dbagan_TFM

June 14, 2022

Daniel Bagan Martínez

0.0.1 Librerías

```python
[1]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     from keras.layers.core import Dense, Activation, Dropout, Flatten
     from keras.layers import LSTM, BatchNormalization
     from keras.models import Sequential
     from tensorflow.keras.optimizers import Adam
     from sklearn.preprocessing import StandardScaler, MinMaxScaler
     from sklearn.utils import shuffle
     import haversine as hs
     from haversine import Unit
     import joblib
     import math
     import geopy
     import geopy.distance
     import folium
     from shapely import geometry, ops
     import time
```

0.0.2 EDA

```python
[2]: df = pd.read_csv("positions_1000.csv")
```

```python
[530]: df = shuffle(df)
```

```python
[531]: df.head()
```

```
[531]:          time  vehicle_id  victim_id   latitude  longitude      speed  \
       2287287  11561        9439         -1  41.391807   2.165370   0.062439
       2793139  14103       11848         -1  41.390956   2.165465   0.000000
       1696045   8589        7100         -1  41.391666   2.165597   0.000000
       3478531  17547       14929         -1  41.388466   2.163740  10.770545
       2083999  10538        9005         -1  41.390053   2.166574   3.943876
```

```
          heading  collision
2287287  317.547946         0
2793139   45.738504         0
1696045  317.458691         0
3478531  136.050655         0
2083999  225.260525         0
```

[532]: `df.shape`

[532]: (3966377, 8)

[533]: `df.describe()`

[533]:
```
                time     vehicle_id      victim_id      latitude     longitude  \
count   3.966377e+06   3.966377e+06   3.966377e+06   3.966377e+06   3.966377e+06
mean    1.003159e+04   8.435308e+03   1.286898e+00   4.139137e+01   2.164493e+00
std     5.755679e+03   4.812723e+03   1.607677e+02   1.097701e-03   1.362169e-03
min     0.000000e+00   3.000000e+00  -1.000000e+00   4.138700e+01   2.157329e+00
25%     5.050000e+03   4.358000e+03  -1.000000e+00   4.139077e+01   2.163715e+00
50%     1.003200e+04   8.346000e+03  -1.000000e+00   4.139136e+01   2.164423e+00
75%     1.501600e+04   1.250500e+04  -1.000000e+00   4.139199e+01   2.165491e+00
max     1.999900e+04   1.710200e+04   1.700300e+04   4.139418e+01   2.167546e+00

               speed        heading      collision
count   3.966377e+06   3.966377e+06   3.966377e+06
mean    2.307072e+00   2.030676e+02   2.657337e-04
std     3.478560e+00   1.001039e+02   1.629918e-02
min     0.000000e+00   4.126121e-04   0.000000e+00
25%     0.000000e+00   1.357220e+02   0.000000e+00
50%     0.000000e+00   2.254755e+02   0.000000e+00
75%     4.954288e+00   3.174587e+02   0.000000e+00
max     1.622598e+01   3.599969e+02   1.000000e+00
```

[534]:
```python
cols = df.columns
fig, ax = plt.subplots(nrows=2, ncols=len(cols)//2, figsize=(25,12))
for i in range (len(cols)):
    ax[i//4][i%4].hist(df[cols[i]])
    ax[i//4][i%4].set_title(cols[i])
```
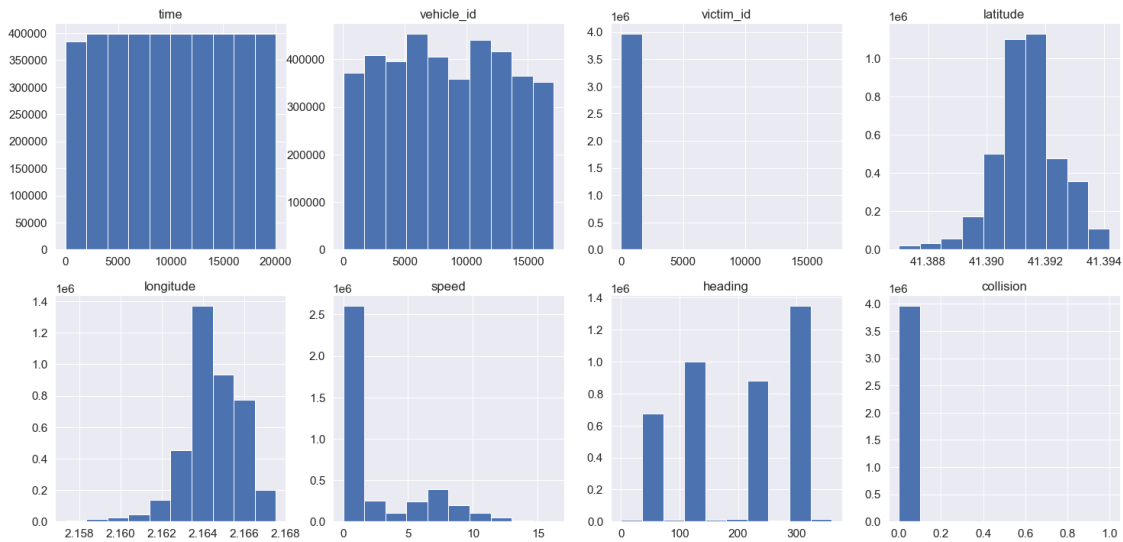
```
[535]: total_collisions = len(df['collision'].loc[df['collision'] != 0])
       print(f"Total collisions: {total_collisions}")
       print("Total different vehicles: {0}".format(len(df["vehicle_id"].unique())))
```

```
Total collisions: 1054
Total different vehicles: 15756
```

```
[536]: plt.scatter(x=df['longitude'], y=df['latitude'])
       plt.show()
```
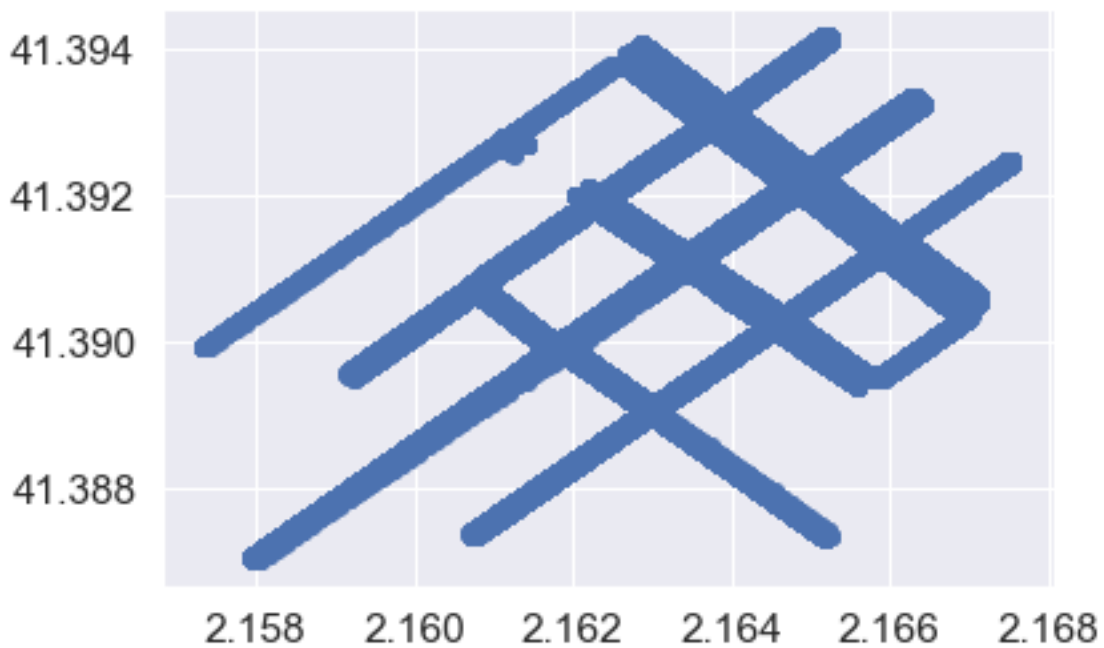
1 Modelos de regresión

1.0.1 Load data

```python
[537]: def format_dataset_regresion(df, n):

          df = df.sort_values(['vehicle_id', 'time'])
          df = df.reset_index(drop=True)

          X, y = [], []
          temp_X = []
          temp_i = 0
          temp_vehicle = df.loc[0, 'vehicle_id']

          for i in range(1, df.shape[0]):

              row = df.loc[i]

              if row["vehicle_id"] != temp_vehicle:
                  temp_vehicle = row["vehicle_id"]
                  temp_i = 0
                  temp_X = []
                  temp_X.append([row["latitude"], row["longitude"]])

              elif temp_i < n:
                  temp_X.append([row["latitude"], row["longitude"]])

              else:
                  X.append(temp_X)
                  y.append([row["latitude"], row["longitude"]])
                  temp_X = []
                  temp_i = 0
                  temp_vehicle = -1

              temp_i += 1

          X = np.array(X, dtype='float64')
          y = np.array(y, dtype='float64')

          return X, y

      #X_set, y_set = format_dataset_regresion(df, 26)
      #np.save(file="./X_nparray_26.npy", arr=X_set)
      #np.save(file="./y_nparray_26.npy", arr=y_set)
```

```
[538]: X_set_full = np.load("./X_nparray_26.npy")
       y_set_full = np.load("./y_nparray_26.npy")
```

```
[539]: X_set_full, y_set_full = shuffle(X_set_full, y_set_full)
       X_set = X_set_full[0:1000]
       y_set = y_set_full[0:1000]
```

```
[540]: y_set_full
```

```
[540]: array([[41.39142019,  2.16394654],
               [41.39182239,  2.16453329],
               [41.39322376,  2.16358408],
               ...,
               [41.3923991 ,  2.16453908],
               [41.39174714,  2.16267759],
               [41.38927802,  2.16333392]])
```

1.0.2 Functions

```
[541]: def get_results(model):
           res = []
           for i in range(len(X_set)):
               X = X_set[i][:-1]
               y = X_set[i][1:]
               model.fit(X, y)
               pred = model.predict([y[-1]])
               res.append(pred[0])
           return res
```

```
[542]: def get_avg_error(pred):
           total_distance = 0
           for i in range(len(pred)):
               input = X_set[i][-1]
               total_distance += np.sqrt(np.sum(np.square(pred[i][0]-input[0]))+np.
        ↪square(pred[i][1]-input[1]))
           avg_distance = total_distance/len(pred)
           return avg_distance
```

```
[543]: def get_avg_distance(pred):
           total_distance = 0
           for i in range(len(pred)):
               input = X_set[i][-1]
               total_distance += hs.haversine(( pred[i][0] , pred[i][1] ),( input[0],
        ↪input[1] ), unit=Unit.METERS)
           avg_distance = total_distance/len(pred)
           return avg_distance
```

```
[544]: def get_avg_distance_10_predictions(pred, real):
           avg_distances = []
           for j in range(10):
               n_pred_distance = 0
               for i in range(len(pred)):
                   p = pred[i][j]
                   r = real[i][j]
                   n_pred_distance += hs.haversine(( p[0] , p[1] ),( r[0] , r[1] ),
         ↪unit=Unit.METERS)
               avg_distances.append((n_pred_distance)/len(pred))
           return avg_distances
```

```
[545]: def call_get_next_10_predictions(dataset, model):
           preds = []
           reals = []
           for data in dataset:
               pred, real = get_next_10_predictions(data, model)
               preds.append(pred)
               reals.append(real)
           return preds, reals
```

```
[546]: def get_next_5_predictions(data, model):
           # Prediction 1
           X = data[:-7]
           y = data[1:-6]
           model.fit(X, y)
           pred_1 = model.predict([y[-1]])
           real_1 = data[-5]

           # Prediction 2
           X = np.concatenate((data[1:-7], pred_1))
           y = np.concatenate((data[2:-7], pred_1, [real_1]))
           model.fit(X, y)
           pred_2 = model.predict([y[-1]])
           real_2 = data[-4]

           # Prediction 3
           X = np.concatenate((data[2:-7], pred_1, pred_2))
           y = np.concatenate((data[3:-7], pred_1, pred_2, [real_2]))
           model.fit(X, y)
           pred_3 = model.predict([y[-1]])
           real_3 = data[-3]

           # Prediction 4
           X = np.concatenate((data[3:-7], pred_1, pred_2, pred_3))
           y = np.concatenate((data[4:-7], pred_1, pred_2, pred_3, [real_3]))
           model.fit(X, y)
```

```
        pred_4 = model.predict([y[-1]])
        real_4 = data[-2]

        # Prediction 5
        X = np.concatenate((data[4:-7], pred_1, pred_2, pred_3, pred_4))
        y = np.concatenate((data[5:-7], pred_1, pred_2, pred_3, pred_4, [real_4]))
        model.fit(X, y)
        pred_5 = model.predict([y[-1]])
        real_5 = data[-1]

        predictions = [l.tolist() for l in␣
 ↪[pred_1[0],pred_2[0],pred_3[0],pred_4[0],pred_5[0]]]
        real_values = [l.tolist() for l in [real_1,real_2,real_3,real_4,real_5]]

        return predictions, real_values
```

[547]:
```
def call_get_next_10_predictions_res(dataset, model):
    preds = []
    reals = []
    for data in dataset:
        pred, real = get_next_10_predictions_res(data, model)
        preds.append(pred)
        reals.append(real)
    return preds, reals
```

[548]:
```
def get_next_10_predictions(data, model):
    # Prediction 1
    X = data[:-12]
    y = data[1:-11]
    model.fit(X, y)
    pred_1 = model.predict([y[-1]])
    real_1 = data[-10]

    # Prediction 2
    X = np.concatenate((data[1:-12], pred_1))
    y = np.concatenate((data[2:-12], pred_1, [real_1]))
    model.fit(X, y)
    pred_2 = model.predict([y[-1]])
    real_2 = data[-9]

    # Prediction 3
    X = np.concatenate((data[2:-12], pred_1, pred_2))
    y = np.concatenate((data[3:-12], pred_1, pred_2, [real_2]))
    model.fit(X, y)
    pred_3 = model.predict([y[-1]])
    real_3 = data[-8]
```

```python
# Prediction 4
X = np.concatenate((data[3:-12], pred_1, pred_2, pred_3))
y = np.concatenate((data[4:-12], pred_1, pred_2, pred_3, [real_3]))
model.fit(X, y)
pred_4 = model.predict([y[-1]])
real_4 = data[-7]

# Prediction 5
X = np.concatenate((data[4:-12], pred_1, pred_2, pred_3, pred_4))
y = np.concatenate((data[5:-12], pred_1, pred_2, pred_3, pred_4, [real_4]))
model.fit(X, y)
pred_5 = model.predict([y[-1]])
real_5 = data[-6]

# Prediction 6
X = np.concatenate((data[5:-12], pred_1, pred_2, pred_3, pred_4, pred_5))
y = np.concatenate((data[6:-12], pred_1, pred_2, pred_3, pred_4, pred_5,␣
↪[real_5]))
model.fit(X, y)
pred_6 = model.predict([y[-1]])
real_6 = data[-5]

# Prediction 7
X = np.concatenate((data[6:-12], pred_1, pred_2, pred_3, pred_4, pred_5,␣
↪pred_6))
y = np.concatenate((data[7:-12], pred_1, pred_2, pred_3, pred_4, pred_5,␣
↪pred_6, [real_6]))
model.fit(X, y)
pred_7 = model.predict([y[-1]])
real_7 = data[-4]

# Prediction 8
X = np.concatenate((data[7:-12], pred_1, pred_2, pred_3, pred_4, pred_5,␣
↪pred_6, pred_7))
y = np.concatenate((data[8:-12], pred_1, pred_2, pred_3, pred_4, pred_5,␣
↪pred_6, pred_7, [real_7]))
model.fit(X, y)
pred_8 = model.predict([y[-1]])
real_8 = data[-3]

# Prediction 9
X = np.concatenate((data[8:-12], pred_1, pred_2, pred_3, pred_4, pred_5,␣
↪pred_6, pred_7, pred_8))
y = np.concatenate((data[9:-12], pred_1, pred_2, pred_3, pred_4, pred_5,␣
↪pred_6, pred_7, pred_8, [real_8]))
model.fit(X, y)
```

```
    pred_9 = model.predict([y[-1]])
    real_9 = data[-2]

    # Prediction 10
    X = np.concatenate((data[9:-12], pred_1, pred_2, pred_3, pred_4, pred_5,␣
 ↪pred_6, pred_7, pred_8, pred_9))
    y = np.concatenate((data[10:-12], pred_1, pred_2, pred_3, pred_4, pred_5,␣
 ↪pred_6, pred_7, pred_8, pred_9, [real_9]))
    model.fit(X, y)
    pred_10 = model.predict([y[-1]])
    real_10 = data[-1]

    predictions = [l.tolist() for l in␣
 ↪[pred_1[0],pred_2[0],pred_3[0],pred_4[0],pred_5[0],pred_6[0],pred_7[0],pred_8[0],pred_9[0],
    real_values = [l.tolist() for l in␣
 ↪[real_1,real_2,real_3,real_4,real_5,real_6,real_7,real_8,real_9,real_10]]

    return predictions, real_values
```

```
[549]: def plot_distance_errors(avg_distances):
    plt.plot(avg_distances)
    plt.title('Error en la predicción de posiciones')
    plt.ylabel('Distancia media')
    plt.xlabel('n predicción')
    plt.legend(['train', 'test'], loc='lower right')
    plt.show()
```

1.0.3 Test modelos de regresión

```
[550]: # Linear regressor
    from sklearn.linear_model import LinearRegression
    model = LinearRegression()
    start = time.time()
    pred = get_results(model)
    avg_distance = get_avg_distance(pred)
    print(f"Average distance {avg_distance}")
    end = time.time()
    print(f"Time: {end - start}s")
```

```
Average distance 267.83609132206743
Time: 1.8150031566619873s
```

```
[551]: # Ridge
    from sklearn.linear_model import Ridge
    model = Ridge()
    start = time.time()
    pred = get_results(model)
```

```python
avg_distance = get_avg_distance(pred)
print(f"Average distance {avg_distance}")
end = time.time()
print(f"Time: {end - start}s")
```

Average distance 24.12832157347379
Time: 0.9639983177185059s

[552]:
```python
# Lasso
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import Lasso
model = Lasso()
start = time.time()
pred = get_results(model)
avg_distance = get_avg_error(pred)
print(f"Average error {avg_distance}")
end = time.time()
print(f"Time: {end - start}s")
```

Average error 0.0002535602550621559
Time: 1.2969977855682373s

[553]:
```python
# Lars
from sklearn.linear_model import Lars
model = Lars(n_nonzero_coefs=500, eps=0.5)
start = time.time()
pred = get_results(model)
avg_distance = get_avg_distance(pred)
print(f"Average distance {avg_distance}")
end = time.time()
print(f"Time: {end - start}s")
```

Average distance 2.8608199417631064
Time: 1.773000955581665s

[554]:
```python
# LassoLars
from sklearn.linear_model import LassoLars
model = LassoLars()
start = time.time()
pred = get_results(model)
avg_distance = get_avg_distance(pred)
print(f"Average distance {avg_distance}")
end = time.time()
print(f"Time: {end - start}s")
```

Average distance 24.128459501029948
Time: 1.256000280380249s

```
[555]:  # MultiTaskElasticNet
        from sklearn.linear_model import MultiTaskElasticNet
        model = MultiTaskElasticNet()
        start = time.time()
        pred = get_results(model)
        avg_distance = get_avg_distance(pred)
        print(f"Average distance {avg_distance}")
        end = time.time()
        print(f"Time: {end - start}s")
```

Average distance 24.128459500669365
Time: 0.9019980430603027s

```
[556]:  # RANSACRegressor
        from sklearn.linear_model import RANSACRegressor
        model = RANSACRegressor()
        start = time.time()
        pred = get_results(model)
        avg_distance = get_avg_distance(pred)
        print(f"Average distance {avg_distance}")
        end = time.time()
        print(f"Time: {end - start}s")
```

Average distance 267.83609132206743
Time: 5.608991384506226s

```
[557]:  # DecisionTreeRegressor
        from sklearn.tree import DecisionTreeRegressor
        model = DecisionTreeRegressor()
        start = time.time()
        pred = get_results(model)
        avg_distance = get_avg_distance(pred)
        print(f"Average distance {avg_distance}")
        end = time.time()
        print(f"Time: {end - start}s")
```

Average distance 0.20931291022926427
Time: 0.7409956455230713s

```
[558]:  # ExtraTreeRegressor
        from sklearn.tree import ExtraTreeRegressor
        model = ExtraTreeRegressor()
        start = time.time()
        pred = get_results(model)
        avg_distance = get_avg_distance(pred)
        print(f"Average distance {avg_distance}")
        end = time.time()
        print(f"Time: {end - start}s")
```

11

```
Average distance 0.1936346509873833
Time: 0.7239949703216553s
```

[559]:
```python
# MLPRegressor
from sklearn.neural_network import MLPRegressor
model = MLPRegressor(hidden_layer_sizes=100, activation='relu',
 ↪solver='lbfgs',alpha=0.001, learning_rate='adaptive', max_fun=10000)
start = time.time()
pred = get_results(model)
avg_distance = get_avg_distance(pred)
print(f"Average distance {avg_distance}")
end = time.time()
print(f"Time: {end - start}s")
```

```
Average distance 16.88185766073174
Time: 7.0050153732299805s
```

[560]:
```python
# KNeighborsRegressor
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=1)
start = time.time()
pred = get_results(model)
avg_distance = get_avg_distance(pred)
print(f"Average distance {avg_distance}")
end = time.time()
print(f"Time: {end - start}s")
```

```
Average distance 0.022122681250334748
Time: 1.300994634628296s
```

