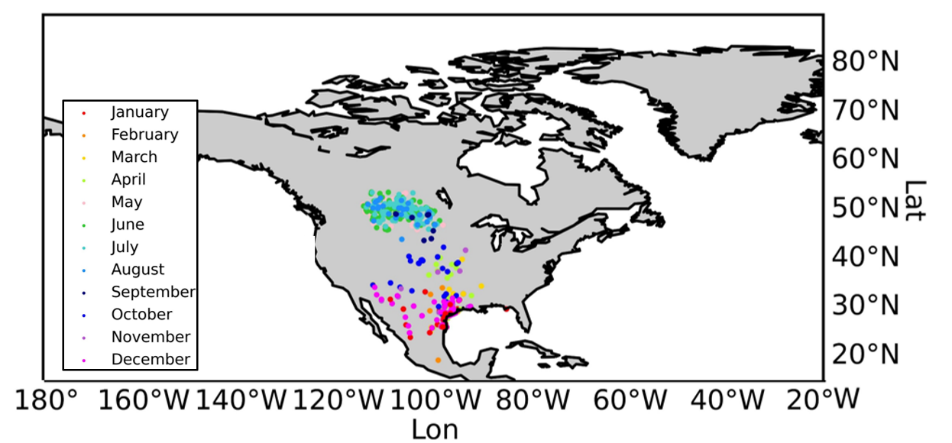
**Figures in the manuscript and the corresponding codes for them:**

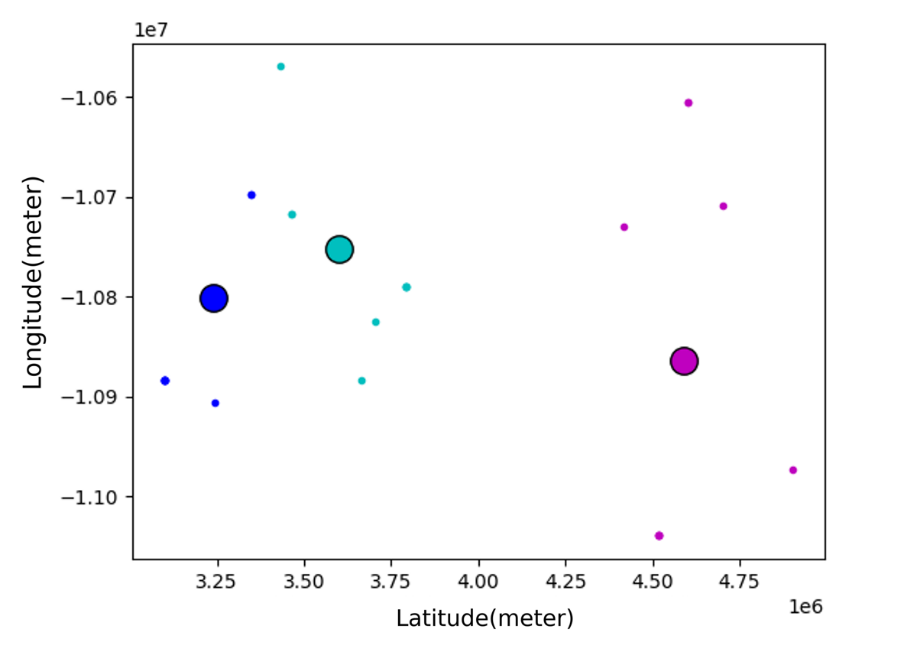
1. Show the distribution of observations



Code:

def map\_2(save\_path,csv\_name):  
 *"""  
 This function is for showing the distribution of raw observation data on the map  
  
 Args:  
 save\_path: Path for storing the raw data distribution figures  
 csv\_name: Name of the species to be processed  
  
 Returns:  
 True: Omitted  
 """* m = Basemap(llcrnrlat=-60, urcrnrlat=90, llcrnrlon=-180, urcrnrlon=-20) # Instantiate a map  
 m.drawcoastlines() # Draw the coastline  
 m.drawmapboundary(fill\_color='white')  
 m.fillcontinents(lake\_color='white') # Draw the continents and fill them in white  
  
 parallels = np.arange(-90., 90., 10.) # Draw latitudes with ranges [-90,90] and intervals of 10  
 m.drawparallels(parallels, labels=[False, True, True, False], color='none')  
 meridians = np.arange(-180., 180., 20.) # Draw the longitude with a range of [-180,180] and an interval of 10  
 m.drawmeridians(meridians, labels=[True, False, False, True], color='none')  
  
