Machine Learning

Predicting the 3-Body Problem Dynamics

ANDRÉ GASPAR 59859
DINIS SILVESTRE 58763
RAFAEL LOPES 57652

Index

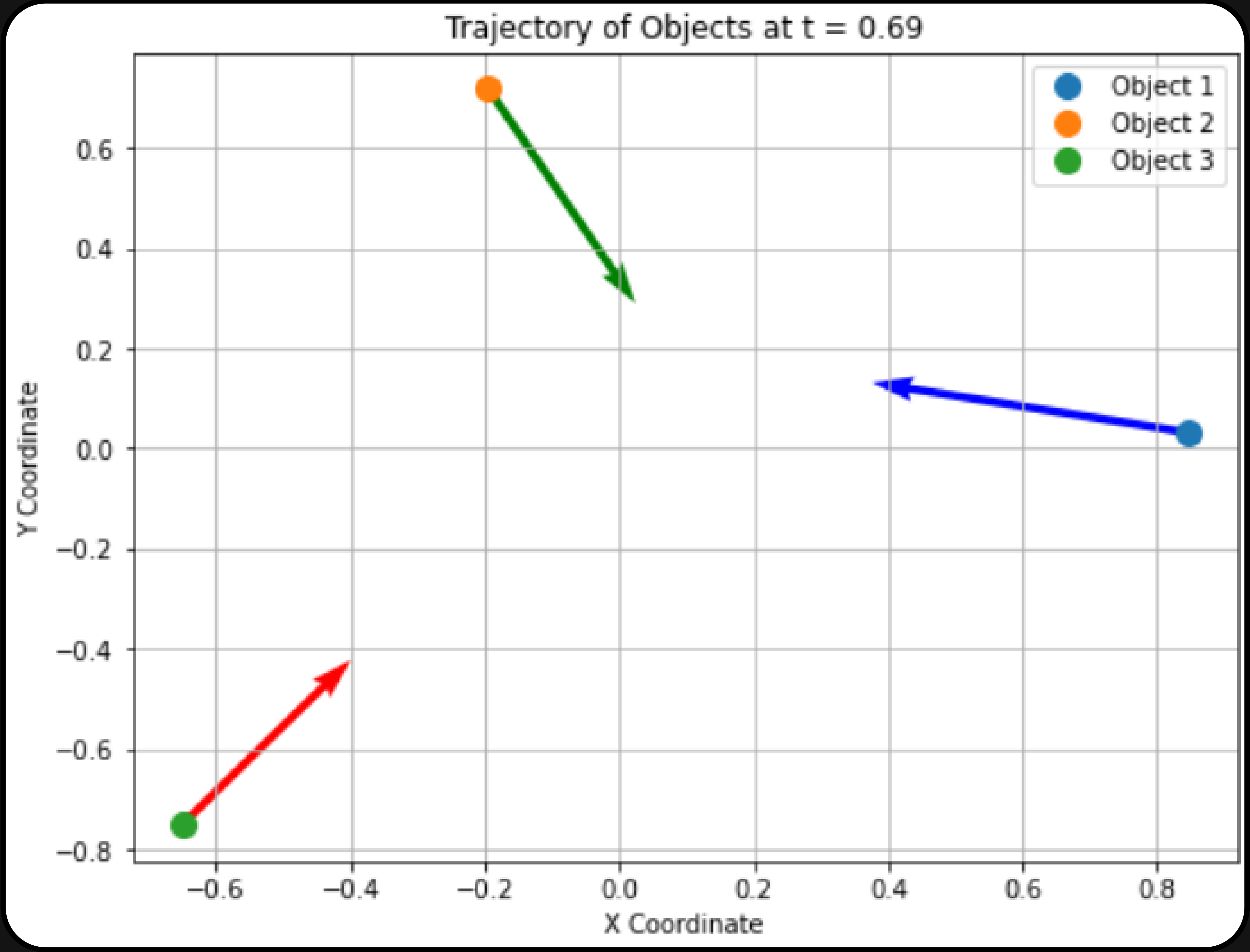
Overview

The 3-Body Problem presents a significant challenge in forecasting the complex interactions of celestial bodies without the use of numerical equations. The goal of this project is to use machine learning techniques to predict the movements of three bodies in a 2D plane.

We developed a solution that covers several decisions regarding model architecture and feature engineering which affected the prediction and evaluation of the model. We will analyse and examine data by displaying the expected positions for each body at a specific Id.