[561]:
```python
# RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=5)
start = time.time()
pred = get_results(model)
avg_distance = get_avg_distance(pred)
print(f"Average distance {avg_distance}")
end = time.time()
print(f"Time: {end - start}s")
```

```
Average distance 1.5348190331825768
Time: 11.36794400215149s
```

[562]:
```python
# BaggingRegressor
from sklearn.ensemble import BaggingRegressor
model = BaggingRegressor()
start = time.time()
pred = get_results(model)
avg_distance = get_avg_distance(pred)
```

```
print(f"Average distance {avg_distance}")
end = time.time()
print(f"Time: {end - start}s")
```

```
Average distance 1.4010788705830168
Time: 25.740262508392334s
```

[563]:
```
# GaussianProcessRegressor
from sklearn.gaussian_process import GaussianProcessRegressor
model = GaussianProcessRegressor()
start = time.time()
pred = get_results(model)
avg_distance = get_avg_distance(pred)
print(f"Average distance {avg_distance}")
end = time.time()
print(f"Time: {end - start}s")
```

```
Average distance 2.930892974294869
Time: 2.759164571762085s
```

[564]:
```
# DummyRegressor
from sklearn.dummy import DummyRegressor
model = DummyRegressor()
start = time.time()
pred = get_results(model)
avg_distance = get_avg_distance(pred)
print(f"Average distance {avg_distance}")
end = time.time()
print(f"Time: {end - start}s")
```

```
Average distance 24.128459501029948
Time: 0.17499732971191406s
```

1.0.4 Media de distancias en 10 predicciones

[565]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
pred, real = call_get_next_10_predictions(X_set, model)
avg_distances = get_avg_distance_10_predictions(pred, real)
plot_distance_errors(avg_distances)
for i in range(len(avg_distances)):
    print(f"Average distance prediction {i}: {avg_distances[i]}")
```

Error en la predicción de posiciones
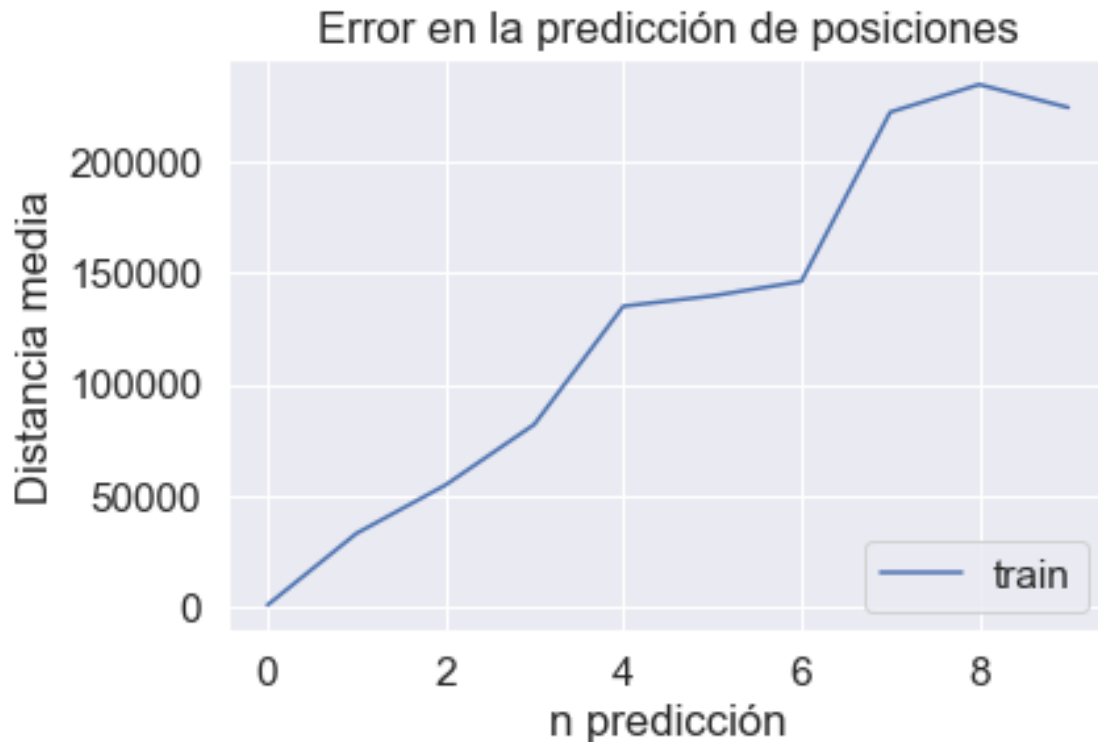
```
Average distance prediction 0: 746.963072935254
Average distance prediction 1: 32980.35856740886
Average distance prediction 2: 54730.10958604189
Average distance prediction 3: 82065.0060769803
Average distance prediction 4: 135219.9372630183
Average distance prediction 5: 139867.9908710348
Average distance prediction 6: 146365.60063533954
Average distance prediction 7: 222558.08287102802
Average distance prediction 8: 234986.66401311656
Average distance prediction 9: 224629.74598005554
```

[566]:
```python
# Ridge
from sklearn.linear_model import Ridge
model = Ridge()
pred, real = call_get_next_10_predictions(X_set, model)
avg_distances = get_avg_distance_10_predictions(pred, real)
plot_distance_errors(avg_distances)
for i in range(len(avg_distances)):
    print(f"Average distance prediction {i}: {avg_distances[i]}")
```
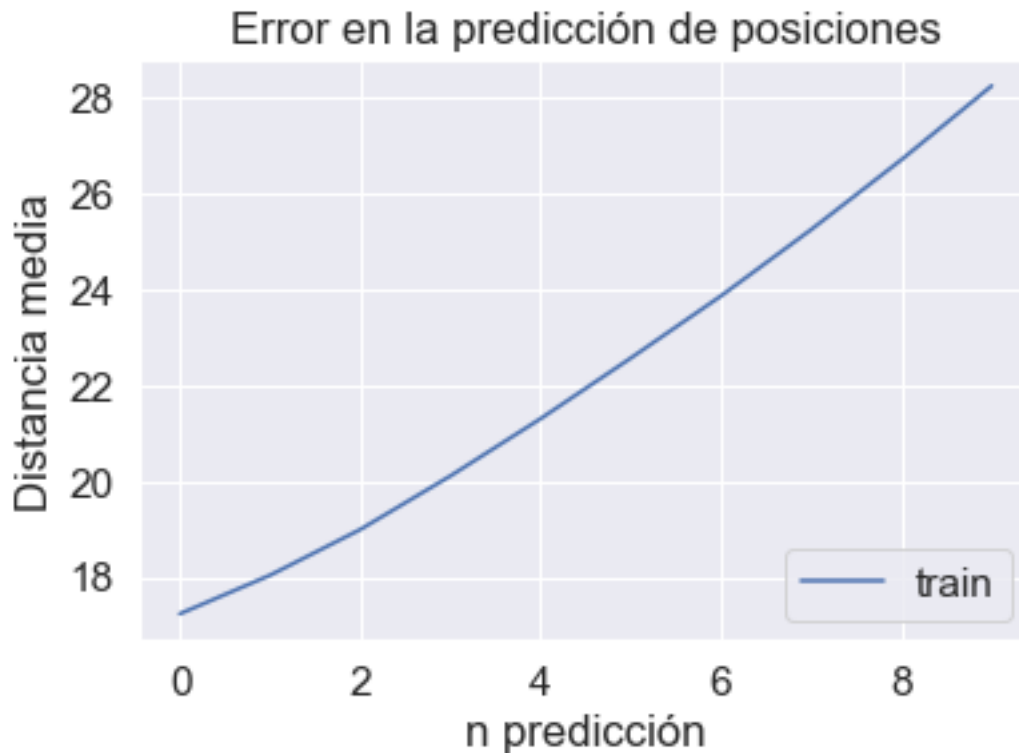
Error en la predicción de posiciones
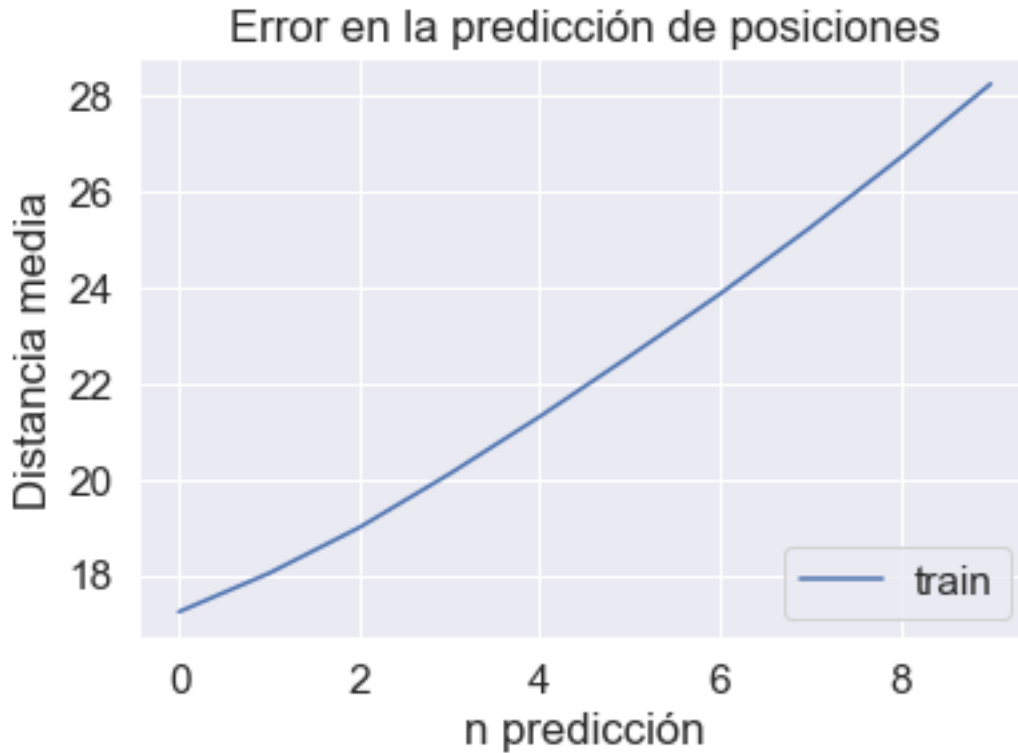

```
Average distance prediction 0: 17.22965741117776
Average distance prediction 1: 18.034240165239012
Average distance prediction 2: 18.986868303562794
Average distance prediction 3: 20.104405027476393
Average distance prediction 4: 21.29843845960165
Average distance prediction 5: 22.561778696542465
Average distance prediction 6: 23.857004634800823
Average distance prediction 7: 25.24111208237998
Average distance prediction 8: 26.697005550597627
Average distance prediction 9: 28.236189186807863
```

[567]:
```python
# Lasso
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import Lasso
model = Lasso()
pred, real = call_get_next_10_predictions(X_set, model)
avg_distances = get_avg_distance_10_predictions(pred, real)
plot_distance_errors(avg_distances)
for i in range(len(avg_distances)):
    print(f"Average distance prediction {i}: {avg_distances[i]}")
```

Error en la predicción de posiciones

```
Average distance prediction 0: 17.22967477896253
Average distance prediction 1: 18.034254298398118
Average distance prediction 2: 18.986880910591964
Average distance prediction 3: 20.104416236887317
Average distance prediction 4: 21.29844855985368
Average distance prediction 5: 22.561788014849558
Average distance prediction 6: 23.857013526964483
Average distance prediction 7: 25.241120922598153
Average distance prediction 8: 26.69701472805039
Average distance prediction 9: 28.236199075512555
```
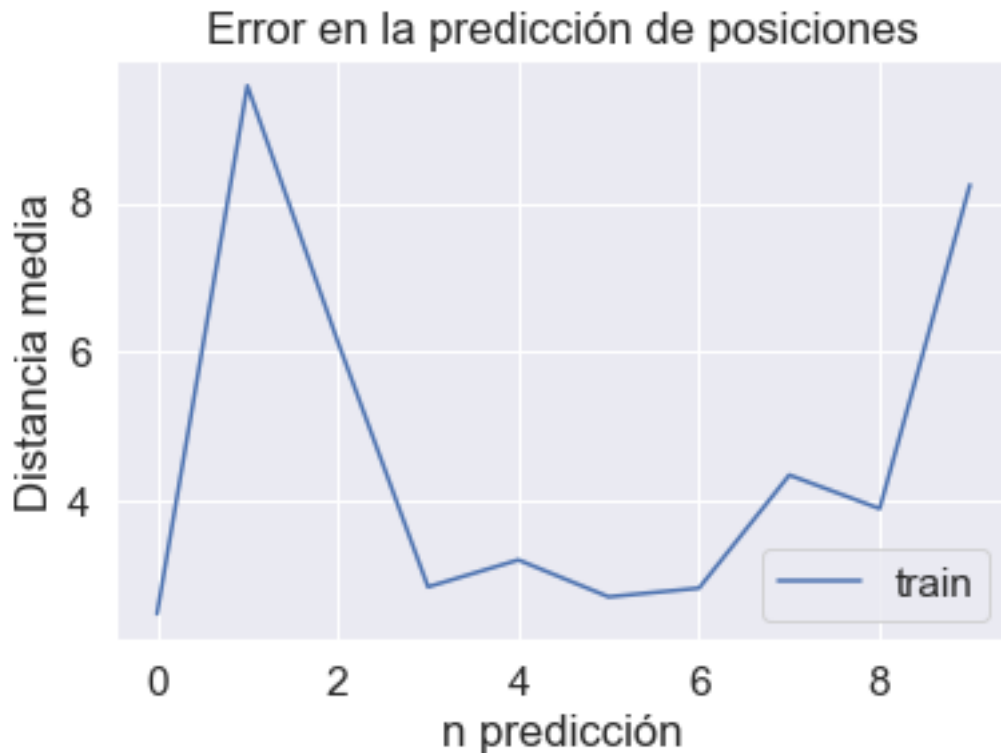
[568]:
```python
# Lars
from sklearn.linear_model import Lars
model = Lars(n_nonzero_coefs=500, eps=0.5, )
pred, real = call_get_next_10_predictions(X_set, model)
avg_distances = get_avg_distance_10_predictions(pred, real)
plot_distance_errors(avg_distances)
for i in range(len(avg_distances)):
    print(f"Average distance prediction {i}: {avg_distances[i]}")
```
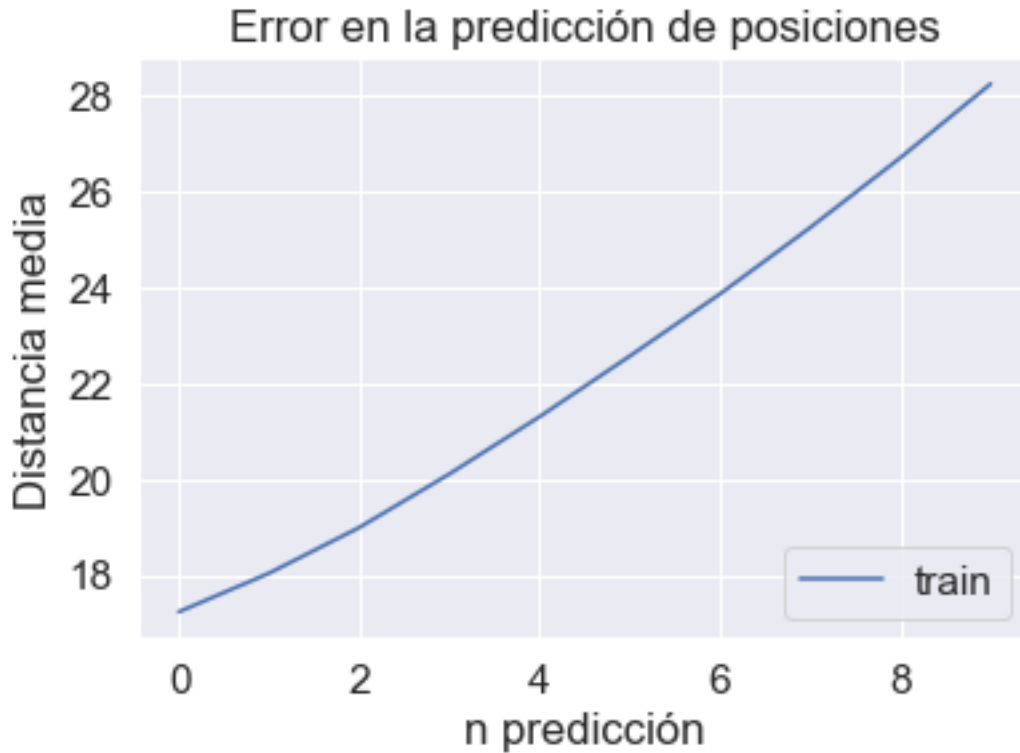
Error en la predicción de posiciones


```
Average distance prediction 0: 2.490397814364312
Average distance prediction 1: 9.572126030119804
Average distance prediction 2: 6.145169121177993
Average distance prediction 3: 2.851976917419119
Average distance prediction 4: 3.2133575668954246
Average distance prediction 5: 2.7181701206898947
Average distance prediction 6: 2.834743463803702
Average distance prediction 7: 4.351860194122392
Average distance prediction 8: 3.901324746183364
Average distance prediction 9: 8.243856399359263
```

```python
[569]:  # LassoLars
        from sklearn.linear_model import LassoLars
        model = LassoLars()
        pred, real = call_get_next_10_predictions(X_set, model)
        avg_distances = get_avg_distance_10_predictions(pred, real)
        plot_distance_errors(avg_distances)
        for i in range(len(avg_distances)):
            print(f"Average distance prediction {i}: {avg_distances[i]}")
```
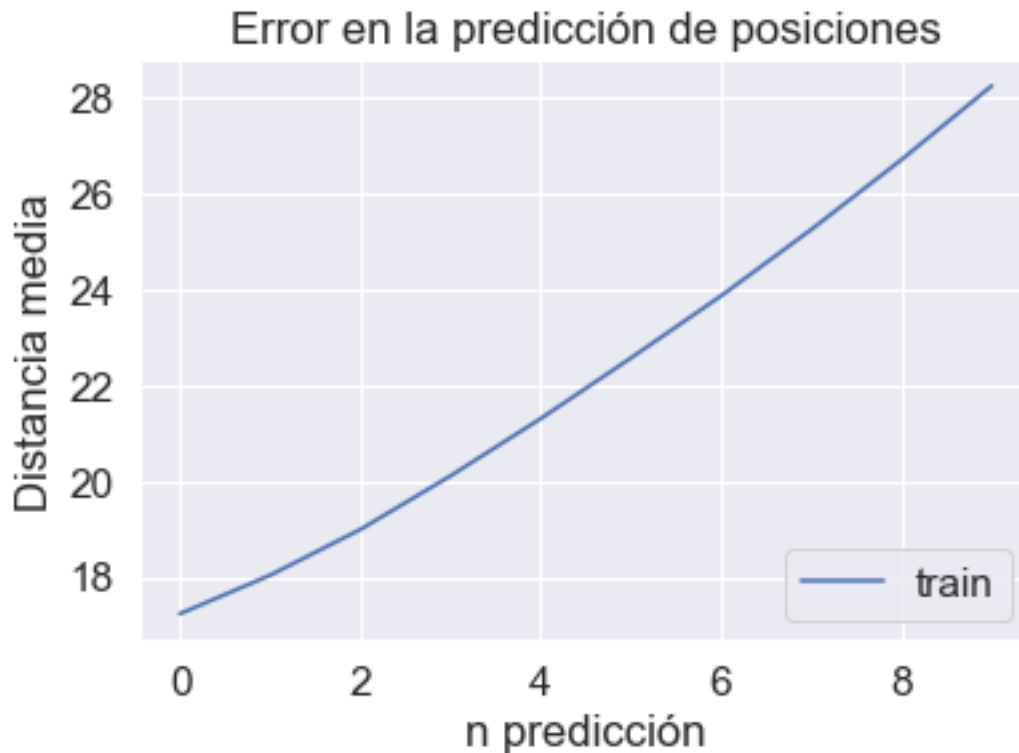
Error en la predicción de posiciones

```
Average distance prediction 0: 17.22967477899418
Average distance prediction 1: 18.034254298435823
Average distance prediction 2: 18.986880910625562
Average distance prediction 3: 20.10441623692093
Average distance prediction 4: 21.298448559876547
Average distance prediction 5: 22.561788014883906
Average distance prediction 6: 23.857013526960273
Average distance prediction 7: 25.241120922574325
Average distance prediction 8: 26.697014728059848
Average distance prediction 9: 28.236199075565224
```

[570]:
```python
# MultiTaskElasticNet
from sklearn.linear_model import MultiTaskElasticNet
model = MultiTaskElasticNet()
pred, real = call_get_next_10_predictions(X_set, model)
avg_distances = get_avg_distance_10_predictions(pred, real)
plot_distance_errors(avg_distances)
for i in range(len(avg_distances)):
    print(f"Average distance prediction {i}: {avg_distances[i]}")
```

Error en la predicción de posiciones


```
Average distance prediction 0: 17.22967477896253
Average distance prediction 1: 18.034254298398118
Average distance prediction 2: 18.986880910591964
Average distance prediction 3: 20.104416236887317
Average distance prediction 4: 21.29844855985368
Average distance prediction 5: 22.561788014849558
Average distance prediction 6: 23.857013526964483
Average distance prediction 7: 25.241120922598153
Average distance prediction 8: 26.69701472805039
Average distance prediction 9: 28.236199075512555
```

