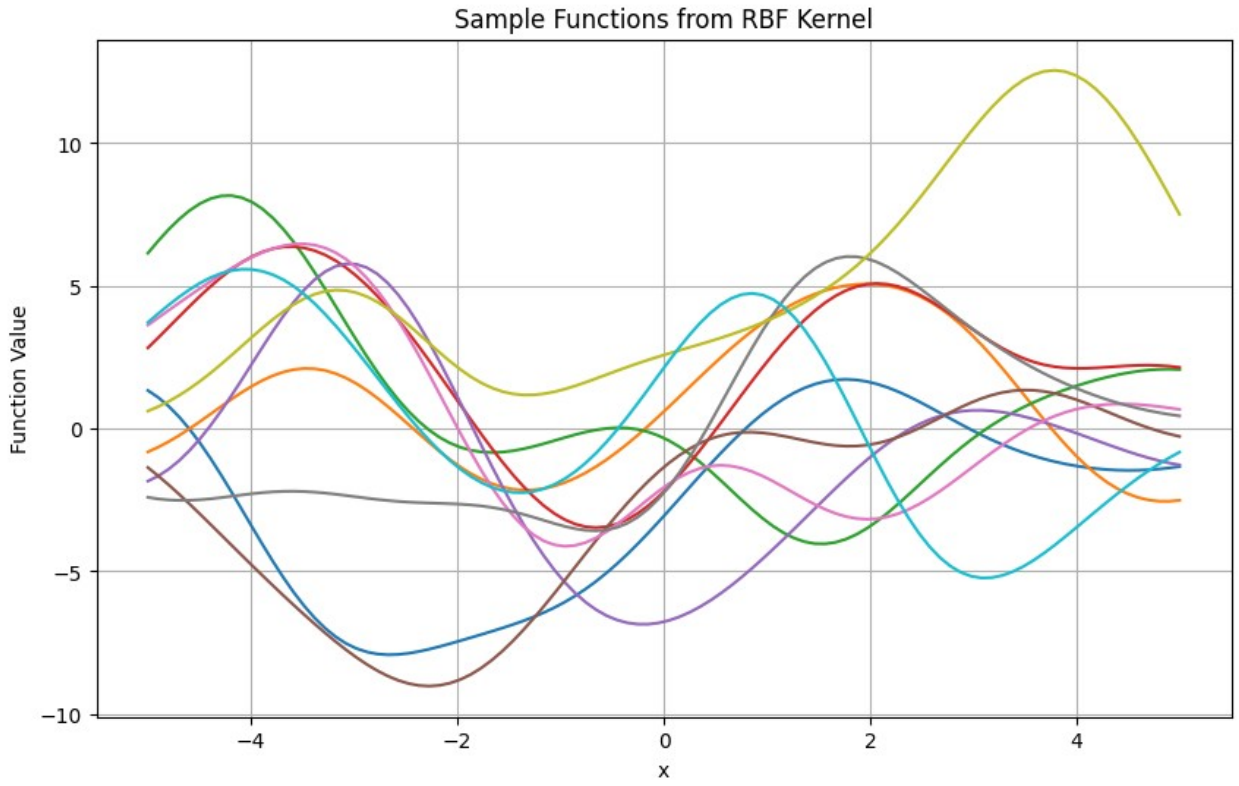
```python
# Define the RBF kernel function
def rbf_kernel(x, y, sigma=1.0):
    return np.exp(-np.square(x - y) / (2 * sigma**2))

# Create a 1D grid of points
x = np.linspace(-5, 5, 100)  # 100 points from -5 to 5
y = np.linspace(-5, 5, 100)  # Same range for y

# Create a meshgrid for the kernel evaluation
X, Y = np.meshgrid(x, y)

# Evaluate the kernel for all pairs of points
sigma = 1.0  # Set the value of sigma
```

```
Z = rbf_kernel(X, Y, sigma)

# Plot the heat map
plt.figure(figsize=(8, 6))
c = plt.pcolormesh(X, Y, Z, shading='auto', cmap='viridis')
plt.colorbar(c, label='Kernel Value')
plt.title(f"RBF Kernel Heat Map (sigma={sigma}, l=1.0)")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```




```
# Define the RBF kernel function
def rbf_kernel(x, y, sigma=1.0):
    return np.exp(-np.square(x - y) / (2 * sigma**2))

# Create a 1D grid of points
x = np.linspace(-5, 5, 100)  # 100 points from -5 to 5
y = np.linspace(-5, 5, 100)  # Same range for y
```

```
# Create a meshgrid for the kernel evaluation
X, Y = np.meshgrid(x, y)