 # plt.rcParams['figure.figsize'] = (28, 8)  
 # plt.show()  
  
 datalist = readcsv(save\_path + csv\_name)  
 datalist = datalist[1:]  
  
 LON = []  
 LAT = []  
 for i in range(1):  
 Lat = [[float(data[0]),int(data[2])] for data in datalist]  
 Lon = [[float(data[1]),int(data[2])] for data in datalist]  
 LON.append(Lon)  
 LAT.append(Lat)  
  
 for doc in range(1):  
 colorMap = ['red', 'darkorange', 'gold', 'greenyellow', 'pink', 'limegreen', 'mediumturquoise',  
 'dodgerblue',  
 'navy', 'blue', 'mediumorchid', 'fuchsia']  
 # Show labels  
 label = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October',  
 'November', 'December']  
  
 marker = ['x', 'o', '.', '+', '<', '\_', '^', 'v', 'H', '|', 's', '\*']  
 j = 0  
 # print(len(lon))  
 flag = True  
  
 col1 = 43101  
 for i in range(13):  
 # print(i)  
 if doc == 0:  
 # m.plot(LON[doc][i:i + 30], LAT[doc][i:i + 30], marker=marker[doc], linewidth=0.4,  
 # color=colorMap[j],  
 # markersize=0.5, label=label[  
 # j])  
 LON1 = [data[0] for data in LON[doc] if data[1]>=col1 and data[1]<col1 + 30]  
 LAT1 = [data[0] for data in LAT[doc] if data[1]>=col1 and data[1]<col1 + 30]  
 col1 = col1 + 30  
 m.scatter(LON1, LAT1, color=colorMap[j],s=1, label=label[j])  
 # plt.show()  
 j += 1  
 if j == 12:  
 j = 0  
 if flag:  
 plt.legend(loc='lower left', shadow=True)  
 flag = False  
 continue  
  
 plt.xlabel('Lon', labelpad=10)  
 plt.ylabel('Lat')  
 plt.savefig(save\_path + '{}.jpg'.format(csv\_name.replace('.csv', '')), dpi=1000)  
 # plt.show()  
 plt.close()

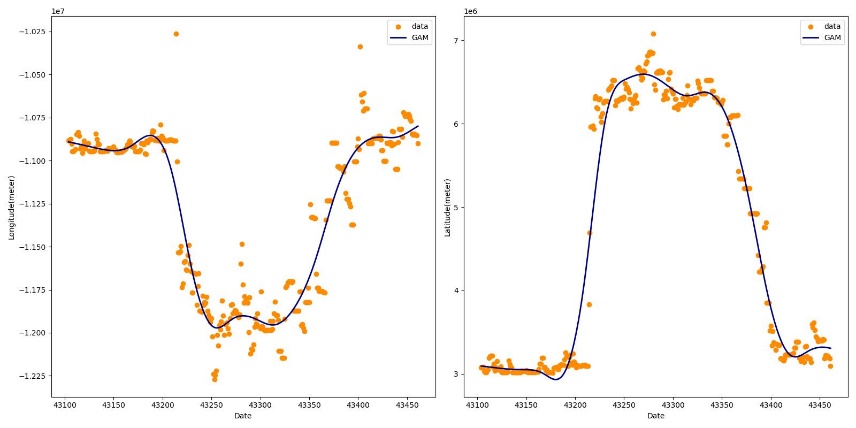
1. Get the daily centroids by Mean shift



Code:

def mean\_shift(SLDF\_df,save\_path,csv\_name):  
 *"""  
 This function is for getting centroids of high-density subpopulations by Meanshift algorithm  
  
 Args:  
 SLDF\_df: The data after sldf outlier detection  
 save\_path: The path for saving the result file  
 csv\_name: The species name being processed  
  