```
[571]: # RANSACRegressor
       from sklearn.linear_model import RANSACRegressor
       model = RANSACRegressor()
       pred, real = call_get_next_10_predictions(X_set, model)
       avg_distances = get_avg_distance_10_predictions(pred, real)
       plot_distance_errors(avg_distances)
       for i in range(len(avg_distances)):
           print(f"Average distance prediction {i}: {avg_distances[i]}")
```

Error en la predicción de posiciones
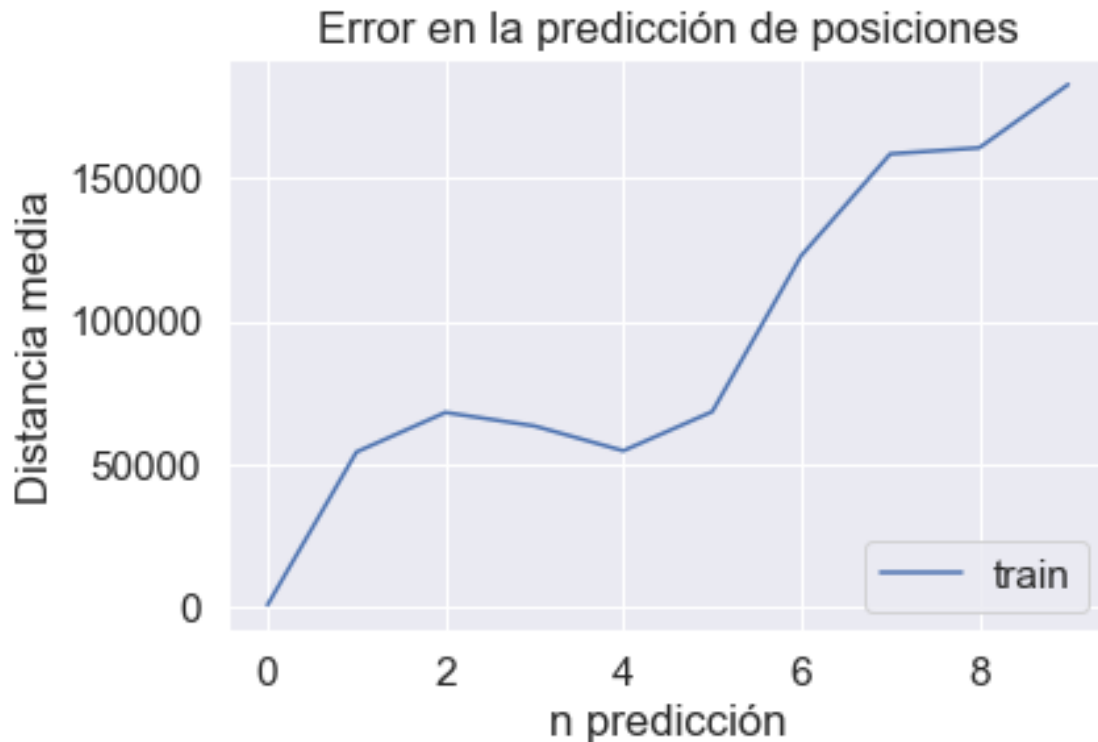

```
Average distance prediction 0: 746.9625153897573
Average distance prediction 1: 54237.016966768635
Average distance prediction 2: 68071.63674747136
Average distance prediction 3: 63352.54050726138
Average distance prediction 4: 54680.30924760415
Average distance prediction 5: 68370.78938748642
Average distance prediction 6: 122878.21123441048
Average distance prediction 7: 158436.37381497983
Average distance prediction 8: 160557.43941087913
Average distance prediction 9: 182694.1720816154
```

[572]:
```python
# DecisionTreeRegressor
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor()
pred, real = call_get_next_10_predictions(X_set, model)
avg_distances = get_avg_distance_10_predictions(pred, real)
plot_distance_errors(avg_distances)
for i in range(len(avg_distances)):
    print(f"Average distance prediction {i}: {avg_distances[i]}")
```

Error en la predicción de posiciones

```
Average distance prediction 0: 4.401235111671969
Average distance prediction 1: 2.7595583926447436
Average distance prediction 2: 2.8522251542289236
Average distance prediction 3: 3.0639891098776797
Average distance prediction 4: 3.248108595168349
Average distance prediction 5: 3.195360285315973
Average distance prediction 6: 3.202830856137052
Average distance prediction 7: 3.264666587591404
Average distance prediction 8: 3.425822552599451
Average distance prediction 9: 3.5891731442798287
```

```python
[573]: # ExtraTreeRegressor
from sklearn.tree import ExtraTreeRegressor
model = ExtraTreeRegressor()
pred, real = call_get_next_10_predictions(X_set, model)
avg_distances = get_avg_distance_10_predictions(pred, real)
plot_distance_errors(avg_distances)
for i in range(len(avg_distances)):
    print(f"Average distance prediction {i}: {avg_distances[i]}")
```

Error en la predicción de posiciones


```
Average distance prediction 0: 4.250151546205461
Average distance prediction 1: 2.5711719383589986
Average distance prediction 2: 2.7721436575449947
Average distance prediction 3: 3.2696173719392236
Average distance prediction 4: 3.4807276631976642
Average distance prediction 5: 3.7659091114647154
Average distance prediction 6: 3.950379413580073
Average distance prediction 7: 4.0487888413698645
Average distance prediction 8: 4.063180047295977
Average distance prediction 9: 4.100316219759913
```

[574]:
```python
# MLPRegressor
from sklearn.neural_network import MLPRegressor
model = MLPRegressor(hidden_layer_sizes=100, activation='relu',
 ↪solver='lbfgs',alpha=0.001, learning_rate='adaptive', max_fun=10000)
pred, real = call_get_next_10_predictions(X_set, model)
avg_distances = get_avg_distance_10_predictions(pred, real)
plot_distance_errors(avg_distances)
for i in range(len(avg_distances)):
    print(f"Average distance prediction {i}: {avg_distances[i]}")
```
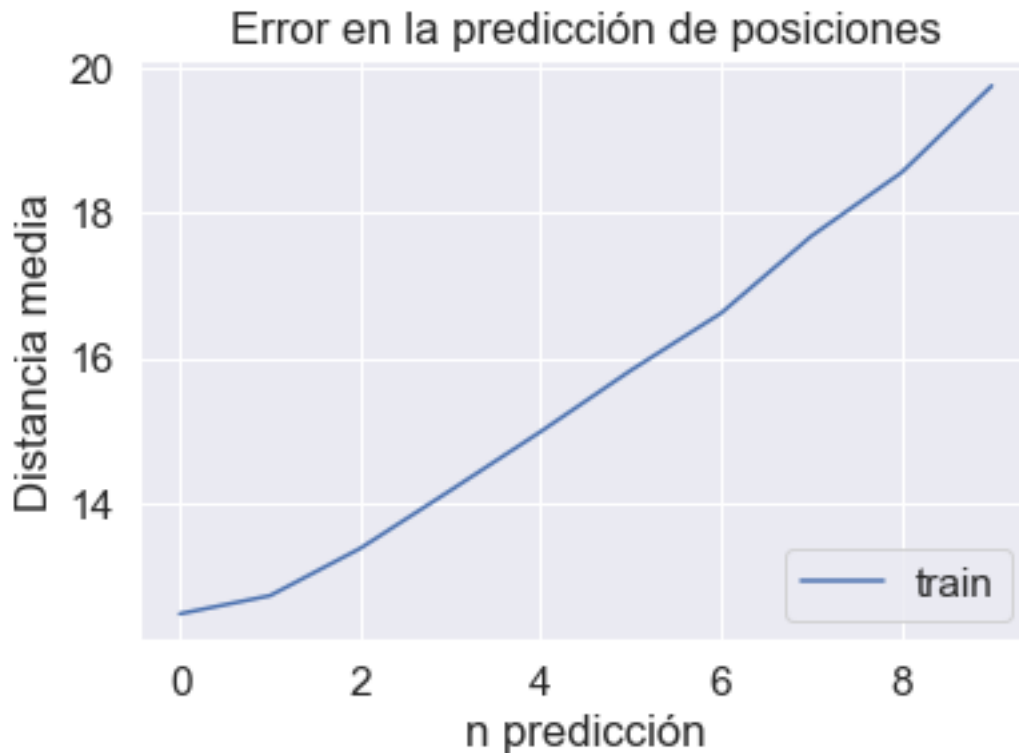
Error en la predicción de posiciones

```
Average distance prediction 0: 12.465900743327735
Average distance prediction 1: 12.716922046924067
Average distance prediction 2: 13.370140182564572
Average distance prediction 3: 14.17088626797173
Average distance prediction 4: 14.983127144636594
Average distance prediction 5: 15.830191720020078
Average distance prediction 6: 16.620066231656637
Average distance prediction 7: 17.683880791873342
Average distance prediction 8: 18.562157921658837
Average distance prediction 9: 19.76083553098903
```

[575]:
```python
# KNeighborsRegressor
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=1)
pred, real = call_get_next_10_predictions(X_set, model)
avg_distances = get_avg_distance_10_predictions(pred, real)
plot_distance_errors(avg_distances)
for i in range(len(avg_distances)):
    print(f"Average distance prediction {i}: {avg_distances[i]}")
```

Error en la predicción de posiciones


```
Average distance prediction 0: 4.183014630167164
Average distance prediction 1: 2.288490733581743
Average distance prediction 2: 2.4125395500691056
Average distance prediction 3: 2.711811372133221
Average distance prediction 4: 3.0250712856107205
Average distance prediction 5: 3.3570110403893008
Average distance prediction 6: 3.7075498826858837
Average distance prediction 7: 4.127609560308159
Average distance prediction 8: 4.596142147457087
Average distance prediction 9: 5.129526813103829
```

[576]:
```python
'''# RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=5)
pred, real = call_get_next_10_predictions(X_set, model)
avg_distances = get_avg_distance_10_predictions(pred, real)
plot_distance_errors(avg_distances)
for i in range(len(avg_distances)):
    print(f"Average distance prediction {i}: {avg_distances[i]}")'''
```

[576]: '# RandomForestRegressor\nfrom sklearn.ensemble import
RandomForestRegressor\nmodel = RandomForestRegressor(n_estimators=5)\npred, real
= call_get_next_10_predictions(X_set, model)\navg_distances =

```
get_avg_distance_10_predictions(pred,
real)\nplot_distance_errors(avg_distances)\nfor i in
range(len(avg_distances)):\n    print(f"Average distance prediction {i}:
{avg_distances[i]}")'
```

[577]:
```python
# BaggingRegressor
from sklearn.ensemble import BaggingRegressor
model = BaggingRegressor()
pred, real = call_get_next_10_predictions(X_set, model)
avg_distances = get_avg_distance_10_predictions(pred, real)
plot_distance_errors(avg_distances)
for i in range(len(avg_distances)):
    print(f"Average distance prediction {i}: {avg_distances[i]}")
```



```
Average distance prediction 0: 5.405870415947143
Average distance prediction 1: 4.791751185699113
Average distance prediction 2: 4.522194414104208
Average distance prediction 3: 4.672396885755094
Average distance prediction 4: 4.71449554797639
Average distance prediction 5: 4.742793085581139
Average distance prediction 6: 4.721606402134539
Average distance prediction 7: 4.603267796768419
Average distance prediction 8: 4.441288253379829
```

Average distance prediction 9: 4.439785708864413

```python
# GaussianProcessRegressor
from sklearn.gaussian_process import GaussianProcessRegressor
model = GaussianProcessRegressor()
pred, real = call_get_next_10_predictions(X_set, model)
avg_distances = get_avg_distance_10_predictions(pred, real)
plot_distance_errors(avg_distances)
for i in range(len(avg_distances)):
    print(f"Average distance prediction {i}: {avg_distances[i]}")
```
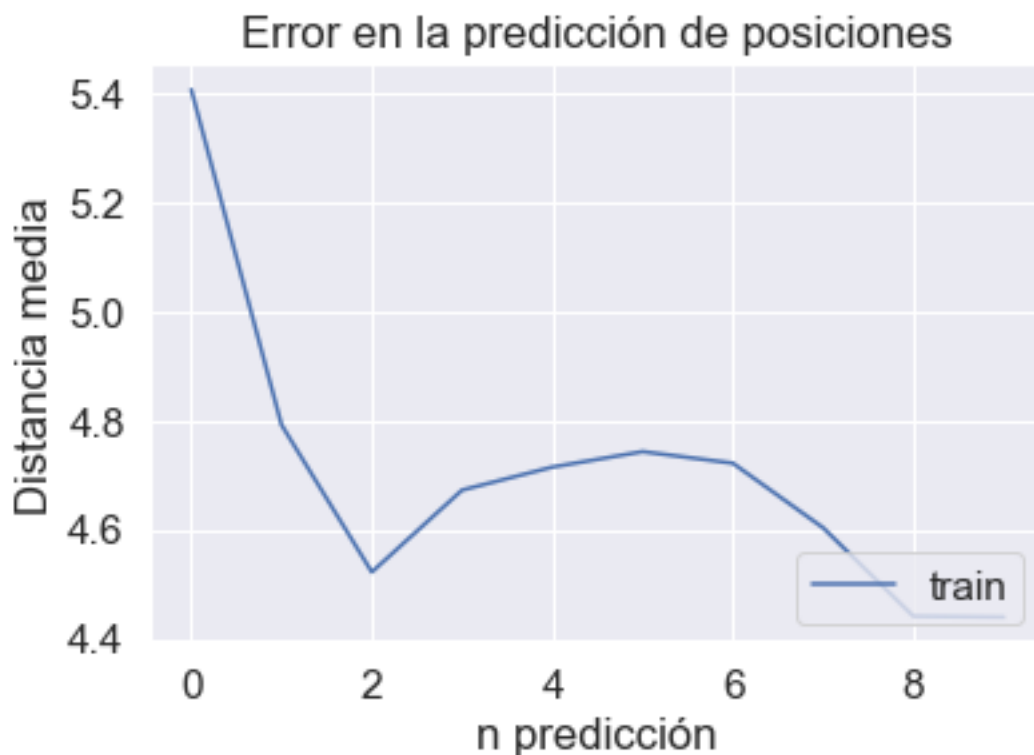

Average distance prediction 0: 2.4035762407747074
Average distance prediction 1: 2.4789935089459245
Average distance prediction 2: 3.002679390803188
Average distance prediction 3: 2.615643318101096
Average distance prediction 4: 2.5906364462646922
Average distance prediction 5: 2.6706546342136415
Average distance prediction 6: 2.543070195044613
Average distance prediction 7: 2.645207795064917
Average distance prediction 8: 2.9015549450656284
Average distance prediction 9: 3.118560610608869

```
[579]:  # DummyRegressor
        from sklearn.dummy import DummyRegressor
        model = DummyRegressor()
        pred, real = call_get_next_10_predictions(X_set, model)
        avg_distances = get_avg_distance_10_predictions(pred, real)
        plot_distance_errors(avg_distances)
        for i in range(len(avg_distances)):
            print(f"Average distance prediction {i}: {avg_distances[i]}")
```



```
Average distance prediction 0: 17.22967477899418
Average distance prediction 1: 18.034254298435823
Average distance prediction 2: 18.986880910625562
Average distance prediction 3: 20.10441623692093
Average distance prediction 4: 21.298448559876547
Average distance prediction 5: 22.561788014883906
Average distance prediction 6: 23.857013526960273
Average distance prediction 7: 25.241120922574325
Average distance prediction 8: 26.697014728059848
Average distance prediction 9: 28.236199075565224
```

2 Modelo MLP

2.0.1 Creación del dataset

```
[580]: df = pd.read_csv("positions_1000.csv")
```

```
[581]: def format_dataset(df, n):

           df = df.sort_values(['vehicle_id', 'time'])
           df = df.reset_index(drop=True)

           data = []
           temp_data = []
           temp_i = 0
           temp_vehicle = df.loc[0, 'vehicle_id']

           for i in range(1, df.shape[0]):

               row = df.loc[i]

               if row["vehicle_id"] != temp_vehicle:
                   temp_vehicle = row["vehicle_id"]
                   temp_i = 0
                   temp_data = []
                   temp_data.append(row["latitude"])
                   temp_data.append(row["longitude"])

               elif temp_i < n:
                   temp_data.append(row["latitude"])
                   temp_data.append(row["longitude"])

               else:
                   temp_data.append(row["latitude"])
                   temp_data.append(row["longitude"])
                   data.append(temp_data)
                   temp_data = []
                   temp_i = 0
                   temp_vehicle = -1
               temp_i += 1

           #X = np.array(X, dtype='float64')
           #y = np.array(y, dtype='float64')

           return data

       #mlp_data = format_dataset(df, 24)
```

```
[582]: columns = ["lat_0", "long_0", "lat_1", "long_1", "lat_2", "long_2", "lat_3",␣
       ↪"long_3", "lat_4", "long_4",
               "lat_5", "long_5", "lat_6", "long_6", "lat_7", "long_7", "lat_8",␣
       ↪"long_8", "lat_9", "long_9",
               "lat_10", "long_10", "lat_11", "long_11", "lat_12", "long_12",␣
       ↪"lat_13", "long_13", "lat_14", "long_14",
               "lat_15", "long_15", "lat_16", "long_16", "lat_17", "long_17",␣
       ↪"lat_18", "long_18", "lat_19", "long_19",
               "lat_20", "long_20", "lat_21", "long_21", "lat_22", "long_22",␣
       ↪"lat_23", "long_23", "lat_24", "long_24"]
```

```
[583]: #df_mlp = pd.DataFrame(mlp_data, columns=columns )
```

```
[584]: #df_mlp.to_csv("df_mlp_19.csv")
```

2.0.2 Preparación de datos

```
[585]: df_mlp = pd.read_csv("df_mlp_24.csv")
```

```
[586]: len(df_mlp)
```

```
[586]: 151099
```

```
[587]: #df_mlp = df_mlp.loc[(df_mlp["lat_0"]!=df_mlp["lat_1"]) & (df_mlp["long_0"]!
       ↪=df_mlp["long_1"])]
```

```
[588]: df_mlp = df_mlp.drop(["Unnamed: 0"], axis=1)
```

```
[589]: y_mlp = df_mlp[["lat_15", "long_15", "lat_16", "long_16", "lat_17", "long_17",␣
       ↪"lat_18", "long_18", "lat_19", "long_19","lat_20", "long_20", "lat_21",␣
       ↪"long_21", "lat_22", "long_22", "lat_23", "long_23", "lat_24", "long_24"]]
       X_mlp = df_mlp#.drop(["lat_15", "long_15"], axis=1)
```

```
[590]: df_mlp
```

```
[590]:              lat_0     long_0      lat_1     long_1      lat_2     long_2  \
       0        41.390949   2.163105  41.390879   2.163014  41.390819   2.162934
       1        41.389290   2.160920  41.389221   2.160824  41.389154   2.160731
       2        41.390644   2.167087  41.390662   2.167065  41.390679   2.167043
       3        41.391037   2.166599  41.391053   2.166580  41.391060   2.166572
       4        41.391187   2.166414  41.391198   2.166400  41.391202   2.166396
       …              …          …          …          …          …          …
       151094   41.389972   2.162020  41.389963   2.162007  41.389962   2.162005
       151095   41.393290   2.163255  41.393283   2.163263  41.393271   2.163279
       151096   41.391992   2.162036  41.391979   2.162054  41.391945   2.162100
       151097   41.390042   2.161647  41.390052   2.161697  41.390035   2.161731
       151098   41.390493   2.166997  41.390525   2.167009  41.390546   2.166983
```

```
           lat_3     long_3     lat_4     long_4  …     lat_20    long_20  \
0       41.390756  2.162850  41.390692  2.162765  …  41.389624   2.161361
1       41.389091  2.160646  41.389025  2.160558  …  41.387971   2.159163
2       41.390698  2.167020  41.390716  2.166998  …  41.390991   2.166657
3       41.391072  2.166556  41.391081  2.166545  …  41.391138   2.166475
4       41.391206  2.166391  41.391214  2.166380  …  41.391243   2.166345
…             …          …         …          …    …        …           …
151094  41.389945  2.161984  41.389910  2.161949  …  41.389313   2.162742
151095  41.393256  2.163297  41.393245  2.163311  …  41.393085   2.163511
151096  41.391890  2.162176  41.391832  2.162255  …  41.391231   2.163095
151097  41.390032  2.161738  41.390031  2.161740  …  41.390030   2.161742
151098  41.390599  2.166971  41.390638  2.166922  …  41.391396   2.166175

           lat_21    long_21    lat_22    long_22    lat_23    long_23  \
0       41.389556  2.161278  41.389495  2.161201  41.389422  2.161102
1       41.387898  2.159066  41.387827  2.158973  41.387761  2.158886
2       41.390997  2.166650  41.391008  2.166636  41.391020  2.166621
3       41.391147  2.166463  41.391155  2.166453  41.391161  2.166447
4       41.391243  2.166345  41.391243  2.166345  41.391243  2.166345
…             …          …         …          …         …          …
151094  41.389249  2.162826  41.389188  2.162906  41.389162  2.162940
151095  41.393077  2.163521  41.393065  2.163536  41.393060  2.163542
151096  41.391231  2.163095  41.391231  2.163095  41.391231  2.163095
151097  41.390030  2.161742  41.390030  2.161742  41.390030  2.161742
151098  41.391454  2.166219  41.391500  2.166270  41.391540  2.166324

           lat_24    long_24
0       41.389358  2.161013
1       41.387690  2.158791
2       41.391035  2.166602
3       41.391170  2.166435
4       41.391243  2.166345
…             …          …
151094  41.389155  2.162950
151095  41.393060  2.163542
151096  41.391231  2.163095
151097  41.390030  2.161742
151098  41.391580  2.166376