# Evaluate the kernel for all pairs of points
sigma = 1.0  # Set the value of sigma
Z = rbf_kernel(X, Y, sigma)

# Sample functions from the kernel
# Generate a set of random weights to combine kernel values
np.random.seed(42)  # For reproducibility


number_functions = 10  # Number of sample functions to generate
# Plot sample functions
plt.figure(figsize=(10, 6))

for i in range(0, number_functions):
    weights = np.random.randn(len(x))
    # Compute sample functions by summing weighted kernel values
    sample_functions = np.array([np.dot(weights, rbf_kernel(x, xi,
sigma)) for xi in x])
    plt.plot(x, sample_functions, label="Sample Function")

plt.title("Sample Functions from RBF Kernel")
plt.xlabel("x")
plt.ylabel("Function Value")
plt.grid()
plt.show()
```



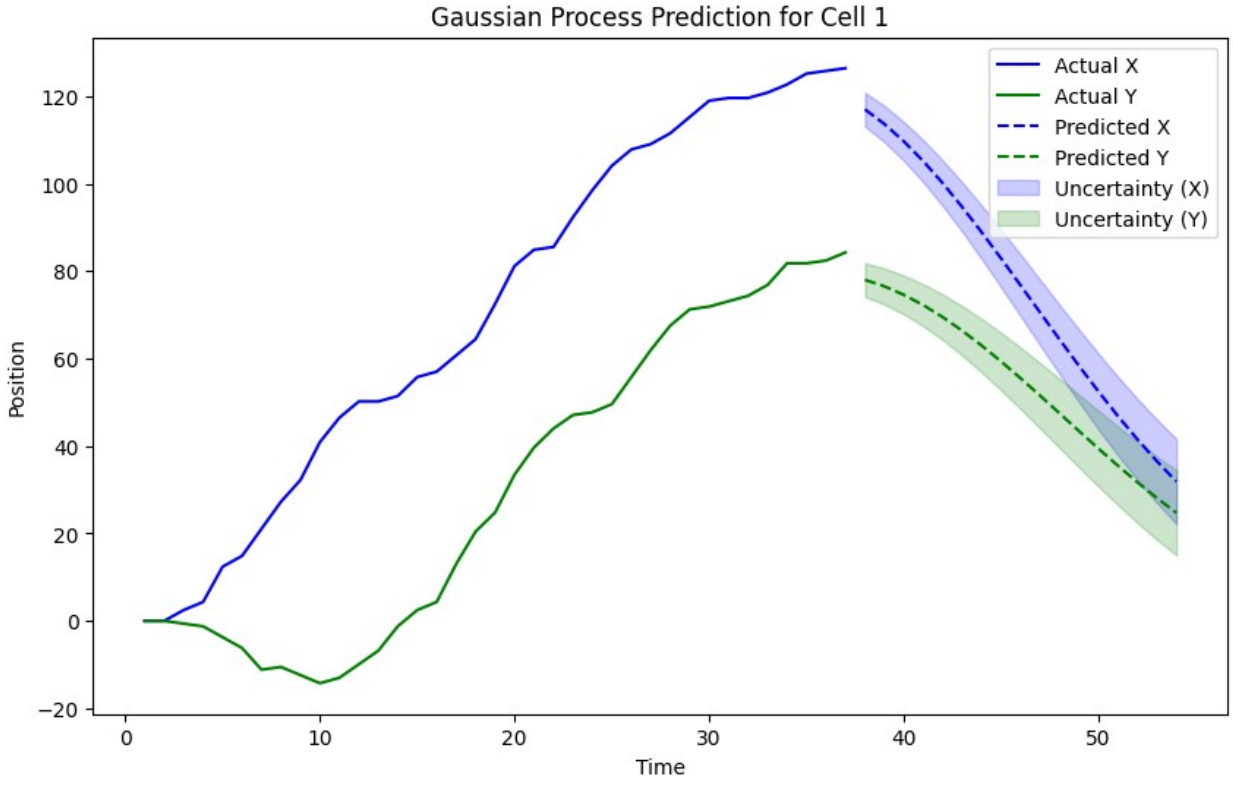
Below we train the cells in a GP for a single cell and see the difference between the actual path and the predicted path.

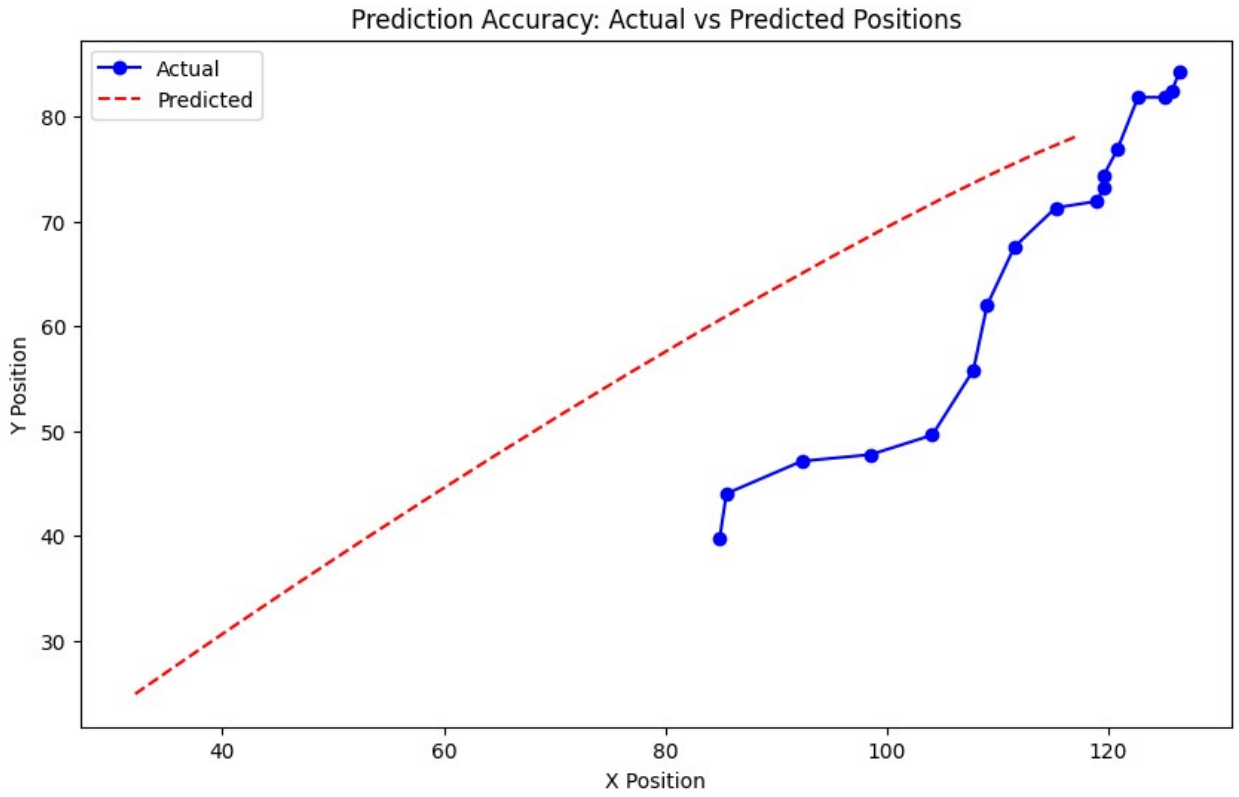
```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C,
WhiteKernel
from sklearn.metrics import mean_squared_error, r2_score