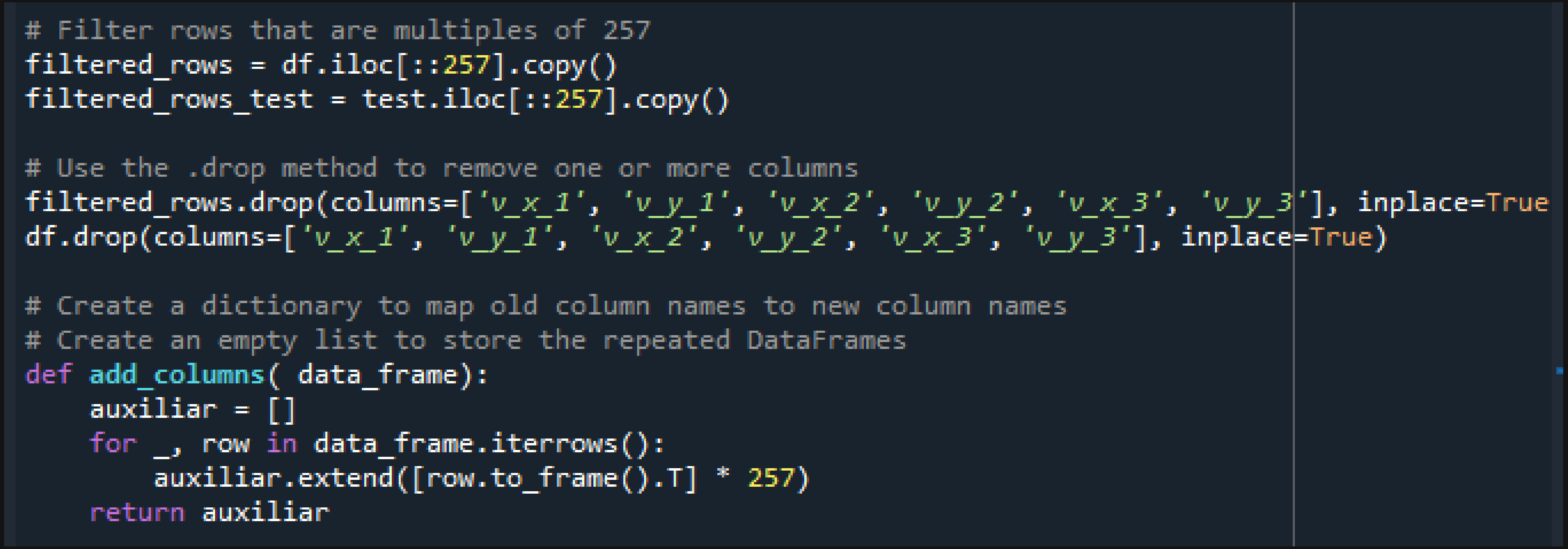
Data Exploration

The generated plot displays the objects' coordinates as points and the related velocities as vectors. This graphic helps to understand the dynamics and movements of celestial objects by illustrating their positions and velocities at a random time t.