[151099 rows x 50 columns]
```

2.0.3 Estandarización

```
[591]: X_mlp_train_full, X_mlp_test_full, y_mlp_train_full, y_mlp_test_full =
        ↪train_test_split(X_mlp, y_mlp, test_size=0.25, random_state=42)
       print(f"Shape X_mlp_train: {X_mlp_train_full.shape}")
       print(f"Shape X_mlp_test: {X_mlp_test_full.shape}")
       print(f"Shape y_mlp_train: {y_mlp_train_full.shape}")
       print(f"Shape y_mlp_test: {y_mlp_test_full.shape}")
```

```
Shape X_mlp_train: (113324, 50)
Shape X_mlp_test: (37775, 50)
Shape y_mlp_train: (113324, 20)
Shape y_mlp_test: (37775, 20)
```

```
[592]: y_aaa = y_mlp_train_full[["lat_15", "long_15"]]
       y_aaaa = y_mlp_test_full[["lat_15", "long_15"]]
       print(f"Shape X_mlp_train: {X_mlp_train_full.shape}")
       print(f"Shape X_mlp_test: {X_mlp_test_full.shape}")
       print(f"Shape y_mlp_train: {y_aaa.shape}")
       print(f"Shape y_mlp_test: {y_aaaa.shape}")
```

```
Shape X_mlp_train: (113324, 50)
Shape X_mlp_test: (37775, 50)
Shape y_mlp_train: (113324, 2)
Shape y_mlp_test: (37775, 2)
```

```
[593]: #Normalizar los datos
       X_mlp_train = X_mlp_train_full.drop(["lat_15", "long_15", "lat_16", "long_16",
        ↪"lat_17", "long_17", "lat_18", "long_18", "lat_19", "long_19", "lat_20",
        ↪"long_20", "lat_21", "long_21", "lat_22", "long_22", "lat_23", "long_23",
        ↪"lat_24", "long_24"], axis=1)
       X_mlp_test = X_mlp_test_full.drop(["lat_15", "long_15", "lat_16", "long_16",
        ↪"lat_17", "long_17", "lat_18", "long_18", "lat_19", "long_19", "lat_20",
        ↪"long_20", "lat_21", "long_21", "lat_22", "long_22", "lat_23", "long_23",
        ↪"lat_24", "long_24"], axis=1)
       scaler = StandardScaler().fit(X_mlp_train)
       X_mlp_train = scaler.transform(X_mlp_train)
       X_mlp_test = scaler.transform(X_mlp_test)


       y_mlp_lat_train_full = y_mlp_train_full[["lat_15", "lat_16", "lat_17",
        ↪"lat_18", "lat_19", "lat_20", "long_20", "lat_21", "long_21", "lat_22",
        ↪"long_22", "lat_23", "long_23", "lat_24", "long_24"]]
       y_mlp_lat_test_full = y_mlp_test_full[["lat_15", "lat_16", "lat_17", "lat_18",
        ↪"lat_19", "lat_20", "long_20", "lat_21", "long_21", "lat_22", "long_22",
        ↪"lat_23", "long_23", "lat_24", "long_24"]]


       scaler_lat = StandardScaler().fit(y_mlp_lat_train_full[["lat_15"]])
```

```
y_mlp_lat_train = y_mlp_lat_train_full[["lat_15"]]
y_mlp_lat_test = y_mlp_lat_test_full[["lat_15"]]
y_mlp_lat_train = scaler_lat.transform(y_mlp_lat_train)
y_mlp_lat_test = scaler_lat.transform(y_mlp_lat_test)


y_mlp_long_train_full = y_mlp_train_full[["long_15", "long_16", "long_17",
  ↪"long_18", "long_19", "lat_20", "long_20", "lat_21", "long_21", "lat_22",
  ↪"long_22", "lat_23", "long_23", "lat_24", "long_24"]]
y_mlp_long_test_full = y_mlp_test_full[["long_15", "long_16", "long_17",
  ↪"long_18", "long_19", "lat_20", "long_20", "lat_21", "long_21", "lat_22",
  ↪"long_22", "lat_23", "long_23", "lat_24", "long_24"]]

scaler_long = StandardScaler().fit(y_mlp_long_train_full[["long_15"]])
y_mlp_long_train = y_mlp_long_train_full[["long_15"]]
y_mlp_long_test = y_mlp_long_test_full[["long_15"]]
y_mlp_long_train = scaler_long.transform(y_mlp_long_train)
y_mlp_long_test = scaler_long.transform(y_mlp_long_test)
```

[594]: `X_mlp_train[0]`

[594]:
```
array([-0.30641891,  0.84317902, -0.2693146 ,  0.88302696, -0.24177196,
        0.91256045, -0.21146159,  0.94512487, -0.18062133,  0.97863912,
       -0.15183434,  1.01042493, -0.12507448,  1.03944564, -0.09318535,
        1.06658539, -0.05707962,  1.09669166, -0.02498172,  1.12365827,
        0.01568591,  1.14410806,  0.05051164,  1.15940871,  0.09024261,
        1.16088545,  0.1201988 ,  1.15258728,  0.13562245,  1.14461973])
```

2.0.4 Modelo

[595]:
```
model_mlp_lat = Sequential([
    Flatten(input_shape=[X_mlp_train.shape[1],]),
    Dense(128, activation="relu", kernel_initializer="normal"),
    Dropout(0.1),
    Dense(64, activation="tanh", kernel_initializer="normal"),
    Dropout(0.2),
    Dense(16, activation="tanh", kernel_initializer="normal"),
    Dropout(0.2),
    Dense(1, activation="linear")
])
```

[596]:
```
model_mlp_long = Sequential([
    Flatten(input_shape=[X_mlp_train.shape[1],]),
    Dense(128, activation="relu", kernel_initializer="normal"),
    Dropout(0.2),
    Dense(64, activation="tanh", kernel_initializer="normal"),
    Dropout(0.2),
```

```
    Dense(16, activation="tanh", kernel_initializer="normal"),
    Dropout(0.2),
    Dense(1, activation="linear")
])
```

[597]: `model_mlp_lat.summary()`

Model: "sequential"

 Layer (type)   Output Shape   Param #
==
 flatten (Flatten)   (None, 30)   0

 dense (Dense)   (None, 128)   3968

 dropout (Dropout)   (None, 128)   0

 dense_1 (Dense)   (None, 64)   8256

 dropout_1 (Dropout)   (None, 64)   0

 dense_2 (Dense)   (None, 16)   1040

 dropout_2 (Dropout)   (None, 16)   0

 dense_3 (Dense)   (None, 1)   17

==
Total params: 13,281
Trainable params: 13,281
Non-trainable params: 0
----

[598]: `model_mlp_long.summary()`

Model: "sequential_1"

 Layer (type)   Output Shape   Param #
==
 flatten_1 (Flatten)   (None, 30)   0

 dense_4 (Dense)   (None, 128)   3968

 dropout_3 (Dropout)   (None, 128)   0

 dense_5 (Dense)   (None, 64)   8256

 dropout_4 (Dropout)   (None, 64)   0
```

```
dense_6 (Dense)                 (None, 16)                    1040

dropout_5 (Dropout)             (None, 16)                    0

dense_7 (Dense)                 (None, 1)                     17

=================================================================
Total params: 13,281
Trainable params: 13,281
Non-trainable params: 0

_____
```

2.0.5 Cálculo latitud

```
[599]: epochs = 18
       batch_size = 256
       optimizer = Adam(learning_rate=0.0001)
       model_mlp_lat.compile(loss='mse', optimizer=optimizer)
       history = model_mlp_lat.fit(X_mlp_train, y_mlp_lat_train, epochs=epochs,␣
         ↪batch_size=batch_size, validation_data=(X_mlp_test, y_mlp_lat_test),␣
         ↪verbose=1)
       model_mlp_lat.summary()
```

```
Epoch 1/18
WARNING:tensorflow:AutoGraph could not transform <function
Model.make_train_function.<locals>.train_function at 0x000002E0C251F0D0> and
will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.
Cause: closure mismatch, requested ('self', 'step_function'), but source
function had ()
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function
Model.make_train_function.<locals>.train_function at 0x000002E0C251F0D0> and
will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.
Cause: closure mismatch, requested ('self', 'step_function'), but source
function had ()
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
439/443 [=============================>.] - ETA: 0s - loss:
0.1537WARNING:tensorflow:AutoGraph could not transform <function
Model.make_test_function.<locals>.test_function at 0x000002E0C0ABDA60> and will
```

run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.
Cause: closure mismatch, requested ('self', 'step_function'), but source
function had ()
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function
Model.make_test_function.<locals>.test_function at 0x000002E0C0ABDA60> and will
run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.
Cause: closure mismatch, requested ('self', 'step_function'), but source
function had ()
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
443/443 [==============================] - 6s 10ms/step - loss: 0.1528 -
val_loss: 0.0164
Epoch 2/18
443/443 [==============================] - 2s 5ms/step - loss: 0.0359 -
val_loss: 0.0055
Epoch 3/18
443/443 [==============================] - 2s 5ms/step - loss: 0.0282 -
val_loss: 0.0037
Epoch 4/18
443/443 [==============================] - 3s 6ms/step - loss: 0.0262 -
val_loss: 0.0034
Epoch 5/18
443/443 [==============================] - 2s 5ms/step - loss: 0.0253 -
val_loss: 0.0030
Epoch 6/18
443/443 [==============================] - 2s 4ms/step - loss: 0.0242 -
val_loss: 0.0023
Epoch 7/18
443/443 [==============================] - 2s 4ms/step - loss: 0.0241 -
val_loss: 0.0025
Epoch 8/18
443/443 [==============================] - 2s 4ms/step - loss: 0.0234 -
val_loss: 0.0026
Epoch 9/18
443/443 [==============================] - 2s 4ms/step - loss: 0.0228 -
val_loss: 0.0017
Epoch 10/18
443/443 [==============================] - 2s 5ms/step - loss: 0.0228 -
val_loss: 0.0017
Epoch 11/18

```
443/443 [==============================] - 3s 6ms/step - loss: 0.0230 -
val_loss: 0.0018
Epoch 12/18
443/443 [==============================] - 3s 7ms/step - loss: 0.0227 -
val_loss: 0.0015
Epoch 13/18
443/443 [==============================] - 2s 5ms/step - loss: 0.0222 -
val_loss: 0.0015
Epoch 14/18
443/443 [==============================] - 2s 6ms/step - loss: 0.0222 -
val_loss: 0.0016
Epoch 15/18
443/443 [==============================] - 3s 6ms/step - loss: 0.0218 -
val_loss: 0.0022
Epoch 16/18
443/443 [==============================] - 3s 6ms/step - loss: 0.0221 -
val_loss: 0.0017
Epoch 17/18
443/443 [==============================] - 2s 5ms/step - loss: 0.0219 -
val_loss: 9.2416e-04
Epoch 18/18
443/443 [==============================] - 3s 6ms/step - loss: 0.0219 -
val_loss: 0.0015
Model: "sequential"
_____
 Layer (type)              Output Shape             Param #
=================================================================
 flatten (Flatten)         (None, 30)               0

 dense (Dense)             (None, 128)              3968

 dropout (Dropout)         (None, 128)              0

 dense_1 (Dense)           (None, 64)               8256

 dropout_1 (Dropout)       (None, 64)               0

 dense_2 (Dense)           (None, 16)               1040

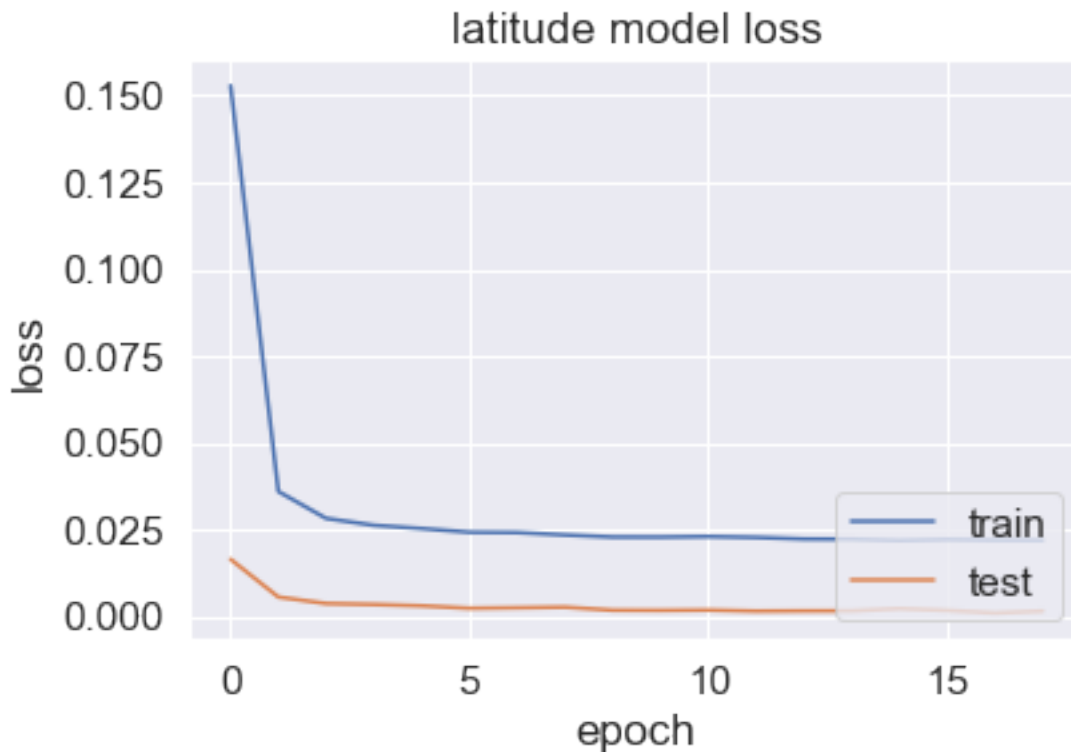 dropout_2 (Dropout)       (None, 16)               0

 dense_3 (Dense)           (None, 1)                17

=================================================================
Total params: 13,281
Trainable params: 13,281
Non-trainable params: 0
_____
```

```
[600]:  #Plots
        %matplotlib inline

        # Visualizamos la evolución de la accuracy
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('latitude model loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='lower right')
        plt.show()
```



```
[601]:  y_lat_pred = model_mlp_lat.predict(X_mlp_test[0:1000])
        y_lat_pred = scaler_lat.inverse_transform(y_lat_pred)
        y_lat_real = scaler_lat.inverse_transform(y_mlp_lat_test)
```

WARNING:tensorflow:AutoGraph could not transform <function
Model.make_predict_function.<locals>.predict_function at 0x000002E0BEDA0700> and
will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.
Cause: closure mismatch, requested ('self', 'step_function'), but source

```
function had ()
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function
Model.make_predict_function.<locals>.predict_function at 0x000002E0BEDA0700> and
will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.
Cause: closure mismatch, requested ('self', 'step_function'), but source
function had ()
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
```

[602]:
```python
print(y_lat_pred[0:10])
print(y_lat_real[0:10])
```

```
[[41.391293]
 [41.390358]
 [41.393314]
 [41.389812]
 [41.392044]
 [41.390003]
 [41.390022]
 [41.390747]
 [41.39087 ]
 [41.392483]]
[[41.39128345]
 [41.3903562 ]
 [41.39332725]
 [41.38976118]
 [41.39206235]
 [41.38996487]
 [41.38994085]
 [41.3906968 ]
 [41.3908445 ]
 [41.39248576]]
```

2.0.6 Cálculo longitud

[603]:
```python
epochs = 18
batch_size = 256
optimizer = Adam(learning_rate=0.0001)
model_mlp_long.compile(loss='mse', optimizer=optimizer)
history = model_mlp_long.fit(X_mlp_train, y_mlp_long_train, epochs=epochs,␣
  ↪batch_size=batch_size, validation_data=(X_mlp_test, y_mlp_long_test),␣
  ↪verbose=1)
model_mlp_long.summary()
```

```
Epoch 1/18
WARNING:tensorflow:AutoGraph could not transform <function
Model.make_train_function.<locals>.train_function at 0x000002E0C22ABD30> and
will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.
Cause: closure mismatch, requested ('self', 'step_function'), but source
function had ()
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function
Model.make_train_function.<locals>.train_function at 0x000002E0C22ABD30> and
will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.
Cause: closure mismatch, requested ('self', 'step_function'), but source
function had ()
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
432/443 [============================>.] - ETA: 0s - loss:
0.1554WARNING:tensorflow:AutoGraph could not transform <function
Model.make_test_function.<locals>.test_function at 0x000002E0C002C280> and will
run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.
Cause: closure mismatch, requested ('self', 'step_function'), but source
function had ()
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function
Model.make_test_function.<locals>.test_function at 0x000002E0C002C280> and will
run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.
Cause: closure mismatch, requested ('self', 'step_function'), but source
function had ()
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
443/443 [==============================] - 3s 5ms/step - loss: 0.1531 -
val_loss: 0.0251
Epoch 2/18
443/443 [==============================] - 2s 4ms/step - loss: 0.0469 -
val_loss: 0.0094
Epoch 3/18
```

```
443/443 [==============================] - 2s 5ms/step - loss: 0.0365 -
val_loss: 0.0071
Epoch 4/18
443/443 [==============================] - 3s 7ms/step - loss: 0.0321 -
val_loss: 0.0062
Epoch 5/18
443/443 [==============================] - 2s 5ms/step - loss: 0.0304 -
val_loss: 0.0052
Epoch 6/18
443/443 [==============================] - 2s 5ms/step - loss: 0.0290 -
val_loss: 0.0049
Epoch 7/18
443/443 [==============================] - 2s 5ms/step - loss: 0.0277 -
val_loss: 0.0037
Epoch 8/18
443/443 [==============================] - 2s 5ms/step - loss: 0.0267 -
val_loss: 0.0031
Epoch 9/18
443/443 [==============================] - 3s 6ms/step - loss: 0.0263 -
val_loss: 0.0036
Epoch 10/18
443/443 [==============================] - 3s 7ms/step - loss: 0.0253 -
val_loss: 0.0028
Epoch 11/18
443/443 [==============================] - 3s 6ms/step - loss: 0.0249 -
val_loss: 0.0018
Epoch 12/18
443/443 [==============================] - 4s 9ms/step - loss: 0.0245 -
val_loss: 0.0023
Epoch 13/18
443/443 [==============================] - 3s 7ms/step - loss: 0.0244 -
val_loss: 0.0015
Epoch 14/18
443/443 [==============================] - 3s 6ms/step - loss: 0.0240 -
val_loss: 0.0018
Epoch 15/18
443/443 [==============================] - 3s 8ms/step - loss: 0.0237 -
val_loss: 0.0013
Epoch 16/18
443/443 [==============================] - 2s 5ms/step - loss: 0.0236 -
val_loss: 0.0015
Epoch 17/18
443/443 [==============================] - 2s 5ms/step - loss: 0.0240 -
val_loss: 0.0017
Epoch 18/18
443/443 [==============================] - 2s 5ms/step - loss: 0.0239 -
val_loss: 0.0016
Model: "sequential_1"
```

```
----------------------------------------------------------------
Layer (type)                Output Shape              Param #
================================================================
flatten_1 (Flatten)         (None, 30)                0

dense_4 (Dense)             (None, 128)               3968

dropout_3 (Dropout)         (None, 128)               0

dense_5 (Dense)             (None, 64)                8256

dropout_4 (Dropout)         (None, 64)                0

dense_6 (Dense)             (None, 16)                1040

dropout_5 (Dropout)         (None, 16)                0

dense_7 (Dense)             (None, 1)                 17


================================================================
Total params: 13,281
Trainable params: 13,281
Non-trainable params: 0

----------------------------------------------------------------
```

[604]:
```python
#Plots
%matplotlib inline
import matplotlib.pyplot as plt

# Visualizamos la evolución de la accuracy
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('longitude model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')
plt.show()
```

longitude model loss
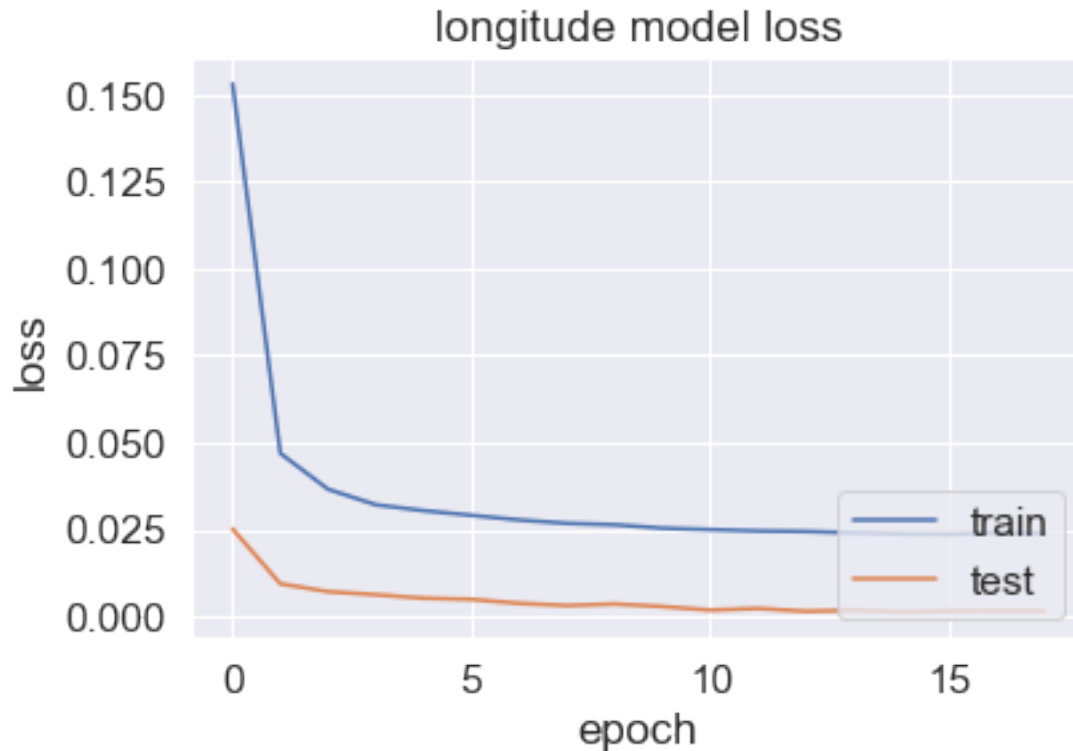
[605]:
```python
y_long_pred = model_mlp_long.predict(X_mlp_test[0:1000])
y_long_pred = scaler_long.inverse_transform(y_long_pred)
y_long_real = scaler_long.inverse_transform(y_mlp_long_test)
print(y_long_pred[0:10])
print(y_long_real[0:10])
```