 Returns:  
 result: The data results after Meanshift clustering  
 """* # datas = pd.read\_excel('data/clean\_window\_data.xlsx')  
 datas = SLDF\_df.drop(['SLDF'], axis=1)  
 result = []  
 result.append(["LATITUDE", "LONGITUDE", "OBSERVATION DATE"])  
 for date in tqdm(range(43101, 43466)):  
 #for date in range(43101, 43119):  
 #print(date)  
 data = datas.loc[date == datas['OBSERVATION DATE']] # .values.tolist()#["answer"]  
 data = data.iloc[:, :2]  
 data = np.array(data)  
 if len(data) == 0:  
 continue  
  
 ms = MeanShift()  
 ms.fit(data)  
 labels = ms.labels\_  
 cluster\_centers = ms.cluster\_centers\_  
  
 labels\_unique = np.unique(labels)  
 n\_clusters = len(labels\_unique)  
  
 for c in cluster\_centers:  
 result.append([float(c[0]), float(c[1]), date])  
  
 colors = cycle('bcmyk')  
 if date % 10 == 0:  
 for k, color in zip(range(n\_clusters), colors):  
 # current\_member indicates true if the label is k and false if not  
 current\_member = labels == k  
 cluster\_center = cluster\_centers[k]  
 # Draw plots  
 plt.plot(data[current\_member, 0], data[current\_member, 1], color + '.')  
 # Draw circles  
 plt.plot(cluster\_center[0], cluster\_center[1], 'o',  
 markerfacecolor=color,  
 markeredgecolor='k',  
 markersize=14)  
  
 plt.xlabel('Latitude(meter)')  
  
 plt.ylabel('Longitude(meter)')  
 # plt.show()  
 plt.savefig(os.path.join(save\_path, csv\_name.replace('.csv', ''), 'centroids\_{}.jpg'.format(date)), dpi=1000)  
 plt.close()  
 return result

1. Fit the latitude and longitude with date by GAM



Code:

def gam(save\_path, csv\_name, key):  
 *"""  
 This function is for gam algorithm  
  
 Args:  
 save\_path: The path for saving the result file  
 csv\_name: The species name being processed  
 key: The number for file naming  
  
 Returns:  
 Lon: The longitude after GAM fitting  
 Lat: The latitude after GAM fitting  
 """* df = pd.read\_csv(os.path.join(save\_path, csv\_name.replace('.csv', ''), 'group{}.csv'.format(key + 1)))  
 date = df["date\_index"]  
 x = df["X"]  
 y = df["Y"]  
 xx = df["date\_index"]  
 xx = xx + 43100  
  
  
 gam\_model\_x = LinearGAM().fit(date, x)  
 gam\_model\_y = LinearGAM().fit(date, y)  
  
 predictions\_x = gam\_model\_x.predict(date)  
 predictions\_y = gam\_model\_y.predict(date)  
  
 # Draw the pictures  
 plt.figure(figsize=(16, 8))  
 plt.subplot(1, 2, 1)  
 plt.scatter(xx, y, color='darkorange', label='data')  
 plt.plot(xx, predictions\_y, color='navy', lw=2, label='GAM')  
 # plt.plot(X\_all, y\_gb\_longitude\_pred, color='c', lw=2, label='Gradient Boosting')  
 plt.xlabel('Date')  
 plt.ylabel('Longitude(meter)')  
 #plt.title('Longitude')  
 plt.legend()  
  
 plt.subplot(1, 2, 2)  
 plt.scatter(xx, x, color='darkorange', label='data')  
 plt.plot(xx, predictions\_x, color='navy', lw=2, label='GAM')  
 # plt.plot(X\_all, y\_gb\_latitude\_pred, color='c', lw=2, label='Gradient Boosting')  
 plt.xlabel('Date')  
 plt.ylabel('Latitude(meter)')  
 #plt.title('Latitude')  
 plt.legend()  
 plt.tight\_layout()  
 # plt.show()  
 plt.savefig(os.path.join(save\_path, csv\_name.replace('.csv', ''), 'gam{}.jpg'.format(key + 1)))  
 plt.close()