# Load data (assuming a CSV format for simplicity)
data = df

# Prepare data for a specific cell
def prepare_data_for_cell(cell_id, df):
    cell_data = df[df['cell_id'] == cell_id]
    X_time = cell_data['time point'].values.reshape(-1, 1)
    y_position = cell_data[['X', 'Y']].values
    return X_time, y_position

# GP model for a single cell
def train_gp_for_cell(X_train, y_train, X_future):
    kernel = C(1.0, (1e-2, 1e2)) * RBF(length_scale=1.0,
length_scale_bounds=(1e-2, 1e2)) + WhiteKernel()
    gp = GaussianProcessRegressor(kernel=kernel,
n_restarts_optimizer=10)
    gp.fit(X_train, y_train)
    y_pred, y_std = gp.predict(X_future, return_std=True)
```

```
    return y_pred, y_std, gp

# Calculate accuracy metrics
def compute_accuracy(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    return mse, r2

# Visualize accuracy
def plot_accuracy(y_true, y_pred):
    plt.figure(figsize=(10, 6))
    plt.plot(y_true[:, 0], y_true[:, 1], 'bo-', label="Actual")
    plt.plot(y_pred[:, 0], y_pred[:, 1], 'r--', label="Predicted")
    plt.xlabel("X Position")
    plt.ylabel("Y Position")
    plt.title("Prediction Accuracy: Actual vs Predicted Positions")
    plt.legend()
    plt.show()

# Select a specific cell to train and predict
cell_id = 1
X_train, y_train = prepare_data_for_cell(cell_id, data)
X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1, 1)