WARNING:tensorflow:AutoGraph could not transform <function
Model.make_predict_function.<locals>.predict_function at 0x000002E0D04A5700> and
will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.
Cause: closure mismatch, requested ('self', 'step_function'), but source
function had ()
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function
Model.make_predict_function.<locals>.predict_function at 0x000002E0D04A5700> and
will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.
Cause: closure mismatch, requested ('self', 'step_function'), but source

```
function had ()
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
[[2.1662493]
 [2.1646667]
 [2.1634624]
 [2.1653879]
 [2.1650794]
 [2.164174 ]
 [2.1648023]
 [2.1640968]
 [2.1636622]
 [2.1644373]]
[[2.16629446]
 [2.16462951]
 [2.16345464]
 [2.16541696]
 [2.16511773]
 [2.164142  ]
 [2.16484642]
 [2.1641474 ]
 [2.1636299 ]
 [2.1644339 ]]
```

2.0.7 Resultados

```
[606]: def get_avg_distance(lat_pred,long_pred,lat_real,long_real):
           total_distance = 0
           square_error = 0
           for i in range(len(lat_pred)):
               distance = hs.haversine(( lat_pred[i][0], long_pred[i][0] ),(␣
        ↪lat_real[i][0], long_real[i][0] ), unit=Unit.METERS)
               total_distance += distance
               square_error += distance**2
           avg_distance = total_distance/len(lat_pred)
           std_deviation = math.sqrt(square_error)/(len(lat_pred)-1)
           return avg_distance, std_deviation
```

```
[607]: avg_distance = get_avg_distance(y_lat_pred,y_long_pred,y_lat_real,y_long_real)
       print(f"Average distance {avg_distance}")
```

```
Average distance (5.067601507336725, 0.24933699686066)
```

2.1 Predicción de siguientes posiciones

```
[608]: columns  = ["lat_0", "long_0", "lat_1", "long_1", "lat_2", "long_2", "lat_3",␣
        ↪"long_3", "lat_4", "long_4",
                   "lat_5", "long_5", "lat_6", "long_6", "lat_7", "long_7", "lat_8",␣
        ↪"long_8", "lat_9", "long_9",
                   "lat_10", "long_10", "lat_11", "long_11", "lat_12", "long_12",␣
        ↪"lat_13", "long_13", "lat_14", "long_14"]
```

```
[609]: df_res = pd.DataFrame(columns=columns)
```

```
[610]: ini = 0
       fin = 1000
       X_mlp_test = X_mlp_test[ini:fin]
       y_mlp_lat_test = y_mlp_lat_test[ini:fin]
       y_mlp_long_test = y_mlp_long_test[ini:fin]
       X_mlp_test_real = X_mlp_test_full[ini:fin]

       # Create predictions dataframe
       df_pred = X_mlp_test_real[columns]


       ############## FIRST ITERATION ##############
       # Predicción 1
       start = time.time()
       y_lat_pred_1 = model_mlp_lat.predict(X_mlp_test)
       y_lat_pred_1 = scaler_lat.inverse_transform(y_lat_pred_1)
       y_lat_real_1 = scaler_lat.inverse_transform(y_mlp_lat_test)

       y_long_pred_1 = model_mlp_long.predict(X_mlp_test)
       y_long_pred_1 = scaler_long.inverse_transform(y_long_pred_1)
       y_long_real_1 = scaler_long.inverse_transform(y_mlp_long_test)
       end = time.time()
       print(f"Time 1000 predictions: {end - start}s")


       ############## SECOND ITERATION ##############
       # Datos para predicción 2
       X_mlp_test = X_mlp_test_real.drop(["lat_0", "long_0", "lat_15", "long_15",␣
         ↪"lat_16", "long_16", "lat_17", "long_17",
                                          "lat_18", "long_18", "lat_19", "long_19",␣
         ↪"lat_20", "long_20", "lat_21", "long_21",
                                          "lat_22", "long_22", "lat_23", "long_23",␣
         ↪"lat_24", "long_24"], axis=1)
       X_mlp_test["lat_15"] = y_lat_pred_1
       X_mlp_test["long_15"] = y_long_pred_1
       X_mlp_test.columns = columns
       scaler = StandardScaler().fit(X_mlp_test)
       X_mlp_test = scaler.transform(X_mlp_test)
```

```python
y_mlp_lat_test = y_mlp_lat_test_full[["lat_16"]]
y_mlp_long_test = y_mlp_long_test_full[["long_16"]]
y_mlp_lat_test.columns = ["lat_15"]
y_mlp_long_test.columns = ["long_15"]
scaler_lat = StandardScaler().fit(y_mlp_lat_test)
scaler_long = StandardScaler().fit(y_mlp_long_test)
y_mlp_lat_test = scaler_lat.transform(y_mlp_lat_test)
y_mlp_long_test = scaler_long.transform(y_mlp_long_test)

# Predicción 2
y_lat_pred_2 = model_mlp_lat.predict(X_mlp_test)
y_lat_pred_2 = scaler_lat.inverse_transform(y_lat_pred_2)
y_lat_real_2 = scaler_lat.inverse_transform(y_mlp_lat_test)

y_long_pred_2 = model_mlp_long.predict(X_mlp_test)
y_long_pred_2 = scaler_long.inverse_transform(y_long_pred_2)
y_long_real_2 = scaler_long.inverse_transform(y_mlp_long_test)


############## THIRD ITERATION ##############
# Datos para predicción 3
X_mlp_test = X_mlp_test_real.drop(["lat_0", "long_0", "lat_1", "long_1",
 ↪"lat_15", "long_15", "lat_16", "long_16",
                                   "lat_17", "long_17", "lat_18", "long_18",
 ↪"lat_19", "long_19", "lat_20", "long_20",
                                   "lat_21", "long_21", "lat_22", "long_22",
 ↪"lat_23", "long_23", "lat_24", "long_24"], axis=1)
X_mlp_test["lat_15"] = y_lat_pred_1
X_mlp_test["long_15"] = y_long_pred_1
X_mlp_test["lat_16"] = y_lat_pred_2
X_mlp_test["long_16"] = y_long_pred_2
X_mlp_test.columns = columns
scaler = StandardScaler().fit(X_mlp_test)
X_mlp_test = scaler.transform(X_mlp_test)

y_mlp_lat_test = y_mlp_lat_test_full[["lat_17"]]
y_mlp_long_test = y_mlp_long_test_full[["long_17"]]
y_mlp_lat_test.columns = ["lat_15"]
y_mlp_long_test.columns = ["long_15"]
scaler_lat = StandardScaler().fit(y_mlp_lat_test)
scaler_long = StandardScaler().fit(y_mlp_long_test)
y_mlp_lat_test = scaler_lat.transform(y_mlp_lat_test)
y_mlp_long_test = scaler_long.transform(y_mlp_long_test)

# Predicción 3
y_lat_pred_3 = model_mlp_lat.predict(X_mlp_test)
```

```python
y_lat_pred_3 = scaler_lat.inverse_transform(y_lat_pred_3)
y_lat_real_3 = scaler_lat.inverse_transform(y_mlp_lat_test)

y_long_pred_3 = model_mlp_long.predict(X_mlp_test)
y_long_pred_3 = scaler_long.inverse_transform(y_long_pred_3)
y_long_real_3 = scaler_long.inverse_transform(y_mlp_long_test)


############## FOURTH ITERATION ##############
# Datos para predicción 4
X_mlp_test = X_mlp_test_real.drop(["lat_0", "long_0", "lat_1", "long_1",
 ↪"lat_2", "long_2", "lat_15", "long_15",
                                   "lat_16", "long_16", "lat_17", "long_17",
 ↪"lat_18", "long_18", "lat_19", "long_19",
                                   "lat_20", "long_20", "lat_21", "long_21",
 ↪"lat_22", "long_22", "lat_23", "long_23", "lat_24", "long_24"], axis=1)
X_mlp_test["lat_15"] = y_lat_pred_1
X_mlp_test["long_15"] = y_long_pred_1
X_mlp_test["lat_16"] = y_lat_pred_2
X_mlp_test["long_16"] = y_long_pred_2
X_mlp_test["lat_17"] = y_lat_pred_3
X_mlp_test["long_17"] = y_long_pred_3
X_mlp_test.columns = columns
scaler = StandardScaler().fit(X_mlp_test)
X_mlp_test = scaler.transform(X_mlp_test)

y_mlp_lat_test = y_mlp_lat_test_full[["lat_18"]]
y_mlp_long_test = y_mlp_long_test_full[["long_18"]]
y_mlp_lat_test.columns = ["lat_15"]
y_mlp_long_test.columns = ["long_15"]
scaler_lat = StandardScaler().fit(y_mlp_lat_test)
scaler_long = StandardScaler().fit(y_mlp_long_test)
y_mlp_lat_test = scaler_lat.transform(y_mlp_lat_test)
y_mlp_long_test = scaler_long.transform(y_mlp_long_test)

# Predicción 4
y_lat_pred_4 = model_mlp_lat.predict(X_mlp_test)
y_lat_pred_4 = scaler_lat.inverse_transform(y_lat_pred_4)
y_lat_real_4 = scaler_lat.inverse_transform(y_mlp_lat_test)

y_long_pred_4 = model_mlp_long.predict(X_mlp_test)
y_long_pred_4 = scaler_long.inverse_transform(y_long_pred_4)
y_long_real_4 = scaler_long.inverse_transform(y_mlp_long_test)

############## FIFTH ITERATION ##############
# Datos para predicción 5
```

```python
X_mlp_test = X_mlp_test_real.drop(["lat_0", "long_0", "lat_1", "long_1",
 ↪"lat_2", "long_2", "lat_3", "long_3", "lat_15",
                                    "long_15", "lat_16", "long_16", "lat_17",
 ↪"long_17", "lat_18", "long_18", "lat_19", "long_19",
                                    "lat_20", "long_20", "lat_21", "long_21",
 ↪"lat_22", "long_22", "lat_23", "long_23", "lat_24", "long_24"], axis=1)
X_mlp_test["lat_15"] = y_lat_pred_1
X_mlp_test["long_15"] = y_long_pred_1
X_mlp_test["lat_16"] = y_lat_pred_2
X_mlp_test["long_16"] = y_long_pred_2
X_mlp_test["lat_17"] = y_lat_pred_3
X_mlp_test["long_17"] = y_long_pred_3
X_mlp_test["lat_18"] = y_lat_pred_4
X_mlp_test["long_18"] = y_long_pred_4
X_mlp_test.columns = columns
scaler = StandardScaler().fit(X_mlp_test)
X_mlp_test = scaler.transform(X_mlp_test)

y_mlp_lat_test = y_mlp_lat_test_full[["lat_19"]]
y_mlp_long_test = y_mlp_long_test_full[["long_19"]]
y_mlp_lat_test.columns = ["lat_15"]
y_mlp_long_test.columns = ["long_15"]
scaler_lat = StandardScaler().fit(y_mlp_lat_test)
scaler_long = StandardScaler().fit(y_mlp_long_test)
y_mlp_lat_test = scaler_lat.transform(y_mlp_lat_test)
y_mlp_long_test = scaler_long.transform(y_mlp_long_test)

# Predicción 5
y_lat_pred_5 = model_mlp_lat.predict(X_mlp_test)
y_lat_pred_5 = scaler_lat.inverse_transform(y_lat_pred_5)
y_lat_real_5 = scaler_lat.inverse_transform(y_mlp_lat_test)

y_long_pred_5 = model_mlp_long.predict(X_mlp_test)
y_long_pred_5 = scaler_long.inverse_transform(y_long_pred_5)
y_long_real_5 = scaler_long.inverse_transform(y_mlp_long_test)

############## SIXTH ITERATION ##############
# Datos para predicción 6
X_mlp_test = X_mlp_test_real.drop(["lat_0", "long_0", "lat_1", "long_1",
 ↪"lat_2", "long_2", "lat_3", "long_3", "lat_4", "long_4",
                                    "lat_15", "long_15", "lat_16", "long_16",
 ↪"lat_17", "long_17", "lat_18", "long_18", "lat_19",
                                    "long_19", "lat_20", "long_20", "lat_21",
 ↪"long_21", "lat_22", "long_22", "lat_23",
                                    "long_23", "lat_24", "long_24"], axis=1)
X_mlp_test["lat_15"] = y_lat_pred_1
```

```python
X_mlp_test["long_15"] = y_long_pred_1
X_mlp_test["lat_16"] = y_lat_pred_2
X_mlp_test["long_16"] = y_long_pred_2
X_mlp_test["lat_17"] = y_lat_pred_3
X_mlp_test["long_17"] = y_long_pred_3
X_mlp_test["lat_18"] = y_lat_pred_4
X_mlp_test["long_18"] = y_long_pred_4
X_mlp_test["lat_19"] = y_lat_pred_5
X_mlp_test["long_19"] = y_long_pred_5

X_mlp_test.columns = columns
scaler = StandardScaler().fit(X_mlp_test)
X_mlp_test = scaler.transform(X_mlp_test)

y_mlp_lat_test = y_mlp_lat_test_full[["lat_20"]]
y_mlp_long_test = y_mlp_long_test_full[["long_20"]]
y_mlp_lat_test.columns = ["lat_15"]
y_mlp_long_test.columns = ["long_15"]
scaler_lat = StandardScaler().fit(y_mlp_lat_test)
scaler_long = StandardScaler().fit(y_mlp_long_test)
y_mlp_lat_test = scaler_lat.transform(y_mlp_lat_test)
y_mlp_long_test = scaler_long.transform(y_mlp_long_test)

# Predicción 6
y_lat_pred_6 = model_mlp_lat.predict(X_mlp_test)
y_lat_pred_6 = scaler_lat.inverse_transform(y_lat_pred_6)
y_lat_real_6 = scaler_lat.inverse_transform(y_mlp_lat_test)

y_long_pred_6 = model_mlp_long.predict(X_mlp_test)
y_long_pred_6 = scaler_long.inverse_transform(y_long_pred_6)
y_long_real_6 = scaler_long.inverse_transform(y_mlp_long_test)

############## SEVENTH ITERATION ##############
# Datos para predicción 7
X_mlp_test = X_mlp_test_real.drop(["lat_0", "long_0", "lat_1", "long_1",
 ↪"lat_2", "long_2", "lat_3", "long_3", "lat_4",
                                   "long_4", "lat_5", "long_5",
                                   "lat_15", "long_15", "lat_16", "long_16",
 ↪"lat_17", "long_17", "lat_18", "long_18", "lat_19",
                                   "long_19", "lat_20", "long_20", "lat_21",
 ↪"long_21", "lat_22", "long_22", "lat_23",
                                   "long_23", "lat_24", "long_24"], axis=1)
X_mlp_test["lat_15"] = y_lat_pred_1
X_mlp_test["long_15"] = y_long_pred_1
X_mlp_test["lat_16"] = y_lat_pred_2
X_mlp_test["long_16"] = y_long_pred_2
X_mlp_test["lat_17"] = y_lat_pred_3
```

```python
X_mlp_test["long_17"] = y_long_pred_3
X_mlp_test["lat_18"] = y_lat_pred_4
X_mlp_test["long_18"] = y_long_pred_4
X_mlp_test["lat_19"] = y_lat_pred_5
X_mlp_test["long_19"] = y_long_pred_5
X_mlp_test["lat_20"] = y_lat_pred_6
X_mlp_test["long_20"] = y_long_pred_6

X_mlp_test.columns = columns
scaler = StandardScaler().fit(X_mlp_test)
X_mlp_test = scaler.transform(X_mlp_test)

y_mlp_lat_test = y_mlp_lat_test_full[["lat_21"]]
y_mlp_long_test = y_mlp_long_test_full[["long_21"]]
y_mlp_lat_test.columns = ["lat_15"]
y_mlp_long_test.columns = ["long_15"]
scaler_lat = StandardScaler().fit(y_mlp_lat_test)
scaler_long = StandardScaler().fit(y_mlp_long_test)
y_mlp_lat_test = scaler_lat.transform(y_mlp_lat_test)
y_mlp_long_test = scaler_long.transform(y_mlp_long_test)

# Predicción 7
y_lat_pred_7 = model_mlp_lat.predict(X_mlp_test)
y_lat_pred_7 = scaler_lat.inverse_transform(y_lat_pred_7)
y_lat_real_7 = scaler_lat.inverse_transform(y_mlp_lat_test)

y_long_pred_7 = model_mlp_long.predict(X_mlp_test)
y_long_pred_7 = scaler_long.inverse_transform(y_long_pred_7)
y_long_real_7 = scaler_long.inverse_transform(y_mlp_long_test)

############## EIGHTH ITERATION ##############
# Datos para predicción 8
X_mlp_test = X_mlp_test_real.drop(["lat_0", "long_0", "lat_1", "long_1",
 ↪"lat_2", "long_2", "lat_3", "long_3", "lat_4",
                                   "long_4", "lat_5", "long_5", "lat_6",
 ↪"long_6",
                                   "lat_15", "long_15", "lat_16", "long_16",
 ↪"lat_17", "long_17", "lat_18", "long_18", "lat_19",
                                   "long_19", "lat_20", "long_20", "lat_21",
 ↪"long_21", "lat_22", "long_22", "lat_23",
                                   "long_23", "lat_24", "long_24"], axis=1)
X_mlp_test["lat_15"] = y_lat_pred_1
X_mlp_test["long_15"] = y_long_pred_1
X_mlp_test["lat_16"] = y_lat_pred_2
X_mlp_test["long_16"] = y_long_pred_2
X_mlp_test["lat_17"] = y_lat_pred_3
X_mlp_test["long_17"] = y_long_pred_3
```

```python
X_mlp_test["lat_18"] = y_lat_pred_4
X_mlp_test["long_18"] = y_long_pred_4
X_mlp_test["lat_19"] = y_lat_pred_5
X_mlp_test["long_19"] = y_long_pred_5
X_mlp_test["lat_20"] = y_lat_pred_6
X_mlp_test["long_20"] = y_long_pred_6
X_mlp_test["lat_21"] = y_lat_pred_7
X_mlp_test["long_21"] = y_long_pred_7


X_mlp_test.columns = columns
scaler = StandardScaler().fit(X_mlp_test)
X_mlp_test = scaler.transform(X_mlp_test)

y_mlp_lat_test = y_mlp_lat_test_full[["lat_22"]]
y_mlp_long_test = y_mlp_long_test_full[["long_22"]]
y_mlp_lat_test.columns = ["lat_15"]
y_mlp_long_test.columns = ["long_15"]
scaler_lat = StandardScaler().fit(y_mlp_lat_test)
scaler_long = StandardScaler().fit(y_mlp_long_test)
y_mlp_lat_test = scaler_lat.transform(y_mlp_lat_test)
y_mlp_long_test = scaler_long.transform(y_mlp_long_test)

# Predicción 8
y_lat_pred_8 = model_mlp_lat.predict(X_mlp_test)
y_lat_pred_8 = scaler_lat.inverse_transform(y_lat_pred_8)
y_lat_real_8 = scaler_lat.inverse_transform(y_mlp_lat_test)

y_long_pred_8 = model_mlp_long.predict(X_mlp_test)
y_long_pred_8 = scaler_long.inverse_transform(y_long_pred_8)
y_long_real_8 = scaler_long.inverse_transform(y_mlp_long_test)

############## NINTH ITERATION ##############
# Datos para predicción 9
X_mlp_test = X_mlp_test_real.drop(["lat_0", "long_0", "lat_1", "long_1",
 ↪"lat_2", "long_2", "lat_3", "long_3", "lat_4",
                                    "long_4", "lat_5", "long_5", "lat_6",
 ↪"long_6", "lat_7", "long_7",
                                    "lat_15", "long_15", "lat_16", "long_16",
 ↪"lat_17", "long_17", "lat_18", "long_18", "lat_19",
                                    "long_19", "lat_20", "long_20", "lat_21",
 ↪"long_21", "lat_22", "long_22", "lat_23",
                                    "long_23", "lat_24", "long_24"], axis=1)
X_mlp_test["lat_15"] = y_lat_pred_1
X_mlp_test["long_15"] = y_long_pred_1
X_mlp_test["lat_16"] = y_lat_pred_2
X_mlp_test["long_16"] = y_long_pred_2
```

```python
X_mlp_test["lat_17"] = y_lat_pred_3
X_mlp_test["long_17"] = y_long_pred_3
X_mlp_test["lat_18"] = y_lat_pred_4
X_mlp_test["long_18"] = y_long_pred_4
X_mlp_test["lat_19"] = y_lat_pred_5
X_mlp_test["long_19"] = y_long_pred_5
X_mlp_test["lat_20"] = y_lat_pred_6
X_mlp_test["long_20"] = y_long_pred_6
X_mlp_test["lat_21"] = y_lat_pred_7
X_mlp_test["long_21"] = y_long_pred_7
X_mlp_test["lat_22"] = y_lat_pred_8
X_mlp_test["long_22"] = y_long_pred_8


X_mlp_test.columns = columns
scaler = StandardScaler().fit(X_mlp_test)
X_mlp_test = scaler.transform(X_mlp_test)

y_mlp_lat_test = y_mlp_lat_test_full[["lat_23"]]
y_mlp_long_test = y_mlp_long_test_full[["long_23"]]
y_mlp_lat_test.columns = ["lat_15"]
y_mlp_long_test.columns = ["long_15"]
scaler_lat = StandardScaler().fit(y_mlp_lat_test)
scaler_long = StandardScaler().fit(y_mlp_long_test)
y_mlp_lat_test = scaler_lat.transform(y_mlp_lat_test)
y_mlp_long_test = scaler_long.transform(y_mlp_long_test)

# Predicción 9
y_lat_pred_9 = model_mlp_lat.predict(X_mlp_test)
y_lat_pred_9 = scaler_lat.inverse_transform(y_lat_pred_9)
y_lat_real_9 = scaler_lat.inverse_transform(y_mlp_lat_test)

y_long_pred_9 = model_mlp_long.predict(X_mlp_test)
y_long_pred_9  = scaler_long.inverse_transform(y_long_pred_9)
y_long_real_9 = scaler_long.inverse_transform(y_mlp_long_test)

############## TENTH ITERATION ##############
# Datos para predicción 10
X_mlp_test = X_mlp_test_real.drop(["lat_0", "long_0", "lat_1", "long_1",
 ↪"lat_2", "long_2", "lat_3", "long_3", "lat_4",
                                    "long_4", "lat_5", "long_5", "lat_6",
 ↪"long_6", "lat_7", "long_7", "lat_8", "long_8",
                                    "lat_15", "long_15", "lat_16", "long_16",
 ↪"lat_17", "long_17", "lat_18", "long_18", "lat_19",
                                    "long_19", "lat_20", "long_20", "lat_21",
 ↪"long_21", "lat_22", "long_22", "lat_23",
                                    "long_23", "lat_24", "long_24"], axis=1)
```

```python
X_mlp_test["lat_15"] = y_lat_pred_1
X_mlp_test["long_15"] = y_long_pred_1
X_mlp_test["lat_16"] = y_lat_pred_2
X_mlp_test["long_16"] = y_long_pred_2
X_mlp_test["lat_17"] = y_lat_pred_3
X_mlp_test["long_17"] = y_long_pred_3
X_mlp_test["lat_18"] = y_lat_pred_4
X_mlp_test["long_18"] = y_long_pred_4
X_mlp_test["lat_19"] = y_lat_pred_5
X_mlp_test["long_19"] = y_long_pred_5
X_mlp_test["lat_20"] = y_lat_pred_6
X_mlp_test["long_20"] = y_long_pred_6
X_mlp_test["lat_21"] = y_lat_pred_7
X_mlp_test["long_21"] = y_long_pred_7
X_mlp_test["lat_22"] = y_lat_pred_8
X_mlp_test["long_22"] = y_long_pred_8
X_mlp_test["lat_23"] = y_lat_pred_9
X_mlp_test["long_23"] = y_long_pred_9

print(X_mlp_test)
print(len(columns))

X_mlp_test.columns = columns
scaler = StandardScaler().fit(X_mlp_test)
X_mlp_test = scaler.transform(X_mlp_test)

y_mlp_lat_test = y_mlp_lat_test_full[["lat_24"]]
y_mlp_long_test = y_mlp_long_test_full[["long_24"]]
y_mlp_lat_test.columns = ["lat_15"]
y_mlp_long_test.columns = ["long_15"]
scaler_lat = StandardScaler().fit(y_mlp_lat_test)
scaler_long = StandardScaler().fit(y_mlp_long_test)
y_mlp_lat_test = scaler_lat.transform(y_mlp_lat_test)
y_mlp_long_test = scaler_long.transform(y_mlp_long_test)

# Predicción 10
y_lat_pred_10 = model_mlp_lat.predict(X_mlp_test)
y_lat_pred_10 = scaler_lat.inverse_transform(y_lat_pred_10)
y_lat_real_10 = scaler_lat.inverse_transform(y_mlp_lat_test)

y_long_pred_10 = model_mlp_long.predict(X_mlp_test)
y_long_pred_10 = scaler_long.inverse_transform(y_long_pred_10)
y_long_real_10 = scaler_long.inverse_transform(y_mlp_long_test)


############## Create results dataframe ##############
```

```
df_pred = df_pred.assign(lat_15=y_lat_pred_1, long_15=y_long_pred_1,␣
  ↪lat_16=y_lat_pred_2, long_16=y_long_pred_2,
                         lat_17=y_lat_pred_3, long_17=y_long_pred_3,␣
  ↪lat_18=y_lat_pred_4, long_18=y_long_pred_4,
                         lat_19=y_lat_pred_5, long_19=y_long_pred_5,␣
  ↪lat_20=y_lat_pred_6, long_20=y_long_pred_6,
                         lat_21=y_lat_pred_7, long_21=y_long_pred_7,␣
  ↪lat_22=y_lat_pred_8, long_22=y_long_pred_8,
                         lat_23=y_lat_pred_9, long_23=y_long_pred_9,␣
  ↪lat_24=y_lat_pred_9, long_24=y_long_pred_9,
                         lat_25=y_lat_pred_10, long_25=y_long_pred_10)
```

```
Time 1000 predictions: 0.26798510551452637s
           lat_9    long_9    lat_10   long_10    lat_11   long_11  \
21503   41.391283  2.166294  41.391283  2.166294  41.391283  2.166294
10056   41.390261  2.164762  41.390261  2.164762  41.390264  2.164758
140155  41.393327  2.163455  41.393327  2.163455  41.393327  2.163455
29624   41.389761  2.165417  41.389761  2.165417  41.389761  2.165417
131647  41.392062  2.165118  41.392062  2.165118  41.392062  2.165118
…          …         …         …         …         …         …
110825  41.391315  2.165713  41.391315  2.165713  41.391315  2.165713
5240    41.390291  2.164367  41.390291  2.164367  41.390291  2.164367
7994    41.393144  2.163438  41.393139  2.163443  41.393132  2.163453
47474   41.391030  2.165618  41.391030  2.165618  41.391040  2.165631
94150   41.391654  2.165613  41.391654  2.165613  41.391654  2.165613


           lat_12   long_12    lat_13   long_13  …     lat_19   long_19  \
21503   41.391283  2.166294  41.391283  2.166294  …  41.391342  2.166167
10056   41.390276  2.164742  41.390294  2.164716  …  41.390530  2.164614
140155  41.393327  2.163455  41.393327  2.163455  …  41.393410  2.163493
29624   41.389761  2.165417  41.389761  2.165417  …  41.389915  2.165330
131647  41.392062  2.165118  41.392062  2.165118  …  41.392082  2.164984
…          …         …         …         …      …     …         …
110825  41.391315  2.165713  41.391315  2.165713  …  41.391319  2.165632
5240    41.390291  2.164367  41.390291  2.164367  …  41.390434  2.164494
7994    41.393118  2.163470  41.393113  2.163476  …  41.393208  2.163543
47474   41.391055  2.165652  41.391093  2.165702  …  41.391350  2.165743
94150   41.391654  2.165613  41.391654  2.165613  …  41.391693  2.165528


           lat_20   long_20    lat_21   long_21    lat_22   long_22  \
21503   41.391335  2.166176  41.391327  2.166187  41.391319  2.166199
10056   41.390560  2.164604  41.390587  2.164599  41.390617  2.164588
140155  41.393425  2.163474  41.393440  2.163453  41.393459  2.163434
29624   41.389919  2.165326  41.389923  2.165324  41.389931  2.165320
131647  41.392071  2.164953  41.392059  2.164917  41.392048  2.164875
…          …         …         …         …         …         …
```

```
110825   41.391300   2.165642   41.391281   2.165652   41.391262   2.165663
5240     41.390442   2.164523   41.390446   2.164555   41.390457   2.164585
7994     41.393234   2.163527   41.393257   2.163507   41.393284   2.163489
47474    41.391373   2.165766   41.391396   2.165790   41.391418   2.165813
94150    41.391682   2.165534   41.391666   2.165540   41.391647   2.165547


             lat_23    long_23
21503    41.391312   2.166209
10056    41.390652   2.164580
140155   41.393475   2.163413
29624    41.389938   2.165315
131647   41.392036   2.164831
…            …          …
110825   41.391243   2.165676
5240     41.390465   2.164616
7994     41.393311   2.163473
47474    41.391441   2.165847
94150    41.391624   2.165554

[1000 rows x 30 columns]
30
```