Model Architecture

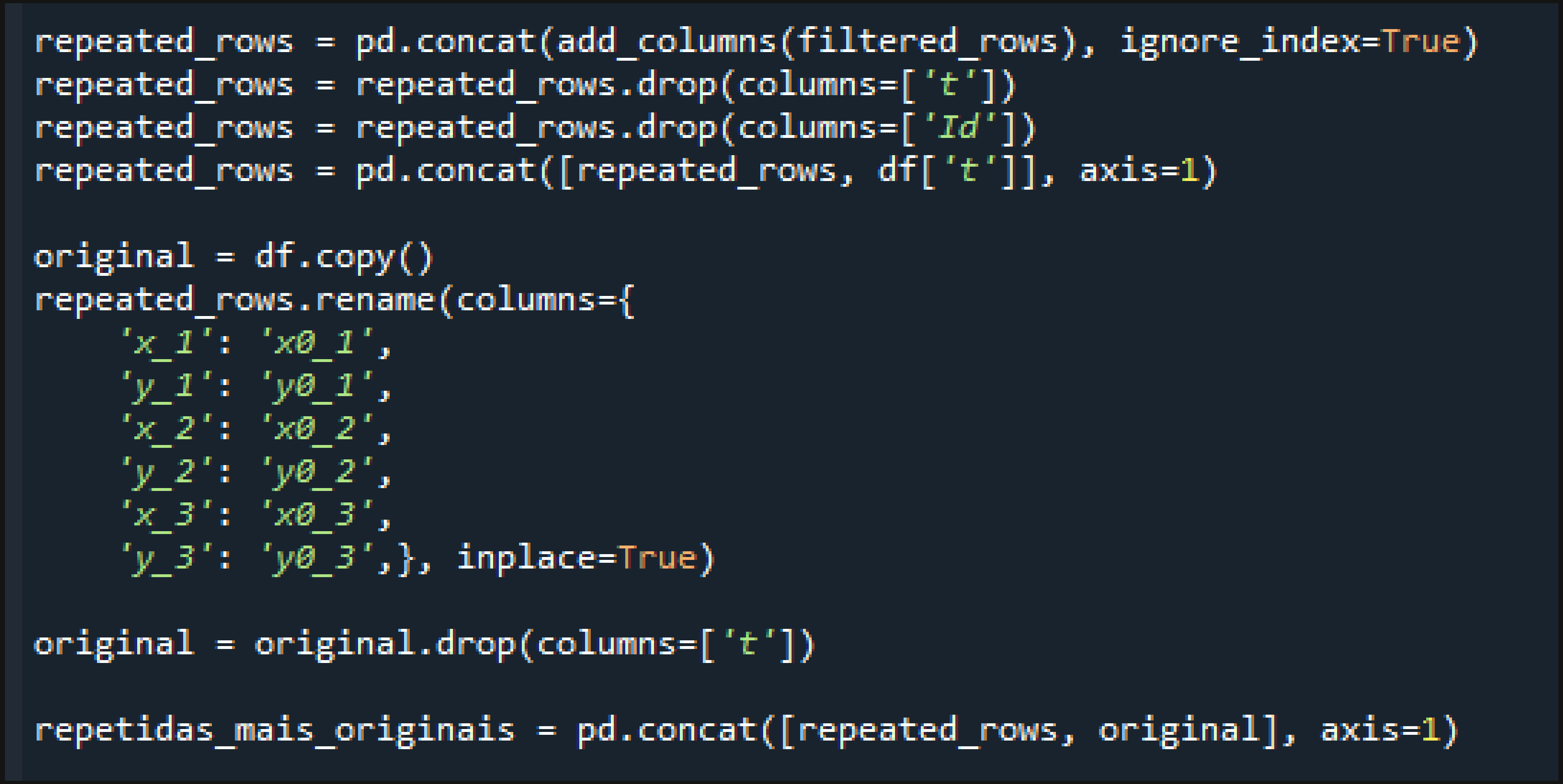
The code first imports data from CSV files and filters the data to filter rows which are multiples of 257, so when t=0. For performance, we decided to remove irrelevant columns. We created a function that duplicates rows to supplement the dataset, guaranteeing that there is enough data for analysis. The code simplifies data preparation and prepares the environment for model training and evaluation.



```python
# Filter rows that are multiples of 257
filtered_rows = df.iloc[::257].copy()
filtered_rows_test = test.iloc[::257].copy()

# Use the .drop method to remove one or more columns
filtered_rows.drop(columns=['v_x_1', 'v_y_1', 'v_x_2', 'v_y_2', 'v_x_3', 'v_y_3'], inplace=True
df.drop(columns=['v_x_1', 'v_y_1', 'v_x_2', 'v_y_2', 'v_x_3', 'v_y_3'], inplace=True)

# Create a dictionary to map old column names to new column names
# Create an empty list to store the repeated DataFrames
def add_columns( data_frame):
    auxiliar = []
    for _, row in data_frame.iterrows():
        auxiliar.extend([row.to_frame().T] * 257)
    return auxiliar
```