# Train GP and predict
y_pred, y_std, gp_model = train_gp_for_cell(X_train, y_train,
X_future)

# Evaluate accuracy for visualization
X_eval = X_train[-17:]
y_eval = y_train[-17:]
mse, r2 = compute_accuracy(y_eval, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Visualization of predictions
plt.figure(figsize=(10, 6))
plt.plot(X_train, y_train[:, 0], 'b-', label="Actual X")
plt.plot(X_train, y_train[:, 1], 'g-', label="Actual Y")
plt.plot(X_future, y_pred[:, 0], 'b--', label="Predicted X")
plt.plot(X_future, y_pred[:, 1], 'g--', label="Predicted Y")
plt.fill_between(X_future.ravel(), y_pred[:, 0] - y_std[:,0],
y_pred[:, 0] + y_std[:,0], color='blue', alpha=0.2, label="Uncertainty
(X)")
plt.fill_between(X_future.ravel(), y_pred[:, 1] - y_std[:,1],
y_pred[:, 1] + y_std[:,1], color='green', alpha=0.2,
label="Uncertainty (Y)")
plt.xlabel("Time")
plt.ylabel("Position")
plt.title(f"Gaussian Process Prediction for Cell {cell_id}")
```

```
plt.legend()
plt.show()

# Plot accuracy in X-Y space
plot_accuracy(y_eval, y_pred)
```

```
<ipython-input-29-9b891ee27f7d>:46: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1, 1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kernels.py:452: ConvergenceWarning: The optimal value found for dimension 0 of parameter k1__k1__constant_value is close to the specified upper bound 100.0. Increasing the bound and calling fit again may find a better value.
  warnings.warn(

Mean Squared Error: 1996.35016310715
R-squared: -9.682126477009362
```





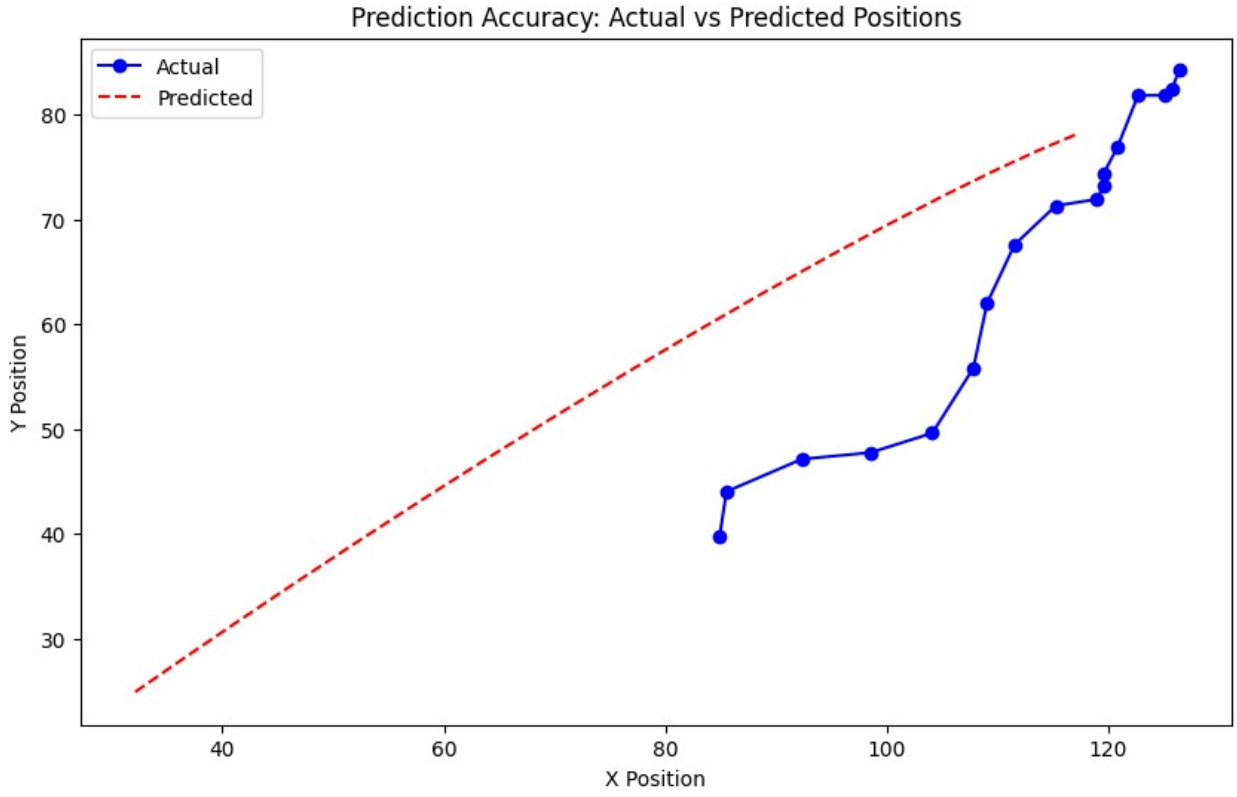
Below we train the same model by taking into account all the 77 cells.