[611]:
```python
len(X_mlp_test[0])
```

[611]: 30

[612]:
```python
avg_distances = []
avg_distance, std_deviation =␣
 ↪get_avg_distance(y_lat_pred_1,y_long_pred_1,y_lat_real_1,y_long_real_1)
avg_distances.append(avg_distance)
print(f"Average distance 1: {avg_distance}")
avg_distance, std_deviation =␣
 ↪get_avg_distance(y_lat_pred_2,y_long_pred_2,y_lat_real_2,y_long_real_2)
avg_distances.append(avg_distance)
print(f"Average distance 2: {avg_distance}")
avg_distance, std_deviation =␣
 ↪get_avg_distance(y_lat_pred_3,y_long_pred_3,y_lat_real_3,y_long_real_3)
avg_distances.append(avg_distance)
print(f"Average distance 3: {avg_distance}")
avg_distance, std_deviation =␣
 ↪get_avg_distance(y_lat_pred_4,y_long_pred_4,y_lat_real_4,y_long_real_4)
avg_distances.append(avg_distance)
print(f"Average distance 4: {avg_distance}")
avg_distance, std_deviation =␣
 ↪get_avg_distance(y_lat_pred_5,y_long_pred_5,y_lat_real_5,y_long_real_5)
avg_distances.append(avg_distance)
print(f"Average distance 5: {avg_distance}")
```

```python
avg_distance, std_deviation =␣
 ↪get_avg_distance(y_lat_pred_6,y_long_pred_6,y_lat_real_6,y_long_real_6)
avg_distances.append(avg_distance)
print(f"Average distance 6: {avg_distance}")
avg_distance, std_deviation =␣
 ↪get_avg_distance(y_lat_pred_7,y_long_pred_7,y_lat_real_7,y_long_real_7)
avg_distances.append(avg_distance)
print(f"Average distance 7: {avg_distance}")
avg_distance, std_deviation =␣
 ↪get_avg_distance(y_lat_pred_8,y_long_pred_8,y_lat_real_8,y_long_real_8)
avg_distances.append(avg_distance)
print(f"Average distance 8: {avg_distance}")
avg_distance, std_deviation =␣
 ↪get_avg_distance(y_lat_pred_9,y_long_pred_9,y_lat_real_9,y_long_real_9)
avg_distances.append(avg_distance)
print(f"Average distance 9: {avg_distance}")
avg_distance, std_deviation =␣
 ↪get_avg_distance(y_lat_pred_10,y_long_pred_10,y_lat_real_10,y_long_real_10)
avg_distances.append(avg_distance)
print(f"Average distance 10: {avg_distance}")
plot_distance_errors(avg_distances)
```

```
Average distance 1: 5.067601507336725
Average distance 2: 15.25185498477966
Average distance 3: 15.926260271347303
Average distance 4: 16.80802075214277
Average distance 5: 18.175948149957375
Average distance 6: 19.963563576409662
Average distance 7: 22.234883446054578
Average distance 8: 24.96743614393459
Average distance 9: 28.016318248081156
Average distance 10: 31.467334734957703
```

Error en la predicción de posiciones
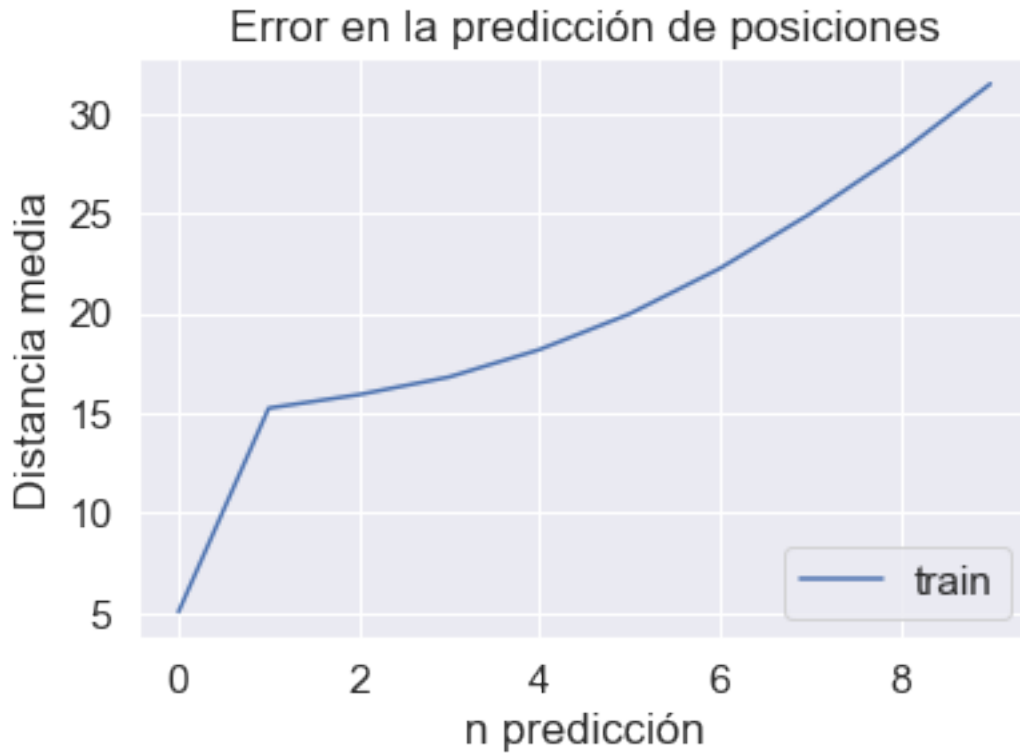
3 Colisiones

3.1 Funciones

3.1.1 Zona de colisión

```
[613]: def get_rectangle_bounds(coordinates, width, length):
           start = geopy.Point(coordinates)
           hypotenuse = math.hypot(width/1000, length/1000)

           # Edit used wrong formula to convert radians to degrees, use math builtin
       →function
           northeast_angle = 0 - math.degrees(math.atan(width/length))
           southwest_angle = 180 - math.degrees(math.atan(width/length))

           d = geopy.distance.distance(kilometers=hypotenuse/2)
           northeast = d.destination(point=start, bearing=northeast_angle)
           southwest = d.destination(point=start, bearing=southwest_angle)
           bounds = []
           for point in [northeast, southwest]:
               coords = (point.latitude, point.longitude)
               bounds.append(coords)
```

```
    return bounds
```

3.1.2 Bearing

```
[614]: def bear(coords1, coords2):
           lat1 = coords1[0]
           lon1 = coords1[1]
           lat2 = coords2[0]
           lon2 = coords2[1]

           rlat1 = math.radians(lat1)
           rlon1 = math.radians(lon1)
           rlat2 = math.radians(lat2)
           rlon2 = math.radians(lon2)
           dlon = math.radians(lon2-lon1)
           b = math.atan2(math.sin(dlon)*math.cos(rlat2),math.cos(rlat1)*math.
        ↪sin(rlat2)-math.sin(rlat1)*math.cos(rlat2)*math.cos(dlon))
           bd = math.degrees(b)
           br,bn = divmod(bd+360,360) # the bearing remainder and final bearing
           return bn
```

3.1.3 Rotating points

```
[615]: # To get a rotated rectangle at a bearing, you need to get the points of the␣
        ↪the recatangle at that bearing
       def get_rotated_points(coordinates, bearing, width, length):
           start = geopy.Point(coordinates)
           width  = width/1000
           length = length/1000
           rectlength = geopy.distance.distance(kilometers=length)
           rectwidth = geopy.distance.distance(kilometers=width)
           halfwidth = geopy.distance.distance(kilometers=width/2)
           halflength = geopy.distance.distance(kilometers=length/2)

           pointAB = halflength.destination(point=start, bearing=bearing)
           pointA = halfwidth.destination(point=pointAB, bearing=0-bearing)
           pointB = rectwidth.destination(point=pointA, bearing=180-bearing)
           pointC = rectlength.destination(point=pointB, bearing=bearing-180)
           pointD = rectwidth.destination(point=pointC, bearing=0-bearing)

           points = []
           for point in [pointA, pointB, pointC, pointD]:
               coords = (point.latitude, point.longitude)
               points.append(coords)

           return points
```

3.1.4 Colisión

```python
[616]: def get_colision_risk(distances, avg_distances=avg_distances):
           min_distance = min(distances)
           time = distances.index(min_distance)
           mean_error = 2*avg_distances[time]
           dif = min_distance-mean_error
           if dif >= 0:
               return 0, time
           risk = max(abs(dif), 0)/mean_error
           return risk, time
```

```python
[617]: def is_colision(vehicle1, vehicle2, init_pred=15, total_preds=5, length = 4,
       ↪width = 1.50, avg_distances=avg_distances):
           distances = []
           for i in range(init_pred,init_pred+total_preds):
               lat1 = vehicle1["lat_"+str(i)]
               lon1 = vehicle1["long_"+str(i)]
               lat2 = vehicle2["lat_"+str(i)]
               lon2 = vehicle2["long_"+str(i)]

               lat1_prev = vehicle1["lat_"+str(i-1)]
               lon1_prev = vehicle1["long_"+str(i-1)]
               lat2_prev = vehicle2["lat_"+str(i-1)]
               lon2_prev = vehicle2["long_"+str(i-1)]

               start_coords1 = [lat1_prev, lon1_prev]
               start_coords2 = [lat2_prev, lon2_prev]

               next_coords1 = [lat1, lon1]
               next_coords2 = [lat2, lon2]

               bearing1 = bear(start_coords1, next_coords1)
               bearing2 = bear(start_coords2, next_coords2)

               #bounds1 = get_rectangle_bounds(tuple(start_coords1),width, length)
               #bounds2 = get_rectangle_bounds(tuple(start_coords2),width, length)

               points1 = get_rotated_points(tuple(start_coords1), bearing1, width,
       ↪length)
               points2 = get_rotated_points(tuple(start_coords2), bearing2, width,
       ↪length)

               polygon1 = geometry.Polygon(points1)
               polygon2 = geometry.Polygon(points2)

               if i==init_pred:
```

```
            map = folium.Map([(start_coords1[0]+start_coords2[0])/2,
    ↪(start_coords1[1]+start_coords2[1])/2], zoom_start=20)
          folium.Polygon(points1, color='red').add_to(map)
          folium.Polygon(points2, color='green').add_to(map)

          if (polygon1.intersection(polygon2).area > 0.0):
              return True, map, 1, i-init_pred

          else:
              distances.append(hs.haversine(( next_coords1 ),( next_coords2 ),
    ↪unit=Unit.METERS))

      collision_risk, time = get_colision_risk(distances,
    ↪avg_distances=avg_distances)
      return False, map, collision_risk, time
```

```
[618]:  # Probabilidades de colisión del vehículo 80 con los otros vehículos
        for i in range(100):
            vehicle1 = df_pred.iloc[80]
            vehicle2 = df_pred.iloc[i]
            colision, map, collision_risk, time = is_colision(vehicle1, vehicle2)
            print(i, collision_risk)
```

```
0 0
1 0.32214338618534627
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 0
11 0
12 0
13 0
14 0
15 0
16 0
17 0
18 0
19 0
20 0
21 0
22 0
23 0
```

```
24 0
25 0
26 0
27 0
28 0
29 0
30 0
31 0
32 0
33 0
34 0
35 0
36 0
37 0
38 0
39 0
40 0
41 0
42 0
43 0
44 0
45 0
46 0
47 0
48 0
49 0
50 0
51 0
52 0
53 0
54 0
55 0
56 0
57 0
58 0
59 0
60 0
61 0
62 0
63 0
64 0
65 0
66 0
67 0
68 0
69 0
70 0
71 0
```

```
72  0
73  0
74  0
75  0
76  0
77  0
78  0
79  0
80  1
81  0
82  0
83  0
84  0
85  0
86  0
87  0
88  0
89  0
90  0
91  0
92  0
93  0
94  0
95  0
96  0
97  0
98  0
99  0
```

[619]:
```python
vehicle1 = X_mlp_test_full.iloc[80]
vehicle2 = X_mlp_test_full.iloc[11]
colision, map, collision_risk, time = is_colision(vehicle1, vehicle2)
print(colision, collision_risk, time)
```

False 0 4

[620]:
```python
map
```

[620]: `<folium.folium.Map at 0x2e0c267b0a0>`

[621]:
```python
vehicle1 = df_pred.iloc[80]
vehicle2 = df_pred.iloc[11]
colision, map, collision_risk, time = is_colision(vehicle1, vehicle2)
print(colision, collision_risk, time)
```