Model Architecture

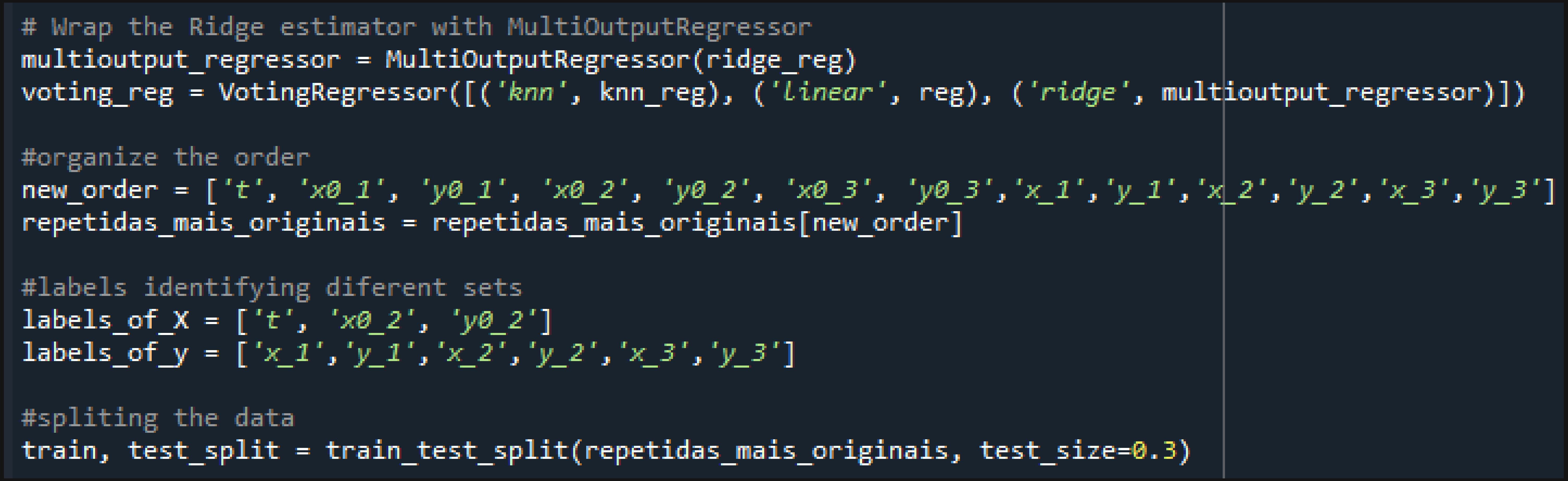
We aggregated the columns from ‘filtered_rows’, removed unnecessary columns, and appended the time to the ‘repeated_rows’. To ensure data reliability, we renamed the columns of the ‘repeated_rows’ DataFrame. Finally, the approach aggregates the duplicated and repeated data to form an aggregated dataset that includes both the repeated and original data, allowing for a thorough examination.



```
repeated_rows = pd.concat(add_columns(filtered_rows), ignore_index=True)
repeated_rows = repeated_rows.drop(columns=['t'])
repeated_rows = repeated_rows.drop(columns=['Id'])
repeated_rows = pd.concat([repeated_rows, df['t']], axis=1)

original = df.copy()
repeated_rows.rename(columns={
    'x_1': 'x0_1',
    'y_1': 'y0_1',
    'x_2': 'x0_2',
    'y_2': 'y0_2',
    'x_3': 'x0_3',
    'y_3': 'y0_3',}, inplace=True)

original = original.drop(columns=['t'])

repetidas_mais_originais = pd.concat([repeated_rows, original], axis=1)
```