```
# Prepare data for a specific cell
def prepare_data_for_cell(cell_id, df):
    cell_data = df[df['cell_id'] == cell_id]
    X_time = cell_data['time point'].values.reshape(-1, 1)
    y_position = cell_data[['X', 'Y']].values
    return X_time, y_position

# GP model for a single cell
def train_gp_for_cell(X_train, y_train, X_future):
    kernel = C(1.0, (1e-2, 1e2)) * RBF(length_scale=1.0,
length_scale_bounds=(1e-2, 1e2)) + WhiteKernel()
    gp = GaussianProcessRegressor(kernel=kernel,
n_restarts_optimizer=10)
    gp.fit(X_train, y_train)
    y_pred, y_std = gp.predict(X_future, return_std=True)
    return y_pred, y_std, gp

# Calculate accuracy metrics
def compute_accuracy(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    return mse, r2
```

```
# Visualize accuracy
def plot_accuracy(y_true, y_pred):
    plt.figure(figsize=(10, 6))
    plt.plot(y_true[:, 0], y_true[:, 1], 'bo-', label="Actual")
    plt.plot(y_pred[:, 0], y_pred[:, 1], 'r--', label="Predicted")
    plt.xlabel("X Position")
    plt.ylabel("Y Position")
    plt.title("Prediction Accuracy: Actual vs Predicted Positions")
    plt.legend()
    plt.show()

# Train and predict for all cells
all_predictions = []
all_actuals = []
mse_list = []
r2_list = []

for cell_id in data['cell_id'].unique():
    X_train, y_train = prepare_data_for_cell(cell_id, data)
    X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-
1, 1)

    y_pred, y_std, gp_model = train_gp_for_cell(X_train, y_train,
X_future)

    X_eval = X_train[-17:]
    y_eval = y_train[-17:]

    mse, r2 = compute_accuracy(y_eval, y_pred)
    mse_list.append(mse)
    r2_list.append(r2)

    all_predictions.append(y_pred)
    all_actuals.append(y_eval)

    # Plot accuracy for each cell
    if cell_id < 5:
        plot_accuracy(y_eval, y_pred)

# Combine predictions for all cells
combined_predictions = np.vstack(all_predictions)
combined_actuals = np.vstack(all_actuals)

plt.figure(figsize=(12, 8))
plt.plot(combined_actuals[:, 0], combined_actuals[:, 1], 'bo-',
label="Actual (All Cells)")
plt.plot(combined_predictions[:, 0], combined_predictions[:, 1],
'r--', label="Predicted (All Cells)")
plt.xlabel("X Position")
plt.ylabel("Y Position")
```

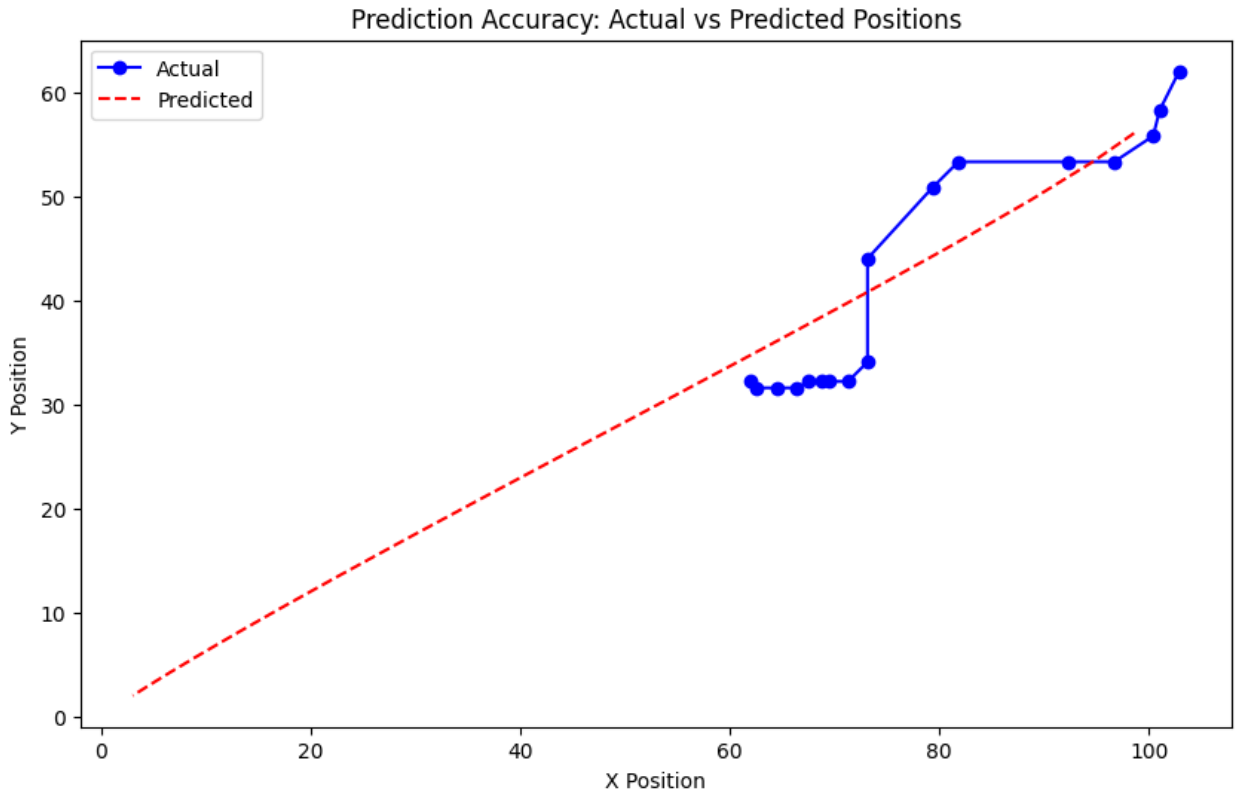
```
plt.title("Combined Prediction Accuracy for All Cells")
plt.legend()
plt.show()