1. Fit the latitude and longitude with date by Random Forests

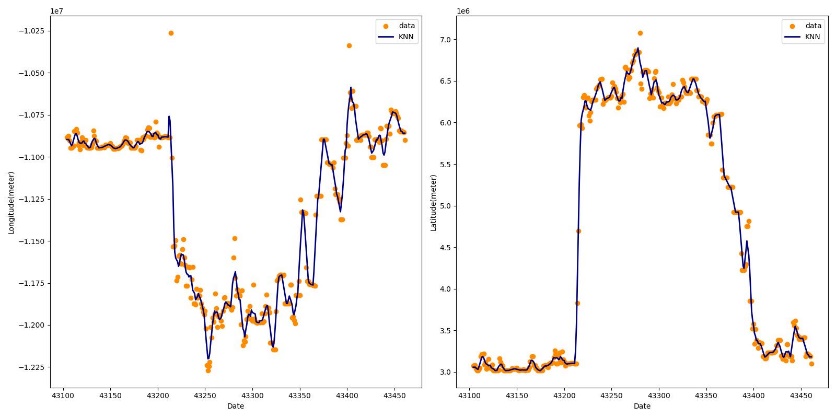
图表, 直方图

描述已自动生成

Code:

def randomforest(save\_path, csv\_name,key,n\_estimators,random\_state):  
 *"""  
 This function is for Random forests algorithm  
  
 Args:  
 save\_path: The path for saving the result file  
 csv\_name: The species name being processed  
 key: The number for file naming  
 n\_estimators: The number of trees  
 random\_state: randomness  
  
 Returns:  
 Lon: The longitude after Random Forests algorithm  
 Lat: The latitude after Random Forests algorithm  
  
 """* df = pd.read\_csv(os.path.join(save\_path, csv\_name.replace('.csv', ''), 'group{}.csv'.format(key + 1)))  
 date = df["date\_index"]  
 xx = df["date\_index"]  
 xx = xx + 43103  
  
 X = df[['date\_index']] # Features (date)  
 y\_longitude = df['Y'] # Target variable (longitude)  
 y\_latitude = df['X'] # Target variable (latitude)  
  
 # Train the model  
 rf\_longitude = RandomForestRegressor(n\_estimators=n\_estimators,random\_state=random\_state)  
 rf\_latitude = RandomForestRegressor(n\_estimators=n\_estimators,random\_state=random\_state)  
 rf\_longitude.fit(X, y\_longitude)  
 rf\_latitude.fit(X, y\_latitude)  
  
 y\_rf\_longitude\_pred = rf\_longitude.predict(X)  
 y\_rf\_latitude\_pred = rf\_latitude.predict(X)  
  
  
 # Draw the pictures  
 plt.figure(figsize=(16, 8))  
 plt.subplot(1, 2, 1)  
 plt.scatter(xx, y\_longitude, color='darkorange', label='data')  
 plt.plot(xx, y\_rf\_longitude\_pred, color='navy', lw=2, label='Random Forest')  
 # plt.plot(X\_all, y\_gb\_longitude\_pred, color='c', lw=2, label='Gradient Boosting')  
 plt.xlabel('Date')  
 plt.ylabel('Longitude(meter)')  
 #plt.title('Longitude')  
 plt.legend()  
  
 plt.subplot(1, 2, 2)  
 plt.scatter(xx, y\_latitude, color='darkorange', label='data')  
 plt.plot(xx, y\_rf\_latitude\_pred, color='navy', lw=2, label='Random Forest')  
 # plt.plot(X\_all, y\_gb\_latitude\_pred, color='c', lw=2, label='Gradient Boosting')  
 plt.xlabel('Date')  
 plt.ylabel('Latitude(meter)')  
 #plt.title('Latitude')  
 plt.legend()  
 plt.tight\_layout()  
 # plt.show()  
 plt.savefig(os.path.join(save\_path, csv\_name.replace('.csv', ''), 'randomforest{}.jpg'.format(key + 1)))  
 plt.close()

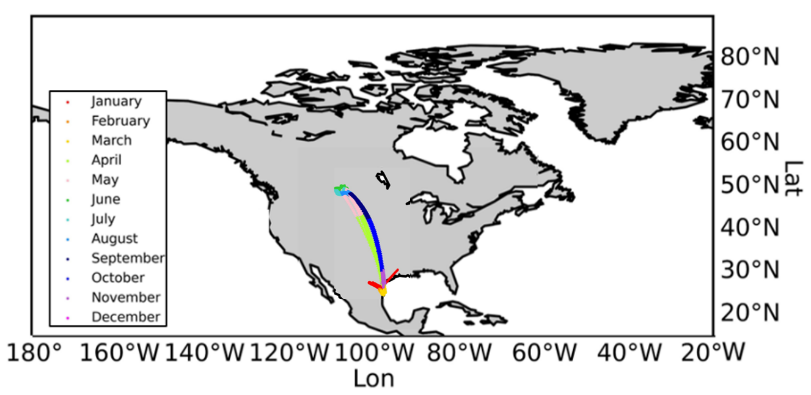
1. Fit the latitude and longitude with date by K-Nearest Neighbors