Feature Engineering

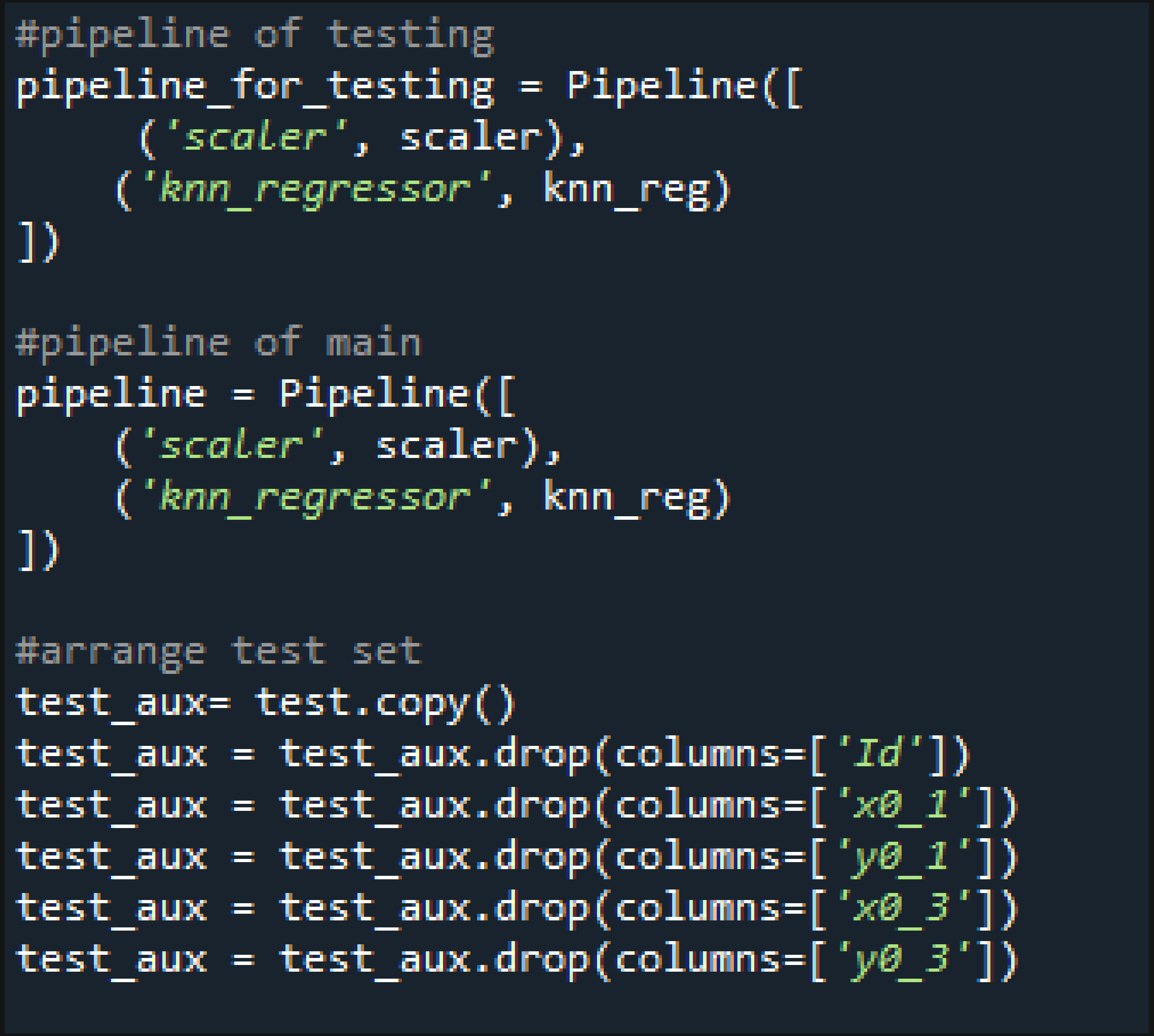
We created regression models to experiment with different strategies and approaches to data prediction, such as K-Nearest Neighbours Regression, Linear and Ridge Regressions. The StandardScaler is used to standardise data, while the Ridge estimator is wrapped in MultiOutputRegressor. We believe that the VotingRegressor improves the overall performance.



```
# Wrap the Ridge estimator with MultiOutputRegressor
multioutput_regressor = MultiOutputRegressor(ridge_reg)
voting_reg = VotingRegressor([('knn', knn_reg), ('linear', reg), ('ridge', multioutput_regressor)])

#organize the order
new_order = ['t', 'x0_1', 'y0_1', 'x0_2', 'y0_2', 'x0_3', 'y0_3','x_1','y_1','x_2','y_2','x_3','y_3']
repetidas_mais_originais = repetidas_mais_originais[new_order]

#labels identifying diferent sets
labels_of_X = ['t', 'x0_2', 'y0_2']
labels_of_y = ['x_1','y_1','x_2','y_2','x_3','y_3']

#spliting the data
train, test_split = train_test_split(repetidas_mais_originais, test_size=0.3)
```