print(f"Overall Mean Squared Error: {np.mean(mse_list)}")
print(f"Overall R-squared: {np.mean(r2_list)}")
```

```
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1, 1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kernels.py:452: ConvergenceWarning: The optimal value found for dimension 0 of parameter k1__k1__constant_value is close to the specified upper bound 100.0. Increasing the bound and calling fit again may find a better value.
  warnings.warn(
```

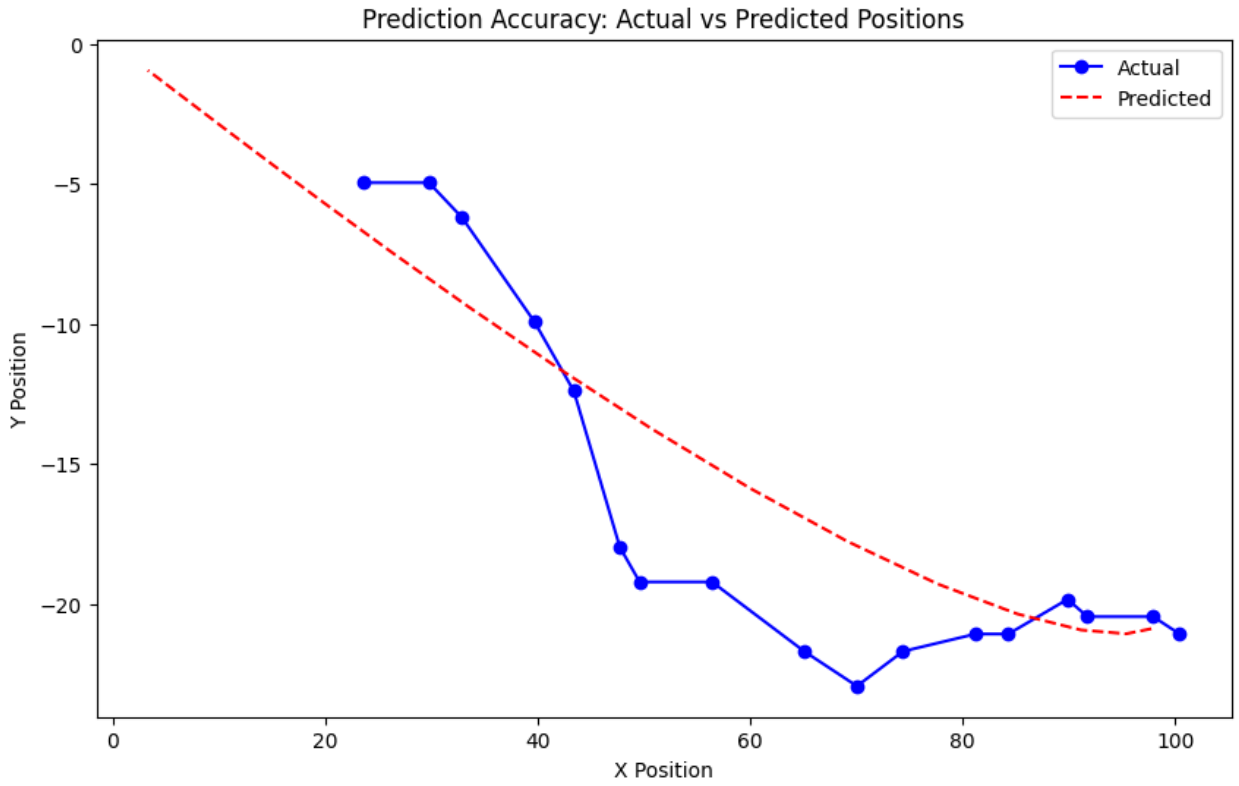