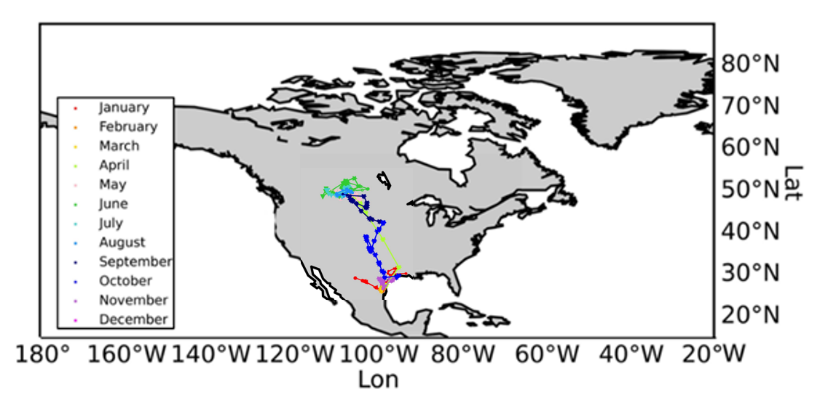
Code:

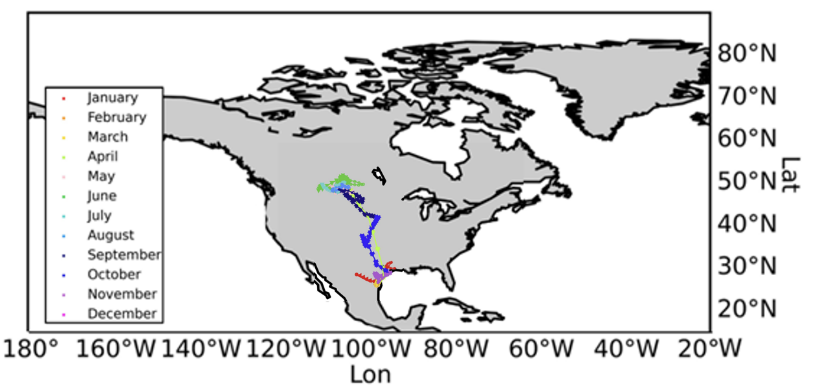
def knn(save\_path, csv\_name,key,n\_neighbors):  
 *"""  
 This function is for KNN algorithm  
  
 Args:  
 save\_path: The path for saving the result file  
 csv\_name: The species name being processed  
 key: The number for file naming  
 n\_neighbors: The number of neighbors  
  
 Returns:  
 Lon: The longitude after KNN fitting  
 Lat: The latitude after KNN fitting  
  
 """* df = pd.read\_csv(os.path.join(save\_path, csv\_name.replace('.csv', ''), 'group{}.csv'.format(key + 1)))  
 date = df["date\_index"]  
 xx = df["date\_index"]  
 xx = xx + 43103  
  
 X = df[['date\_index']] # Features (date)  
 y\_longitude = df['Y'] # Target variable (longitude)  
 y\_latitude = df['X'] # Target variable (latitude)  
  
  
 # Train the model  
 knn\_longitude = KNeighborsRegressor(n\_neighbors=n\_neighbors)  
 knn\_latitude = KNeighborsRegressor(n\_neighbors=n\_neighbors)  
 knn\_longitude.fit(X, y\_longitude)  
 knn\_latitude.fit(X, y\_latitude)  
  
 y\_knn\_longitude\_pred = knn\_longitude.predict(X)  
 y\_knn\_latitude\_pred = knn\_latitude.predict(X)  
  