Feature Engineering

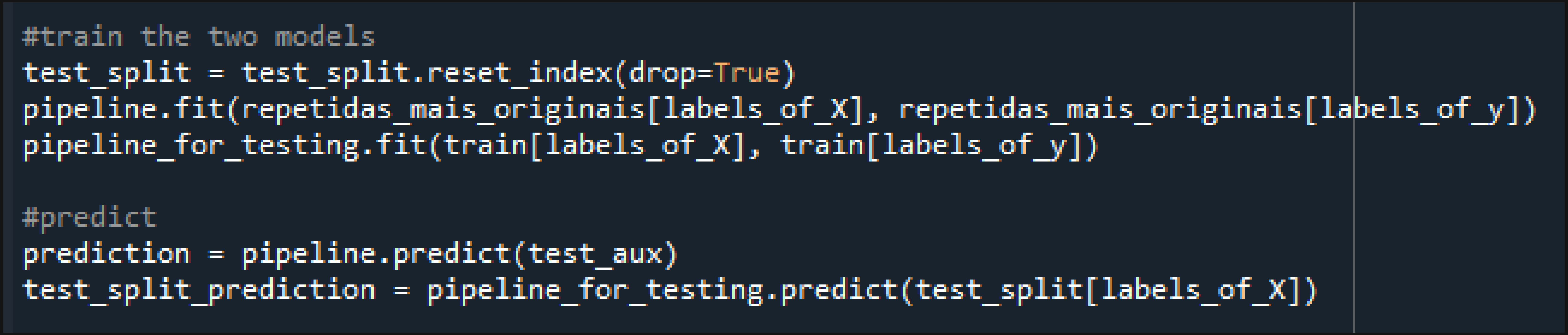
We defined two pipelines, one for testing and the other as the primary pipeline. The test set is prepared by dropping certain columns to ensure consistency.



```
#pipeline of testing
pipeline_for_testing = Pipeline([
     ('scaler', scaler),
    ('knn_regressor', knn_reg)
])

#pipeline of main
pipeline = Pipeline([
    ('scaler', scaler),
    ('knn_regressor', knn_reg)
])

#arrange test set
test_aux= test.copy()
test_aux = test_aux.drop(columns=['Id'])
test_aux = test_aux.drop(columns=['x0_1'])
test_aux = test_aux.drop(columns=['y0_1'])
test_aux = test_aux.drop(columns=['x0_3'])
test_aux = test_aux.drop(columns=['y0_3'])
```

Model Training

The training data is used to train the two models. By resetting the index, the test split data is prepared for the training and testing pipelines. To train the models on the individual datasets, the fit approach is used. The trained models are then used to forecast outputs for the test and test split data.

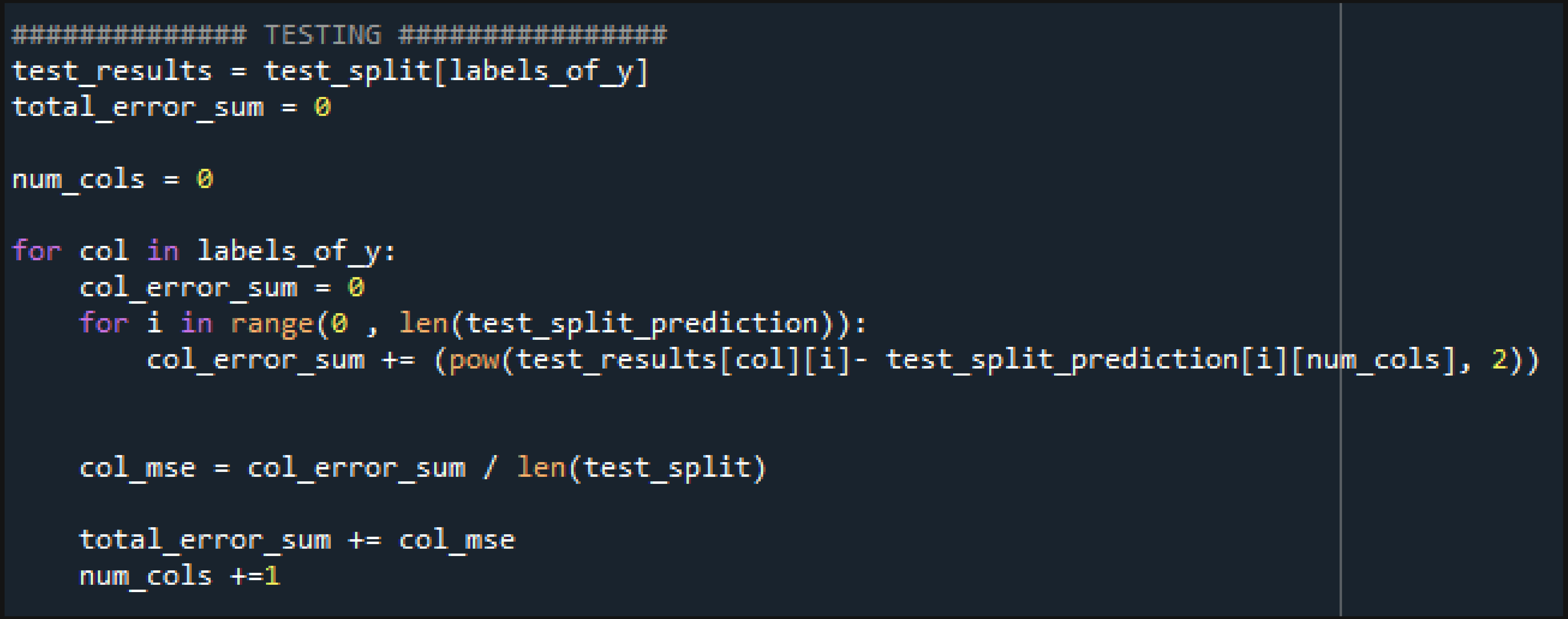


```
#train the two models
test_split = test_split.reset_index(drop=True)
pipeline.fit(repetidas_mais_originais[labels_of_X], repetidas_mais_originais[labels_of_y])
pipeline_for_testing.fit(train[labels_of_X], train[labels_of_y])

#predict
prediction = pipeline.predict(test_aux)
test_split_prediction = pipeline_for_testing.predict(test_split[labels_of_X])
```