```
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
```

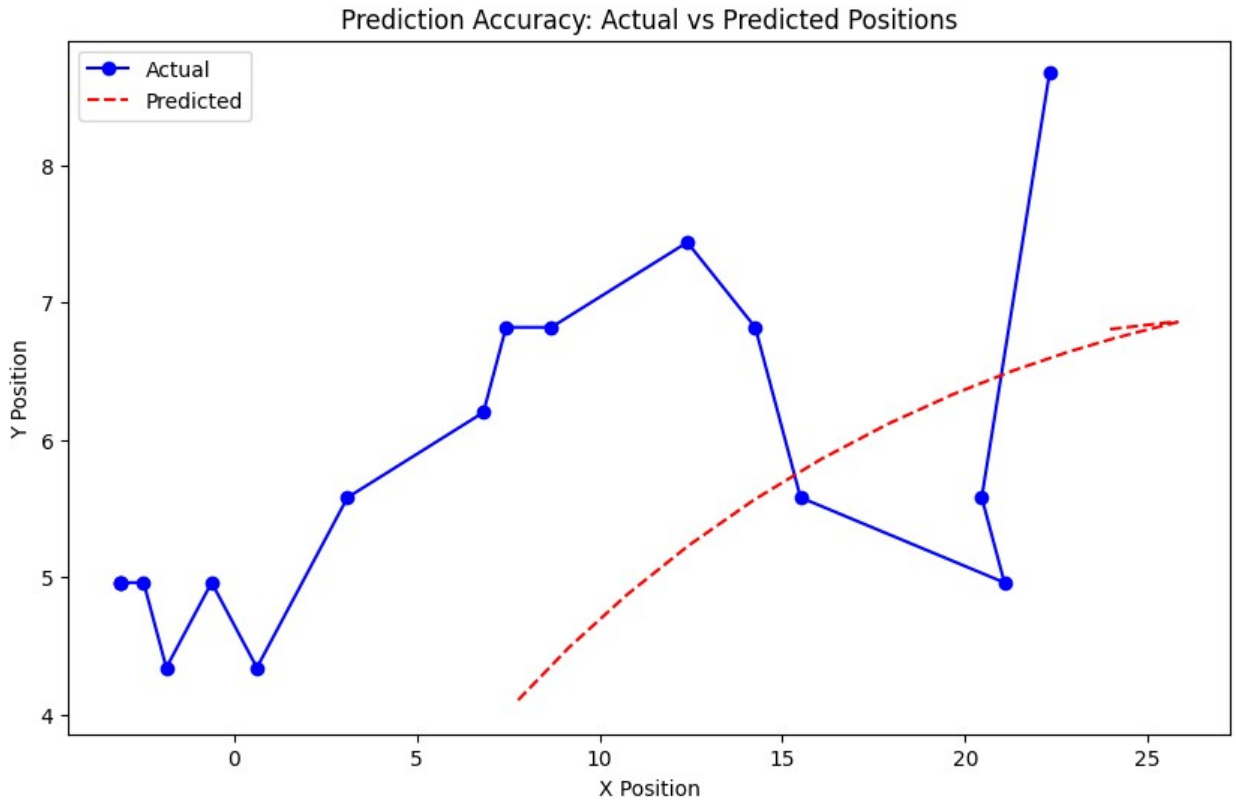
```
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
```




```
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
```




```
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
```




```
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
```

```
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
```

```
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
```

```
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
```

```
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
```

```
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kernels.py:452: ConvergenceWarning: The optimal value found for dimension 0 of parameter k1__k1__constant_value is close to the specified upper bound 100.0. Increasing the bound and calling fit again may find a better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1, 1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kernels.py:452: ConvergenceWarning: The optimal value found for dimension 0 of parameter k1__k1__constant_value is close to the specified upper bound 100.0. Increasing the bound and calling fit again may find a better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1, 1)
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1, 1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kernels.py:452: ConvergenceWarning: The optimal value found for dimension 0 of parameter k1__k1__constant_value is close to the specified upper bound 100.0. Increasing the bound and calling fit again may find a better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1, 1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kernels.py:452: ConvergenceWarning: The optimal value found for dimension 0 of parameter k1__k1__constant_value is close to the specified upper bound 100.0. Increasing the bound and calling fit again may find a better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
```