False 0 4

[622]:
```python
map
```

```
[622]: <folium.folium.Map at 0x2e0c0384d30>
```

4 Predicción de colisiones

4.0.1 Preparación de datos

```python
[260]: def format_dataset_collisions(df, initial_time, n):

           final_time = initial_time+n

           df = df[(df["time"]>=initial_time) & (df["time"]<final_time)]

           df = df.sort_values(['vehicle_id', 'time'])
           df = df.reset_index(drop=True)

           data = []
           temp_data = []
           temp_i = 0
           temp_vehicle = df.loc[0, 'vehicle_id']

           for i in range(1, df.shape[0]):
               row = df.loc[i]

               if row["vehicle_id"] != temp_vehicle:
                   temp_vehicle = row["vehicle_id"]
                   temp_i = 0
                   temp_data = []
                   temp_data.append(row["vehicle_id"])
                   temp_data.append(row["latitude"])
                   temp_data.append(row["longitude"])
                   temp_data.append(row["collision"])
                   temp_data.append(row["victim_id"])

               elif temp_i < n-1:
                   temp_data.append(row["latitude"])
                   temp_data.append(row["longitude"])
                   temp_data.append(row["collision"])
                   temp_data.append(row["victim_id"])

               else:
                   temp_data.append(row["latitude"])
                   temp_data.append(row["longitude"])
                   temp_data.append(row["collision"])
                   temp_data.append(row["victim_id"])
                   data.append(temp_data)
                   temp_data = []
                   temp_i = 0
```

```
                temp_vehicle = -1
            temp_i += 1


    return data

#mlp_data = format_dataset_collisions(df, 0, 25)
```

```
[261]: columns  = ["vehicle_id", "lat_0", "long_0", "collision_0",  "victim_id_0",
                    "lat_1", "long_1", "collision_1", "victim_id_1", "lat_2", "long_2",␣
          ↪"collision_2", "victim_id_2",
                    "lat_3", "long_3", "collision_3",  "victim_id_3", "lat_4",␣
          ↪"long_4", "collision_4",  "victim_id_4",
                    "lat_5", "long_5", "collision_5",  "victim_id_5", "lat_6",␣
          ↪"long_6", "collision_6",  "victim_id_6",
                    "lat_7", "long_7", "collision_7",  "victim_id_7", "lat_8",␣
          ↪"long_8", "collision_8",  "victim_id_8",
                    "lat_9", "long_9", "collision_9",  "victim_id_9", "lat_10",␣
          ↪"long_10", "collision_10",  "victim_id_10",
                    "lat_11", "long_11", "collision_11",  "victim_id_11", "lat_12",␣
          ↪"long_12", "collision_12",  "victim_id_12",
                    "lat_13", "long_13", "collision_13",  "victim_id_13", "lat_14",␣
          ↪"long_14", "collision_14",  "victim_id_14",
                    "lat_15", "long_15", "collision_15",  "victim_id_15", "lat_16",␣
          ↪"long_16", "collision_16",  "victim_id_16",
                    "lat_17", "long_17", "collision_17",  "victim_id_17", "lat_18",␣
          ↪"long_18", "collision_18",  "victim_id_18",
                    "lat_19", "long_19", "collision_19",  "victim_id_19", "lat_20",␣
          ↪"long_20", "collision_20",  "victim_id_20",
                    "lat_21", "long_21", "collision_21",  "victim_id_21", "lat_22",␣
          ↪"long_22", "collision_22",  "victim_id_22",
                    "lat_23", "long_23", "collision_23",  "victim_id_23", "lat_24",␣
          ↪"long_24", "collision_24",  "victim_id_24"]

       for i in range(8, 10008, 25):
           col_data = format_dataset_collisions(df, i, 25)
           df_col = pd.DataFrame(col_data, columns=columns )
           df_col.to_csv(f"collision_dfs/df_collisions_{i}_{i+25}.csv")
```

```
[262]: #pd.read_csv("collision_dfs/df_collisions_100_125.csv")
```

```
[263]: df_names = []
       df_full_names = []
       for i in range(8, 10008, 25):
           df_names.append(f"collision_dfs/df_collisions_{i}_{i+25}.csv"[14:-4])
           df_full_names.append(f"collision_dfs/df_collisions_{i}_{i+25}.csv")
```

```
collision_dfs = {}
for i in range(len(df_names)):
    collision_dfs[df_names[i]] = pd.read_csv(df_full_names[i]).drop(["Unnamed:
↪0"], axis=1)
```

[264]:
```
pred_columns  = ["lat_0", "long_0", "lat_1", "long_1", "lat_2", "long_2",
↪"lat_3", "long_3", "lat_4", "long_4",
                "lat_5", "long_5", "lat_6", "long_6", "lat_7", "long_7", "lat_8",
↪"long_8", "lat_9", "long_9",
                "lat_10", "long_10", "lat_11", "long_11", "lat_12", "long_12",
↪"lat_13", "long_13", "lat_14", "long_14",
                "lat_15", "long_15", "lat_16", "long_16", "lat_17", "long_17",
↪"lat_18", "long_18", "lat_19", "long_19",
                "lat_20", "long_20", "lat_21", "long_21", "lat_22", "long_22",
↪"lat_23", "long_23", "lat_24", "long_24"]

real_columns  = ["lat_0", "long_0", "lat_1", "long_1", "lat_2", "long_2",
↪"lat_3", "long_3", "lat_4", "long_4",
                "lat_5", "long_5", "lat_6", "long_6", "lat_7", "long_7", "lat_8",
↪"long_8", "lat_9", "long_9",
                "lat_10", "long_10", "lat_11", "long_11", "lat_12", "long_12",
↪"lat_13", "long_13", "lat_14", "long_14"]
```

[265]:
```
dfs_with_collisions = []
for key in collision_dfs.keys():
    df_col = collision_dfs[key]
    for i in range(15,25):
        hay_colision = len(df_col[df_col['collision_'+str(i)]==1])
        if(hay_colision)>0:
            dfs_with_collisions.append(key)
            print(key)
```

```
df_collisions_1033_1058
df_collisions_1508_1533
df_collisions_2233_2258
df_collisions_2558_2583
df_collisions_3433_3458
df_collisions_4133_4158
df_collisions_4333_4358
df_collisions_4833_4858
df_collisions_5058_5083
df_collisions_5133_5158
df_collisions_5158_5183
df_collisions_5408_5433
df_collisions_6258_6283
df_collisions_6758_6783
```

```
df_collisions_7583_7608
df_collisions_8183_8208
df_collisions_8783_8808
df_collisions_9258_9283
df_collisions_9858_9883
df_collisions_9908_9933
```

```
[266]: df_colls = {k: collision_dfs[k] for k in dfs_with_collisions}
       df_colls_pos = {k: collision_dfs[k][pred_columns] for k in dfs_with_collisions}
```

4.0.2 Transformar para regresión

```
[297]: def format_dataset_regresion_collisions(df, n):

           data = []
           temp_data = []

           for i in range(0, df.shape[0]):
               row = df.loc[i]
               temp_data = []
               for i in range(n):
                   temp_data.append([row['lat_'+str(i)], row['long_'+str(i)]])
               data.append(temp_data)
           data = np.array(data, dtype='float64')
           return data

       for key in df_colls_pos.keys():
           data_colls = format_dataset_regresion_collisions(df_colls[key], 25)
           np.save(file=f"reg_coll_data/{key}.npy", arr=data_colls)
```

```
[300]: data_colls ={}
       for key in df_colls_pos.keys():
           data_colls[key] = np.load(f"reg_coll_data/{key}.npy")
```

4.0.3 Predecir trayectorias

```
[301]: def call_get_next_10_positions_res(dataset, model):
           df_colls_res = pd.DataFrame(columns=pred_columns)
           array_colls_res = np.empty((0,50))
           array_colls_res = []
           for data in dataset:
               prediction = get_next_10_positions(data, model)
               array_colls_res.append(prediction)
           df_colls_res = pd.DataFrame(array_colls_res, columns=pred_columns)
           return df_colls_res
```

```python
[302]: def get_next_10_positions(data, model):

           df_res = data[0:15]

           # Prediction 1
           X = data[0:15]
           y = data[1:16]
           model.fit(X, y)
           pred_1 = model.predict([y[-1]])
           real_1 = data[16]

           # Prediction 2
           X = np.concatenate((data[1:15], pred_1))
           y = np.concatenate((data[2:15], pred_1, [real_1]))
           model.fit(X, y)
           pred_2 = model.predict([y[-1]])
           real_2 = data[17]

           # Prediction 3
           X = np.concatenate((data[2:15], pred_1, pred_2))
           y = np.concatenate((data[3:15], pred_1, pred_2, [real_2]))
           model.fit(X, y)
           pred_3 = model.predict([y[-1]])
           real_3 = data[18]

           # Prediction 4
           X = np.concatenate((data[3:15], pred_1, pred_2, pred_3))
           y = np.concatenate((data[4:15], pred_1, pred_2, pred_3, [real_3]))
           model.fit(X, y)
           pred_4 = model.predict([y[-1]])
           real_4 = data[19]

           # Prediction 5
           X = np.concatenate((data[4:15], pred_1, pred_2, pred_3, pred_4))
           y = np.concatenate((data[5:15], pred_1, pred_2, pred_3, pred_4, [real_4]))
           model.fit(X, y)
           pred_5 = model.predict([y[-1]])
           real_5 = data[20]

           # Prediction 6
           X = np.concatenate((data[5:15], pred_1, pred_2, pred_3, pred_4, pred_5))
           y = np.concatenate((data[6:15], pred_1, pred_2, pred_3, pred_4, pred_5,
       [real_5]))
           model.fit(X, y)
           pred_6 = model.predict([y[-1]])
           real_6 = data[21]
```

```python
    # Prediction 7
    X = np.concatenate((data[6:15], pred_1, pred_2, pred_3, pred_4, pred_5,
↪pred_6))
    y = np.concatenate((data[7:15], pred_1, pred_2, pred_3, pred_4, pred_5,
↪pred_6, [real_6]))
    model.fit(X, y)
    pred_7 = model.predict([y[-1]])
    real_7 = data[22]


    # Prediction 8
    X = np.concatenate((data[7:15], pred_1, pred_2, pred_3, pred_4, pred_5,
↪pred_6, pred_7))
    y = np.concatenate((data[8:15], pred_1, pred_2, pred_3, pred_4, pred_5,
↪pred_6, pred_7, [real_7]))
    model.fit(X, y)
    pred_8 = model.predict([y[-1]])
    real_8 = data[23]


    # Prediction 9
    X = np.concatenate((data[8:15], pred_1, pred_2, pred_3, pred_4, pred_5,
↪pred_6, pred_7, pred_8))
    y = np.concatenate((data[9:15], pred_1, pred_2, pred_3, pred_4, pred_5,
↪pred_6, pred_7, pred_8, [real_8]))
    model.fit(X, y)
    pred_9 = model.predict([y[-1]])
    real_9 = data[24]


    # Prediction 10
    X = np.concatenate((data[9:15], pred_1, pred_2, pred_3, pred_4, pred_5,
↪pred_6, pred_7, pred_8, pred_9))
    y = np.concatenate((data[10:15], pred_1, pred_2, pred_3, pred_4, pred_5,
↪pred_6, pred_7, pred_8, pred_9, [real_9]))
    model.fit(X, y)
    pred_10 = model.predict([y[-1]])
    real_10 = pred_10


    #predictions = [l.tolist() for l in
↪[pred_1[0],pred_2[0],pred_3[0],pred_4[0],pred_5[0],pred_6[0],pred_7[0],pred_8[0],pred_9[0],
    #real_values = [l.tolist() for l in
↪[real_1,real_2,real_3,real_4,real_5,real_6,real_7,real_8,real_9,real_10]]


    df_res = np.append(df_res, [pred_1, pred_2, pred_3, pred_4, pred_5, pred_6,
↪pred_7, pred_8, pred_9, pred_10])


    return df_res
```

```
[442]: '''from sklearn.tree import ExtraTreeRegressor
       df_predictions ={}
       for key in data_colls.keys():
           model = ExtraTreeRegressor()
           df_predictions[key] = call_get_next_10_positions_res(data_colls[key],␣
       ↪model)'''
```

```
[425]: '''from sklearn.gaussian_process import GaussianProcessRegressor
       df_predictions ={}
       for key in data_colls.keys():
           model = GaussianProcessRegressor()
           df_predictions[key] = call_get_next_10_positions_res(data_colls[key],␣
       ↪model)'''
```

4.0.4 Lista de colisiones

```
[443]: real_collisions_df = pd.DataFrame(columns=columns + ['df_time'])
       for key in df_colls.keys():
           curr_df = df_colls[key]
           for i in range(15, 25):
               real_collisions = curr_df[curr_df['collision_'+str(i)]==1]
               real_collisions['df_time'] = key
               real_collisions_df = pd.concat([real_collisions_df, real_collisions])
```

```
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
```

```
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
```

```
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
```

```
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_7940\3475772495.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
```

[444]: `real_collisions_df`

[444]:
```
     vehicle_id      lat_0     long_0  collision_0  victim_id_0      lat_1  \
83       1053.0  41.391101   2.165712          0.0         -1.0  41.391101
170      1537.0  41.391258   2.163362          0.0         -1.0  41.391306
167      2159.0  41.388898   2.162727          0.0         -1.0  41.388941
```

```
101     2322.0  41.391561   2.164133          0.0          -1.0  41.391561
51      2927.0  41.392079   2.165081          0.0          -1.0  41.392108
180     3825.0  41.390532   2.164952          0.0          -1.0  41.390551
162     3977.0  41.393194   2.163621          0.0          -1.0  41.393194
27      3985.0  41.391911    2.1654           0.0          -1.0  41.391955
170     4659.0  41.393364   2.163408          0.0          -1.0  41.393364
106     4600.0  41.391832   2.165496          0.0          -1.0  41.391832
147     4699.0  41.390261   2.164762          0.0          -1.0  41.390261
163     4943.0  41.393078   2.166203          0.0          -1.0  41.393031
166     5562.0  41.389343   2.162701          0.0          -1.0  41.389272
156     5856.0  41.393017   2.163595          0.0          -1.0  41.393017
156     6502.0  41.391673   2.161901          0.0          -1.0  41.391716
102     6956.0   41.39103    2.16368          0.0          -1.0  41.391057
171     7462.0  41.388772   2.162561          0.0          -1.0  41.388823
60      7671.0  41.391587   2.165538          0.0          -1.0  41.391587
53      8150.0  41.391145   2.166466          0.0          -1.0  41.391162
30      7951.0  41.391474   2.165784          0.0          -1.0  41.391553

        long_1 collision_1 victim_id_1      lat_2 … victim_id_22      lat_23  \
83    2.165712         0.0        -1.0  41.391101 …         -1.0  41.392022
170   2.163291         0.0        -1.0  41.391353 …         -1.0  41.392811
167   2.162785         0.0        -1.0  41.388985 …         -1.0  41.389914
101   2.164133         0.0        -1.0  41.391561 …         -1.0  41.391358
51    2.165018         0.0        -1.0  41.392103 …         -1.0   41.39142
180   2.164978         0.0        -1.0   41.39059 …         -1.0  41.391446
162   2.163621         0.0        -1.0  41.393194 …         -1.0  41.392572
27    2.165349         0.0        -1.0  41.391999 …         -1.0  41.392965
170   2.163408         0.0        -1.0  41.393364 …         -1.0  41.392871
106   2.165496         0.0        -1.0  41.391832 …         -1.0  41.392753
147   2.164762         0.0        -1.0  41.390261 …         -1.0   41.39057
163   2.166142         0.0        -1.0   41.39297 …         -1.0  41.391593
166   2.162796         0.0        -1.0  41.389199 …         -1.0  41.389965
156   2.163595         0.0        -1.0  41.393017 …         -1.0  41.392426
156   2.161959         0.0        -1.0  41.391724 …         -1.0  41.392882
102   2.163644         0.0        -1.0  41.391076 …         -1.0  41.391744
171   2.162629         0.0        -1.0  41.388876 …         -1.0  41.389971
60    2.165538         0.0        -1.0  41.391587 …         -1.0  41.390637
53    2.166445         0.0        -1.0  41.391167 …         -1.0  41.391858
30    2.165685         0.0        -1.0  41.391595 …         -1.0  41.391602

       long_23 collision_23 victim_id_23      lat_24  long_24 collision_24  \
83    2.165269          0.0         -1.0  41.392061  2.165224          1.0
170   2.163589          0.0         -1.0  41.392836  2.163624          1.0
167   2.164129          0.0         -1.0  41.389947  2.164172          1.0
101   2.163864          0.0         -1.0  41.391328  2.163825          1.0
51    2.163893          0.0         -1.0  41.391388   2.16385          1.0
180   2.165872          0.0         -1.0  41.391519  2.165832          1.0
```

```
162  2.164381            0.0        -1.0  41.392535  2.164426           1.0
27    2.16411            0.0        -1.0  41.393002  2.164065           1.0
170  2.163967            0.0        -1.0  41.392843      2.164           1.0
106  2.164368            0.0        -1.0  41.392793    2.16432           1.0
147  2.164949            0.0        -1.0  41.390603  2.164993           1.0
163  2.165583            0.0        -1.0  41.391552  2.165633           1.0
166  2.164142            0.0        -1.0  41.390012  2.164204           1.0
156  2.164506            0.0        -1.0  41.392386  2.164555           1.0
156  2.163522            0.0        -1.0    41.3929  2.163547           1.0
102  2.162681            0.0        -1.0  41.391782  2.162631           1.0
171   2.16415            0.0        -1.0  41.390015  2.164208           1.0
60   2.166713            0.0        -1.0  41.390597  2.166762           1.0
53    2.16536            0.0        -1.0  41.391887  2.165326           1.0
30   2.164188            0.0        -1.0  41.391554  2.164123           1.0

     victim_id_24                 df_time
83         1162.0  df_collisions_1033_1058
170        1541.0  df_collisions_1508_1533
167        2104.0  df_collisions_2233_2258
101        2274.0  df_collisions_2558_2583
51         2701.0  df_collisions_3433_3458
180        3802.0  df_collisions_4133_4158
162        3967.0  df_collisions_4333_4358
27         4079.0  df_collisions_4833_4858
170        4621.0  df_collisions_5058_5083
106        4301.0  df_collisions_5133_5158
147        4711.0  df_collisions_5158_5183
163        4848.0  df_collisions_5408_5433
166        5556.0  df_collisions_6258_6283
156        5873.0  df_collisions_6758_6783
156        6491.0  df_collisions_7583_7608
102        6962.0  df_collisions_8183_8208
171        7400.0  df_collisions_8783_8808
60         7648.0  df_collisions_9258_9283
53         8145.0  df_collisions_9858_9883
30         7689.0  df_collisions_9908_9933

[20 rows x 102 columns]
```

4.0.5 Probabilidad de colisión

```python
[445]: def get_df_collision_risk(vehicles_list, df_preds, df_full):
           risk_list = []
           collision_list = []
           columns_list = ['vehicle_id']
           columns_list.extend(vehicles_list)
           for i in range(len(vehicles_list)):
```

```
        is_colision_data = []
        colision_risk_data = []
        vehicle1 = df_preds.iloc[i]
        vehicle1_id = int(df_full.iloc[i]['vehicle_id'])
        is_colision_data.append(vehicle1_id)
        colision_risk_data.append(vehicle1_id)

        for j in range(len(vehicles_list)):
            if i!=j:
                vehicle2 = df_preds.iloc[j]
                colision, map, collision_risk, time = is_colision(vehicle1,␣
 ↪vehicle2)

                is_colision_data.append(colision)
                colision_risk_data.append(collision_risk)
            else:
                is_colision_data.append(-1)
                colision_risk_data.append(-1)

        collision_list.append(is_colision_data)
        risk_list.append(colision_risk_data)
    df_is_collision_1to1 = pd.DataFrame(collision_list, columns=columns_list)
    df_collision_risk_1to1 = pd.DataFrame(risk_list, columns=columns_list)
    return df_is_collision_1to1, df_collision_risk_1to1
```

```
[446]: df_is_collision_1to1 = {}
       df_collision_risk_1to1 = {}
       for key in df_predictions.keys():
           print(key)
           vehicles_list = list(df_colls[key]['vehicle_id'].astype(int))
           preds_df = df_predictions[key]
           full_df = df_colls[key]
           df_is_collision_1to1[key], df_collision_risk_1to1[key] =␣
         ↪get_df_collision_risk(vehicles_list, preds_df, full_df)
```

```
df_collisions_1033_1058
df_collisions_1508_1533
df_collisions_2233_2258
df_collisions_2558_2583
df_collisions_3433_3458
df_collisions_4133_4158
df_collisions_4333_4358
df_collisions_4833_4858
df_collisions_5058_5083
df_collisions_5133_5158
df_collisions_5158_5183
df_collisions_5408_5433
df_collisions_6258_6283
df_collisions_6758_6783
```

```
df_collisions_7583_7608
df_collisions_8183_8208
df_collisions_8783_8808
df_collisions_9258_9283
df_collisions_9858_9883
df_collisions_9908_9933
```

[447]:
```python
import pickle
with open('df_is_collision_1to1_tree.pickle', 'wb') as f:
    pickle.dump(df_is_collision_1to1, f)

with open('df_collision_risk_1to1_tree.pickle', 'wb') as f:
    pickle.dump(df_collision_risk_1to1, f)
```

4.0.6 Comprobación colisiones Extra Tree Regressor

[448]:
```python
file = open('df_is_collision_1to1_tree.pickle', 'rb')
df_is_collision_1to1 = pickle.load(file)
file.close()

file = open('df_collision_risk_1to1_tree.pickle', 'rb')
df_collision_risk_1to1 = pickle.load(file)
file.close()
```