Evaluation metrics

We intended to compute the Mean Squared Error (MSE) for the predicted values and the actual test results with this script. It iterates through the test labels, calculating the sum of squared errors for each input.

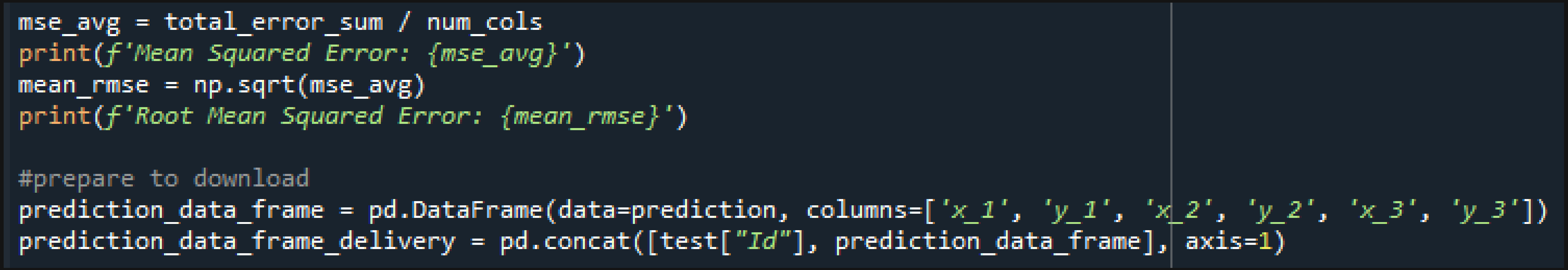
- test_results: stores the actual test results associated with the labels of interest.
- total_error_sum: sum of MSE values across all the columns in the test results.
- num_cols: used to obtain the average MSE.



```python
############### TESTING #################
test_results = test_split[labels_of_y]
total_error_sum = 0

num_cols = 0

for col in labels_of_y:
    col_error_sum = 0
    for i in range(0 , len(test_split_prediction)):
        col_error_sum += (pow(test_results[col][i]- test_split_prediction[i][num_cols], 2))


    col_mse = col_error_sum / len(test_split)

    total_error_sum += col_mse
    num_cols +=1
```

Evaluation metrics

The mean squared error was computed by dividing the total error sum by the number of columns. The MSE value indicates that the average squared difference between actual and predicted values across all the celestial bodies' positions is around 0.1534. The RMSE value of about 0.3917 implies that the average difference between predicted and actual celestial body locations is approximately 0.3917.