```
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
```

```
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
```

```
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
```

```
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
```

```
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
```

```
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
```

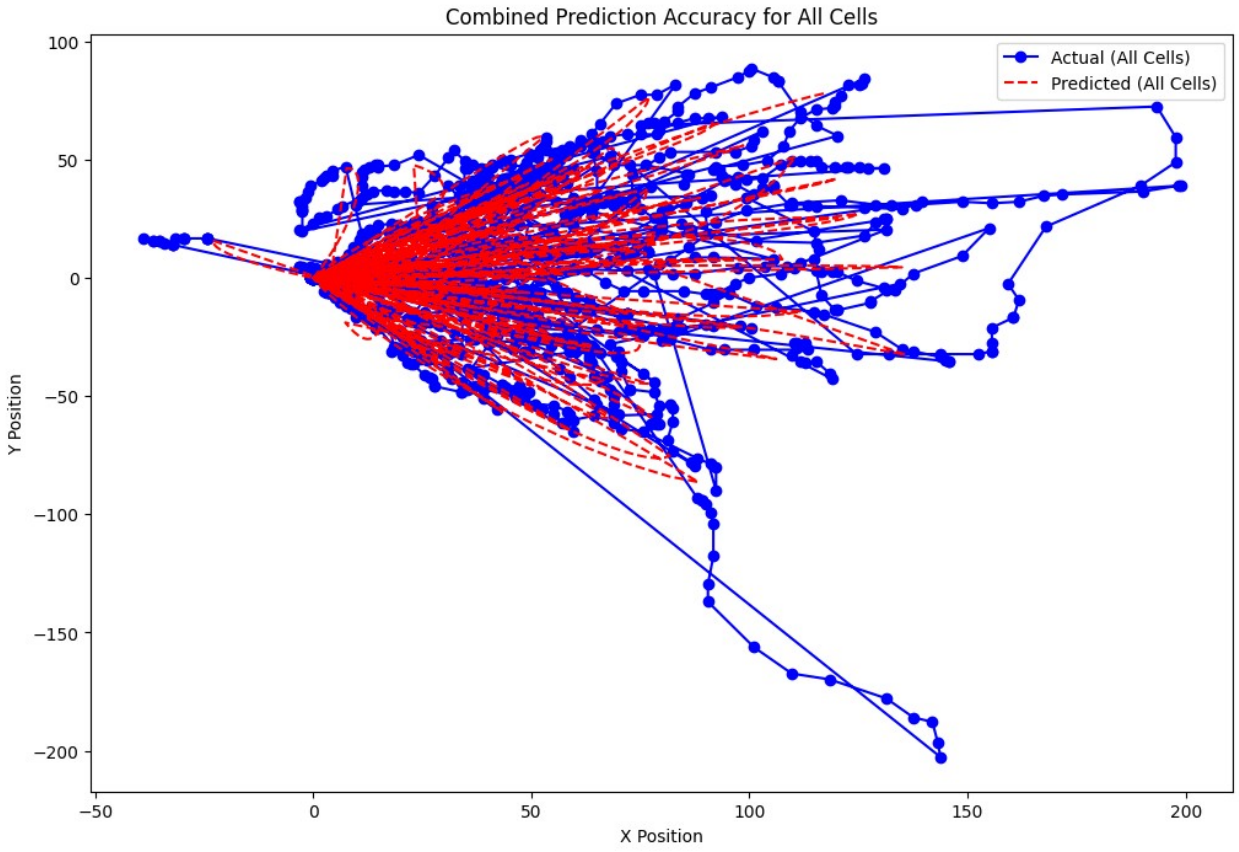
```
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
```

```
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
```

```
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1, 
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1, 
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1, 
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1, 
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1, 
```

```
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kerne
ls.py:452: ConvergenceWarning: The optimal value found for dimension 0
of parameter k1__k1__constant_value is close to the specified upper
bound 100.0. Increasing the bound and calling fit again may find a
better value.
  warnings.warn(
<ipython-input-46-5648e933e3ce>:51: DeprecationWarning: Conversion of
an array with ndim > 0 to a scalar is deprecated, and will error in
future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
  X_future = np.arange(X_train[-1] + 1, X_train[-1] + 18).reshape(-1,
1)
```




```
Overall Mean Squared Error: 1258.5072697610574
Overall R-squared: -7.242576702949513e+28
```

Inferences

We can see that there is a very high error of three magnitudes. We can also see that the GP fails to learn the general trend because it is being training for each cell. This could also be caused by a poor choice of kernel. A more descriptive kernel would learn better the pattern of the cells. A way to increase the accuracy of the code is by having more sample of cells, which can be done by using simulated data and to figure out a better kernel to model the localization of cells.

References

Sargent, B., Jafari, M., Marquez, G. et al. A machine learning based model accurately predicts cellular response to electric fields in multiple cell types. Sci Rep 12, 9912 (2022). https://doi.org/10.1038/s41598-022-13925-4

Arocena, M., Rajnicek, A. M., & Collinson, J. M. (2017). Requirement of Pax6 for the integration of guidance cues in cell migration. Royal Society Open Science, 4(10), 170625. https://doi.org/10.1098/rsos.170625