[482]:
```python
df_is_collision_1to1['df_collisions_1033_1058']
```

[482]:
```
     vehicle_id   157    158    171    176    197    433    465    503  \
0           157    -1  False  False  False  False  False  False  False
1           158  False    -1  False  False  False  False  False  False
2           171  False  False    -1  False  False  False  False  False
3           176  False  False  False    -1  False  False  False  False
4           197  False  False  False  False    -1  False  False  False
..          ...   ...    ...    ...    ...    ...    ...    ...    ...
191        1233  False  False  False  False  False  False  False  False
192        1234  False  False  False  False  False  False  False  False
193        1235  False  False  False  False  False  False  False  False
194        1236  False  False  False  False  False  False  False  False
195        1237  False  False  False  False  False  False  False  False

       515  …   1225   1227   1228   1230   1232   1233   1234   1235  \
0    False  …  False  False  False  False  False  False  False  False
1    False  …  False  False  False  False  False  False  False  False
2    False  …  False  False  False  False  False  False  False  False
3    False  …  False  False  False  False  False  False  False  False
4    False  …  False  False  False  False  False  False  False  False
..     … …     …      …      …      …      …      …      …      …
191  False  …  False  False  False  False  False    -1  False  False
```

```
192  False  …  False  False  False  False  False  False     -1  False
193  False  …  False  False  False  False  False  False  False     -1
194  False  …  False  False  False  False  False  False  False  False
195  False  …  False  False  False  False  False  False  False  False

         1236    1237
0     False  False
1     False  False
2     False  False
3     False  False
4     False  False
..       …      …
191   False  False
192   False  False
193   False  False
194      -1  False
195   False     -1

[196 rows x 197 columns]
```

```
[436]: pred_collisions_df = pd.DataFrame(columns=['vehicle_id', 'victim_id',␣
        ↪'df_time'])
       pred_collisions = []
       for key in df_is_collision_1to1.keys():
           curr_df = df_is_collision_1to1[key]
           for index, row in curr_df.iterrows():
               for col in curr_df.columns:
                   if row[col] == True:
                       row_vehicle = row['vehicle_id']
                       row_victim = col
                       row_time = key
                       pred_collisions.append([row_vehicle, row_victim, row_time])
```

```
[ ]: print(f"Total predicted collisions: {len(pred_collisions)/2}")
```

```
[437]: tp = 0
       for vehicle_collisioned in pred_collisions:
           if vehicle_collisioned[0] in list(real_collisions_df['vehicle_id']):
               print(vehicle_collisioned)
               tp += 1
       fp = int((len(pred_collisions) - 2*tp)/2)
       fn = len(real_collisions_df)-tp

       tn = 0
       for key in df_is_collision_1to1.keys():
```

```
    tn += (df_is_collision_1to1[key].shape[0]*(df_is_collision_1to1[key].
    ↪shape[1]-2))/2-tp-fp-fn
print(f"True positives: {tp}")
print(f"False positives: {fp}")
print(f"True negatives: {tn}")
print(f"False negatives: {fn}")
```

```
[1053, 157, 'df_collisions_1033_1058']
[3985, 4292, 'df_collisions_4833_4858']
True positives: 2
False positives: 70
True negatives: 651038
False negatives: 18
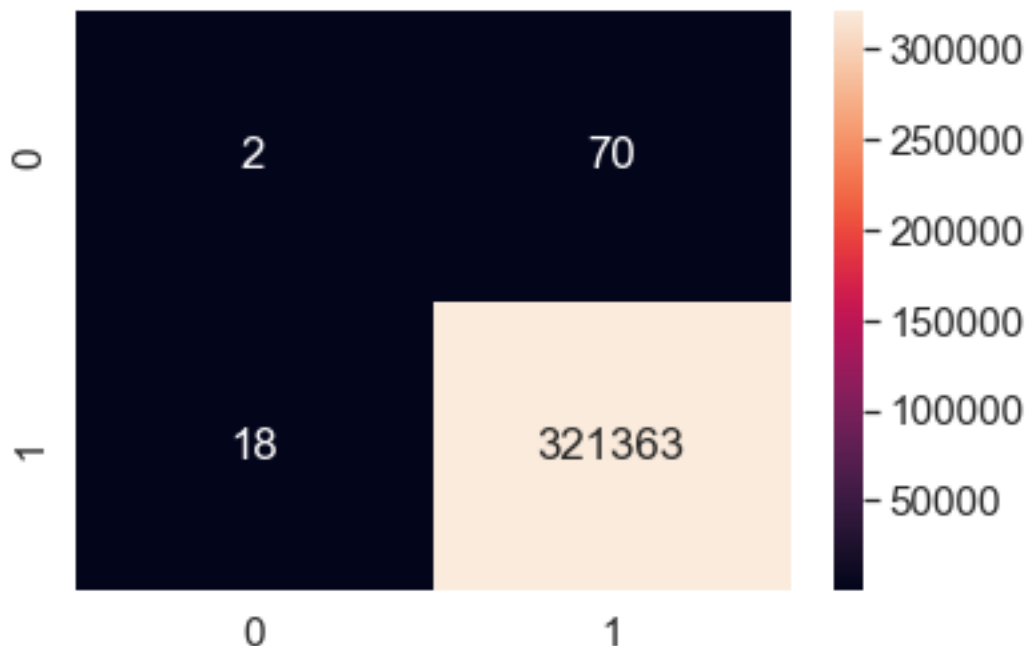```

[487]:
```
confusion_matrix = [[tp, fp], [fn, tn]]
```

[488]:
```
confusion_matrix
```

[488]: `[[2, 70], [18, 321363]]`

[489]:
```python
import seaborn as sn
df_cm = pd.DataFrame(confusion_matrix)
sn.heatmap(df_cm, annot=True, fmt='g')
plt.show()
```

```
[490]: accuracy = (tn+tp)/(tn+fp+tp+fn)
       precision = tp/(tp+fp)
       recall = tp/(tp+fn)
       f1_score = 2*((precision*recall)/(precision+recall))
       print(f"Accuracy: {accuracy}")
       print(f"Precision: {precision}")
       print(f"Recall: {recall}")
       print(f"F1 score: {f1_score}")
```

```
Accuracy: 0.9997262430277521
Precision: 0.027777777777777776
Recall: 0.1
F1 score: 0.04347826086956522
```

4.0.7 Resultados probabilidad de colisión

```
[ ]: real_collisions_df
```

```
[449]: real_collisions_df_prob = real_collisions_df[['vehicle_id', 'df_time',␣
       ↪'victim_id_15', 'victim_id_16', 'victim_id_17',
                                           'victim_id_18', 'victim_id_19',␣
       ↪'victim_id_20', 'victim_id_21',
                                           'victim_id_22', 'victim_id_23',␣
       ↪'victim_id_24']]
       real_collisions_prob_vehicle = list(real_collisions_df_prob['vehicle_id'])
       real_collisions_prob_victim = list(real_collisions_df_prob['victim_id_24'])
       real_collisions_prob_time = list(real_collisions_df_prob['df_time'])
```

4.1 Gaussian Process Regressor

```
[453]: from sklearn.gaussian_process import GaussianProcessRegressor
       df_predictions ={}
       for key in data_colls.keys():
           model = GaussianProcessRegressor()
           df_predictions[key] = call_get_next_10_positions_res(data_colls[key], model)
```

4.1.1 Probabilidad de colisión (Gaussian Process Regressor)

```
[454]: df_is_collision_1to1 = {}
       df_collision_risk_1to1 = {}
       for key in df_predictions.keys():
           print(key)
           vehicles_list = list(df_colls[key]['vehicle_id'].astype(int))
           preds_df = df_predictions[key]
           full_df = df_colls[key]
           df_is_collision_1to1[key], df_collision_risk_1to1[key] =␣
       ↪get_df_collision_risk(vehicles_list, preds_df, full_df)
```

```
df_collisions_1033_1058
df_collisions_1508_1533
df_collisions_2233_2258
df_collisions_2558_2583
df_collisions_3433_3458
df_collisions_4133_4158
df_collisions_4333_4358
df_collisions_4833_4858
df_collisions_5058_5083
df_collisions_5133_5158
df_collisions_5158_5183
df_collisions_5408_5433
df_collisions_6258_6283
df_collisions_6758_6783
df_collisions_7583_7608
df_collisions_8183_8208
df_collisions_8783_8808
df_collisions_9258_9283
df_collisions_9858_9883
df_collisions_9908_9933
```

[455]:
```python
import pickle
with open('df_is_collision_1to1_gauss.pickle', 'wb') as f:
    pickle.dump(df_is_collision_1to1, f)


with open('df_collision_risk_1to1_gauss.pickle', 'wb') as f:
    pickle.dump(df_collision_risk_1to1, f)
```

4.1.2 Comprobación colisiones (Gaussian Process Regressor

[456]:
```python
file = open('df_is_collision_1to1_gauss.pickle', 'rb')
df_is_collision_1to1 = pickle.load(file)
file.close()


file = open('df_collision_risk_1to1_gauss.pickle', 'rb')
df_collision_risk_1to1 = pickle.load(file)
file.close()
```

[457]:
```python
pred_collisions_df = pd.DataFrame(columns=['vehicle_id', 'victim_id',
 ↪'df_time'])
pred_collisions = []
for key in df_is_collision_1to1.keys():
    curr_df = df_is_collision_1to1[key]
    for index, row in curr_df.iterrows():
        for col in curr_df.columns:
            if row[col] == True:
                row_vehicle = row['vehicle_id']
```

```
                    row_victim = col
                    row_time = key
                    pred_collisions.append([row_vehicle, row_victim, row_time])
```

[458]:
```python
print(f"Total predicted collisions: {len(pred_collisions)/2}")
```

```
Total predicted collisions: 199.0
```

[466]:
```python
tp = 0
for vehicle_collisioned in pred_collisions:
    if vehicle_collisioned[0] in list(real_collisions_df['vehicle_id']):
        tp += 1
fp = int((len(pred_collisions) - 2*tp)/2)
fn = len(real_collisions_df)-tp

tn = 0
for key in df_is_collision_1to1.keys():
    tn += (df_is_collision_1to1[key].shape[0]*(df_is_collision_1to1[key].
  ↪shape[1]-2))/2-tp-fp-fn
print(f"True positives: {tp}")
print(f"False positives: {fp}")
print(f"True negatives: {tn}")
print(f"False negatives: {fn}")
```

```
True positives: 5
False positives: 194
True negatives: 321239.0
False negatives: 15
```

[467]:
```python
confusion_matrix = [[tp, fp], [fn, tn]]
```

[468]:
```python
confusion_matrix
```

[468]:
```
[[5, 194], [15, 321239.0]]
```

[469]:
```python
import seaborn as sn
df_cm = pd.DataFrame(confusion_matrix)
sn.heatmap(df_cm, annot=True, fmt='g')
plt.show()
```
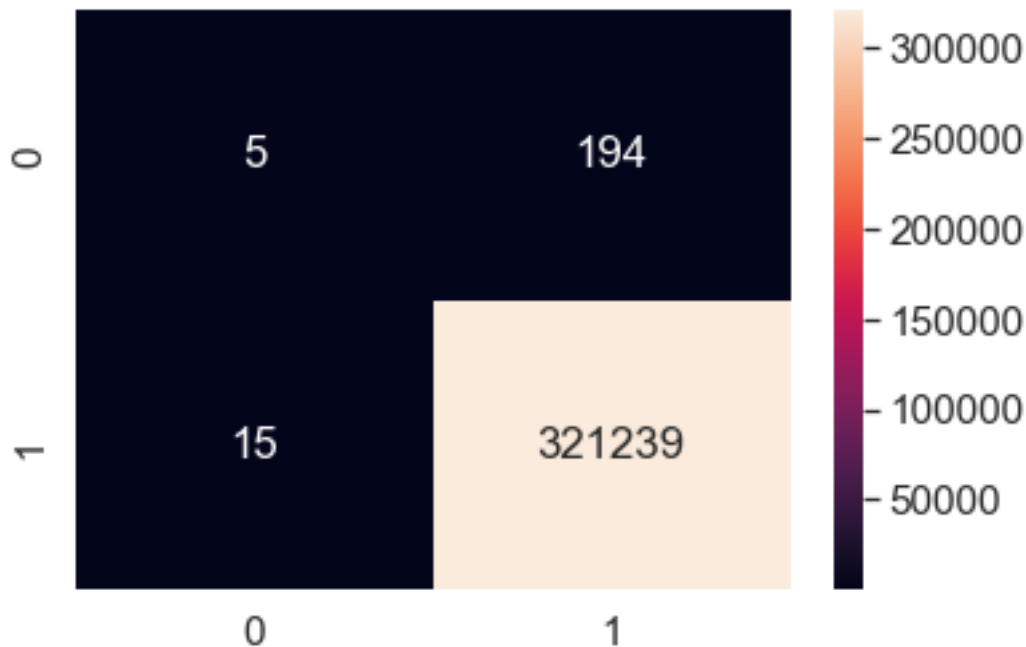
```
[479]: accuracy = (tn+tp)/(tn+fp+tp+fn)
       precision = tp/(tp+fp)
       recall = tp/(tp+fn)
       f1_score = 2*((precision*recall)/(precision+recall))
       print(f"Accuracy: {accuracy}")
       print(f"Precision: {precision}")
       print(f"Recall: {recall}")
       print(f"F1 score: {f1_score}")
```

```
Accuracy: 0.9997262430277521
Precision: 0.027777777777777776
Recall: 0.1
F1 score: 0.04347826086956522
```

4.2 Riesgo de colisión (Gaussian Process Regressor) (5 posicones)

```
[3]: df_names = []
     df_full_names = []
     for i in range(0, 8000, 20):
         df_names.append(f"collision_dfs/df_collisions_{i}_{i+20}.csv"[14:-4])
         df_full_names.append(f"collision_dfs/df_collisions_{i}_{i+20}.csv")

     collision_dfs = {}
     for i in range(len(df_names)):
```

```
    collision_dfs[df_names[i]] = pd.read_csv(df_full_names[i]).drop(["Unnamed:
↪0"], axis=1)
```

```
[4]: pred_columns  = ["lat_0", "long_0", "lat_1", "long_1", "lat_2", "long_2",
↪"lat_3", "long_3", "lat_4", "long_4",
                "lat_5", "long_5", "lat_6", "long_6", "lat_7", "long_7", "lat_8",
↪"long_8", "lat_9", "long_9",
                "lat_10", "long_10", "lat_11", "long_11", "lat_12", "long_12",
↪"lat_13", "long_13", "lat_14", "long_14",
                "lat_15", "long_15", "lat_16", "long_16", "lat_17", "long_17",
↪"lat_18", "long_18", "lat_19", "long_19"]

     real_columns  = ["lat_0", "long_0", "lat_1", "long_1", "lat_2", "long_2",
↪"lat_3", "long_3", "lat_4", "long_4",
                "lat_5", "long_5", "lat_6", "long_6", "lat_7", "long_7", "lat_8",
↪"long_8", "lat_9", "long_9",
                "lat_10", "long_10", "lat_11", "long_11", "lat_12", "long_12",
↪"lat_13", "long_13", "lat_14", "long_14"]

     columns = pred_columns
```

```
[5]: dfs_with_collisions = []
     for key in collision_dfs.keys():
         df_col = collision_dfs[key]
         for i in range(15,20):
             hay_colision = len(df_col[df_col['collision_'+str(i)]==1])
             if(hay_colision)>0:
                 dfs_with_collisions.append(key)
                 print(key)
```

```
df_collisions_0_20
df_collisions_620_640
df_collisions_1000_1020
df_collisions_1440_1460
df_collisions_2040_2060
df_collisions_2240_2260
df_collisions_2680_2700
df_collisions_2860_2880
df_collisions_4020_4040
df_collisions_4620_4640
df_collisions_4780_4800
df_collisions_5360_5380
df_collisions_5480_5500
df_collisions_6400_6420
df_collisions_6720_6740
df_collisions_7500_7520
df_collisions_7860_7880
```

```
[6]: df_colls = {k: collision_dfs[k] for k in dfs_with_collisions}
     df_colls_pos = {k: collision_dfs[k][pred_columns] for k in dfs_with_collisions}
```

```
[7]: real_collisions_df = pd.DataFrame(columns=columns + ['df_time'])
     for key in df_colls.keys():
         curr_df = df_colls[key]
         for i in range(15, 20):
             real_collisions = curr_df[curr_df['collision_'+str(i)]==1]
             real_collisions['df_time'] = key
             real_collisions_df = pd.concat([real_collisions_df, real_collisions])
```

```
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key
C:\Users\DANIE\AppData\Local\Temp\ipykernel_15044\538006242.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  real_collisions['df_time'] = key

```python
[8]: data_colls ={}
     for key in df_colls_pos.keys():
         data_colls[key] = np.load(f"reg_coll_data/{key}.npy")
```

```python
[9]: pred_columns  = ["lat_0", "long_0", "lat_1", "long_1", "lat_2", "long_2",
     ↪"lat_3", "long_3", "lat_4", "long_4",
                 "lat_5", "long_5", "lat_6", "long_6", "lat_7", "long_7", "lat_8",
     ↪"long_8", "lat_9", "long_9",
                 "lat_10", "long_10", "lat_11", "long_11", "lat_12", "long_12",
     ↪"lat_13", "long_13", "lat_14", "long_14",
                 "lat_15", "long_15", "lat_16", "long_16", "lat_17", "long_17",
     ↪"lat_18", "long_18", "lat_19", "long_19"]

     real_columns  = ["lat_0", "long_0", "lat_1", "long_1", "lat_2", "long_2",
     ↪"lat_3", "long_3", "lat_4", "long_4",
                 "lat_5", "long_5", "lat_6", "long_6", "lat_7", "long_7", "lat_8",
     ↪"long_8", "lat_9", "long_9",
                 "lat_10", "long_10", "lat_11", "long_11", "lat_12", "long_12",
     ↪"lat_13", "long_13", "lat_14", "long_14"]
```

```python
[10]: def call_get_next_5_positions_res(dataset, model):
          df_colls_res = pd.DataFrame(columns=pred_columns)
          array_colls_res = np.empty((0,40))
          array_colls_res = []
          for data in dataset:
              prediction = get_next_5_positions(data, model)
              array_colls_res.append(prediction)
          df_colls_res = pd.DataFrame(array_colls_res, columns=pred_columns)
          return df_colls_res
```

```python
[11]: def get_next_5_positions(data, model):

          df_res = data[0:15]

          # Prediction 1
          X = data[0:15]
          y = data[1:16]
          model.fit(X, y)
          pred_1 = model.predict([y[-1]])
          real_1 = data[16]

          # Prediction 2
          X = np.concatenate((data[1:15], pred_1))
```

```python
        y = np.concatenate((data[2:15], pred_1, [real_1]))
        model.fit(X, y)
        pred_2 = model.predict([y[-1]])
        real_2 = data[17]

        # Prediction 3
        X = np.concatenate((data[2:15], pred_1, pred_2))
        y = np.concatenate((data[3:15], pred_1, pred_2, [real_2]))
        model.fit(X, y)
        pred_3 = model.predict([y[-1]])
        real_3 = data[18]

        # Prediction 4
        X = np.concatenate((data[3:15], pred_1, pred_2, pred_3))
        y = np.concatenate((data[4:15], pred_1, pred_2, pred_3, [real_3]))
        model.fit(X, y)
        pred_4 = model.predict([y[-1]])
        real_4 = data[19]

        # Prediction 5
        X = np.concatenate((data[4:15], pred_1, pred_2, pred_3, pred_4))
        y = np.concatenate((data[5:15], pred_1, pred_2, pred_3, pred_4, [real_4]))
        model.fit(X, y)
        pred_5 = model.predict([y[-1]])


        #predictions = [l.tolist() for l in␣
    ↪[pred_1[0],pred_2[0],pred_3[0],pred_4[0],pred_5[0],pred_6[0],pred_7[0],pred_8[0],pred_9[0],
        #real_values = [l.tolist() for l in␣
    ↪[real_1,real_2,real_3,real_4,real_5,real_6,real_7,real_8,real_9,real_10]]

        df_res = np.append(df_res, [pred_1, pred_2, pred_3, pred_4, pred_5])

        return df_res
```

```python
[12]: from sklearn.tree import ExtraTreeRegressor
      df_predictions ={}
      for key in data_colls.keys():
          model = ExtraTreeRegressor()
          df_predictions[key] = call_get_next_5_positions_res(data_colls[key], model)
```

```python
[13]: import pickle
      file = open('df_collision_risk_1to1_gauss_5.pickle', 'rb')
      df_collision_risk_1to1 = pickle.load(file)
      file.close()
```

```
[14]:  # RIESGO COLISIÓN
       vehicles = [int(x) for x in list(real_collisions_df['vehicle_id'])]
       victims = [int(x) for x in list(real_collisions_df['victim_id_19'])]
       keys = list(real_collisions_df['df_time'])
       res = []
       for i in range(len(vehicles)-1):
           if (victims[i]!=868): # vehicle out of this data
               curr_res = []
               dataframe = df_collision_risk_1to1[keys[i]]
               fila = dataframe.loc[dataframe['vehicle_id']==vehicles[i]]
               prob = float(fila[victims[i]])
               curr_res.append([vehicles[i], victims[i], keys[i], prob])
               res.append(curr_res[0])
       res_df = pd.DataFrame(res,␣
         ↪columns=['vehicle_id','victim_id','df_time','collision_risk'])
```

```
[15]:  res_df
```

```
[15]:      vehicle_id  victim_id                df_time   collision_risk
       0          200        286      df_collisions_0_20         0.121915
       1         1166       1161   df_collisions_1000_1020         0.000000
       2         1347       1283   df_collisions_1440_1460         0.203768
       3         1749       1800   df_collisions_2040_2060         0.207146
       4         2157       2155   df_collisions_2240_2260         0.207375
       5         2022       2473   df_collisions_2680_2700         0.177073
       6         2384       2365   df_collisions_2860_2880         0.315616
       7         3690       3640   df_collisions_4020_4040         0.167884
       8         4241       4234   df_collisions_4620_4640         0.154425
       9         4279       4280   df_collisions_4780_4800         0.122508
       10        4572       4744   df_collisions_5360_5380         0.147923
       11        4980       4987   df_collisions_5480_5500         0.295958
       12        5222       4682   df_collisions_6400_6420         0.314585
       13        5823       5807   df_collisions_6720_6740         0.000000
       14        6400       6386   df_collisions_7500_7520         0.123619
```