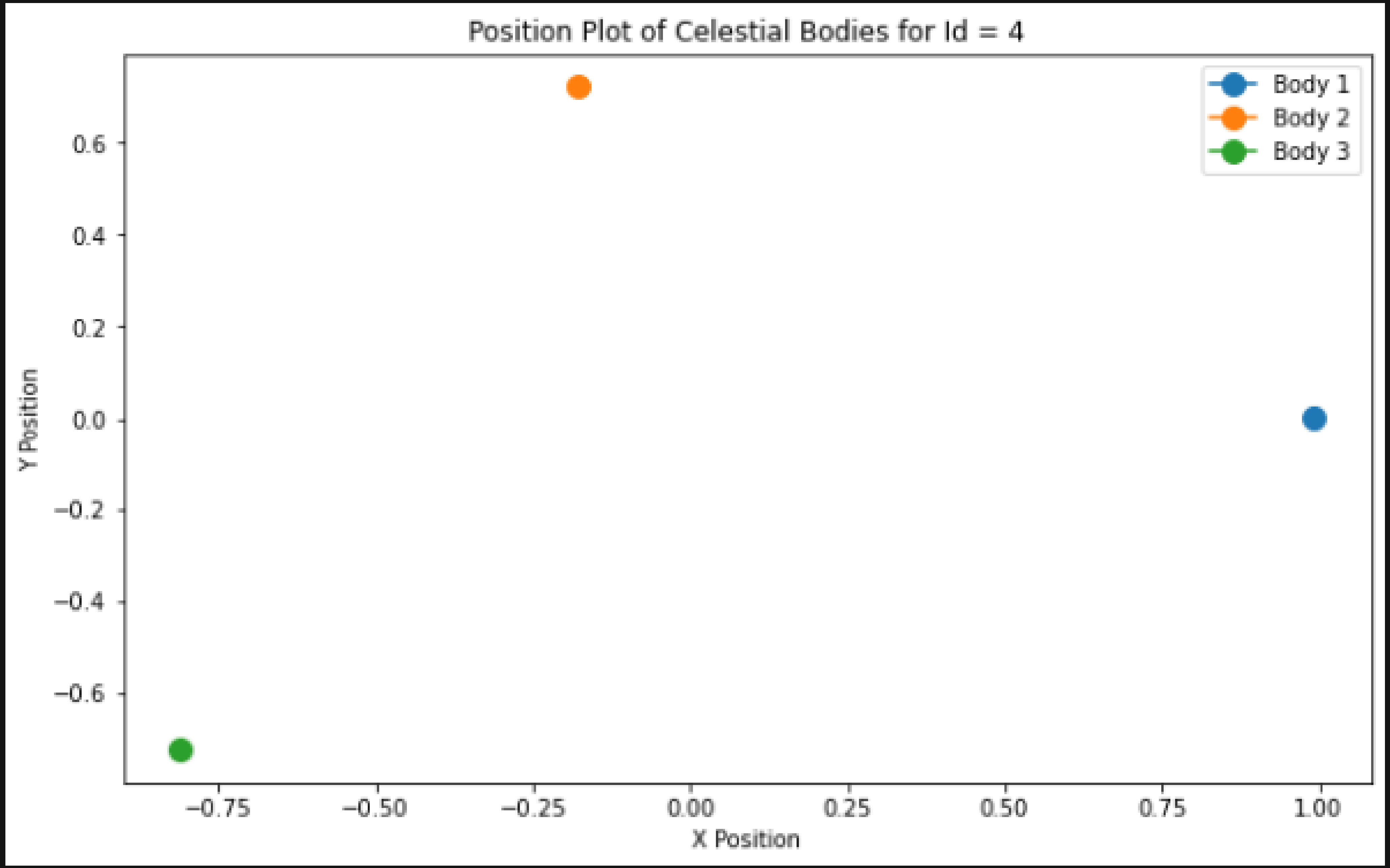


```
mse_avg = total_error_sum / num_cols
print(f'Mean Squared Error: {mse_avg}')
mean_rmse = np.sqrt(mse_avg)
print(f'Root Mean Squared Error: {mean_rmse}')

#prepare to download
prediction_data_frame = pd.DataFrame(data=prediction, columns=['x_1', 'y_1', 'x_2', 'y_2', 'x_3', 'y_3'])
prediction_data_frame_delivery = pd.concat([test["Id"], prediction_data_frame], axis=1)
```

Root Mean Squared Error	0.3916835513890955
Mean Squared Error	0.1534160044287742

Results and Analysis

A two-dimensional trajectory graphic depicting the projected positions of three heavenly entities with ID 4. This analysis tries to graphically display the celestial bodies' simulated movements based on the prediction model's output, so leading to a better understanding of their motion patterns.



Challenges and Limitations

Despite the robustness of the approach used, difficulties were faced in dealing with outliers and ensuring consistent data quality. The model's shortcomings were mostly linked with the intrinsic difficulties of 3-Body dynamics and the possibility of unexpected collisions between the bodies.

Finally, the Multi-Output Ridge Regression model demonstrated encouraging results in predicting the complicated motions of three bodies in a 2D plane. This effort advances our understanding of the 3-Body Problem and highlights the effectiveness of machine learning approaches in dealing with complex celestial dynamics.

CONCLUSION