 # Draw the pictures  
 plt.figure(figsize=(16, 8))  
 plt.subplot(1, 2, 1)  
 plt.scatter(xx, y\_longitude, color='darkorange', label='data')  
 plt.plot(xx, y\_knn\_longitude\_pred, color='navy', lw=2, label='KNN')  
 # plt.plot(X\_all, y\_gb\_longitude\_pred, color='c', lw=2, label='Gradient Boosting')  
 plt.xlabel('Date')  
 plt.ylabel('Longitude(meter)')  
 #plt.title('Longitude')  
 plt.legend()  
  
 plt.subplot(1, 2, 2)  
 plt.scatter(xx, y\_latitude, color='darkorange', label='data')  
 plt.plot(xx, y\_knn\_latitude\_pred, color='navy', lw=2, label='KNN')  
 # plt.plot(X\_all, y\_gb\_latitude\_pred, color='c', lw=2, label='Gradient Boosting')  
 plt.xlabel('Date')  
 plt.ylabel('Latitude(meter)')  
 #plt.title('Latitude')  
 plt.legend()  
 plt.tight\_layout()  
 # plt.show()  
 plt.savefig(os.path.join(save\_path, csv\_name.replace('.csv', ''), 'KNN{}.jpg'.format(key + 1)))  
 plt.close()

1. Show the trajectories according to different centroids fitting model

****

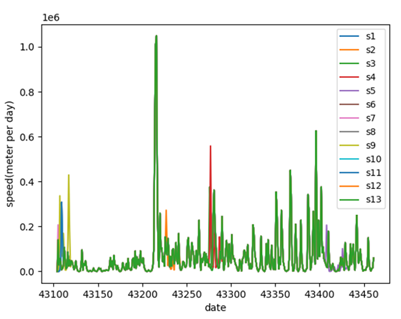
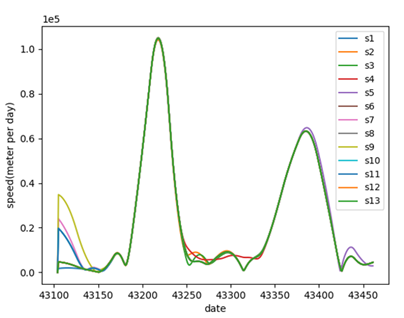
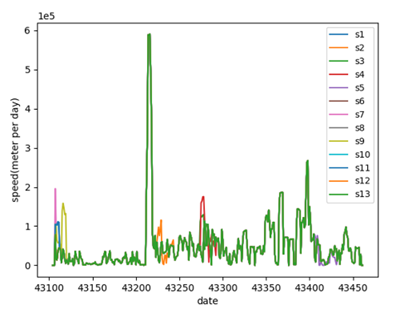
1. Results of trajectory estimation with GAM fitting

****

1. Results of trajectory estimation with RF fitting
2. Results of trajectory estimation with K-NN fitting

Code:

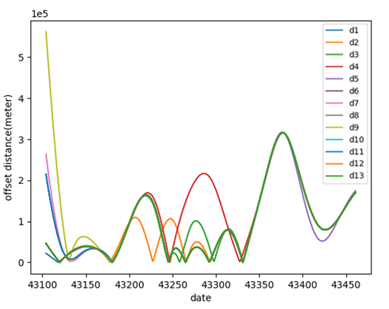
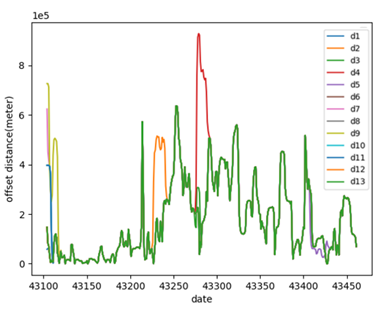
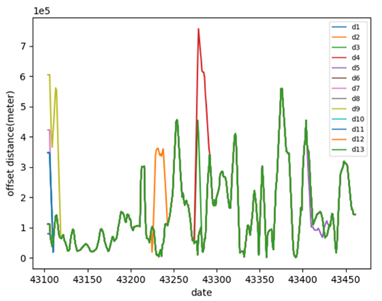
def map\_1(save\_path,csv\_name,type\_name, ):  
 *"""  
 This function is for showing the trajectories on the map  
  
 Args:  
 save\_path: The path for storing the result figures  
 csv\_name: The name of the species to be processed  
 type\_name: The fitting model chosen for centroids fitting  
  
 Returns:  
 True: Omitted  
  
 """* # plt.rcParams['figure.figsize'] = (28, 8)  
 # plt.show()  
  
 excel\_list = os.listdir(os.path.join(save\_path, csv\_name.replace('.csv', '')))  
 excel\_list1 = []  
 for csv\_excel in excel\_list:  
 if 'group' in csv\_excel:  
 excel\_list1.append(csv\_excel)  
  
 LON = []  
 LAT = []  
 for i in range(len(excel\_list1)):  
 if type\_name == 'gam':  
 Lon, Lat = gam(save\_path, csv\_name, i)  
 elif type\_name == 'randomforest':  
 n\_estimators = 100  
 random\_state = 42  
 Lon, Lat = randomforest(save\_path, csv\_name, i,n\_estimators,random\_state)  
 elif type\_name == 'knn':  
 n\_neighbors = 5  
 Lon, Lat = knn(save\_path, csv\_name, i,n\_neighbors)  
  
 LON.append(Lon)  
 LAT.append(Lat)  
  
 m = Basemap(llcrnrlat=-60, urcrnrlat=90, llcrnrlon=-180, urcrnrlon=-20) # Instantiate a map  
 m.drawcoastlines() # Draw the coastline  
 m.drawmapboundary(fill\_color='white')  
 m.fillcontinents(lake\_color='white') # Draw the continents and fill them in white  
  
 parallels = np.arange(-90., 90., 10.) # Draw latitudes with ranges [-90,90] and intervals of 10  
 m.drawparallels(parallels, labels=[False, True, True, False], color='none')  
 meridians = np.arange(-180., 180., 20.) # Draw the longitude with a range of [-180,180] and an interval of 10  
 m.drawmeridians(meridians, labels=[True, False, False, True], color='none')  
 for doc in range(0, len(LON)):  
 colorMap = ['red', 'darkorange', 'gold', 'greenyellow', 'pink', 'limegreen', 'mediumturquoise',  
 'dodgerblue',  
 'navy', 'blue', 'mediumorchid', 'fuchsia']  
 # Show labels  
 label = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October',  
 'November', 'December']  
  
 marker = ['x', '.', 'o', '|', '\*', '.', '<', '>', ',', '.', '.', 'v', 'x', 'o', '|', '\*', '<', '^', '.', '\*', 'v', '\*', ',', 'y', '.', '.', '.', '.']  
 j = 0  
 # print(len(lon))  
 flag = True  
  
  
 for i in range(0, len(LON[doc]) - 30, 30):  
 # print(i)  
 if doc == 0:  
 m.plot(LON[doc][i:i + 30], LAT[doc][i:i + 30], marker=marker[doc], linewidth=0.4,  
 color=colorMap[j],  
 markersize=0.5, label=label[  
 j])  
 # plt.show()  
 j += 1  
 if j == 12:  
 j = 0  
 if flag:  
 plt.legend(loc='lower left', shadow=True)  
 flag = False  
 continue  
 else:  
 m.plot(LON[doc][i:i + 30], LAT[doc][i:i + 30], marker=marker[doc], linewidth=0.4,  
 color=colorMap[j],  
 markersize=0.5)  
 # plt.show()  
 j += 1  
 if j == 12:  
 j = 0  
 if flag:  
 plt.legend(loc='lower left', shadow=True)  
 flag = False  
 continue  
  
 plt.xlabel('Lon', labelpad=10)  
 plt.ylabel('Lat')  
 plt.savefig(os.path.join(save\_path, csv\_name.replace('.csv', ''), 'trajectories.jpg'), dpi=1000)  
 # plt.show()  
 plt.close()

1. Speed calculation according to different centroids fitting model

Code:

def calculate\_speed(path):  
 *"""  
 This function is for calculating the speed  
  
 Args:  
 path: The file path  
  
 Returns:  
 list\_speed: The speed results  
   
 """* df = pd.read\_csv(path, header=None)  
 list\_speed = [0]  
 for i in range(1,358):  
 # x1 = float(df.iloc[i + 1, 0])  
 x = float(df.iloc[i + 1, 0]) - float(df.iloc[i, 0])  
 y = float(df.iloc[i + 1, 1]) - float(df.iloc[i, 1])  
 distance = (x \*\* 2 + y \*\* 2) \*\* 0.5  
 list\_speed.append(distance)  
 return list\_speed  
  
def speed\_main(path):  
 *"""  
 This function is for saving and drawing the results  
  
 Args:  
 path: The file path  
  
 Returns:  
 True: Omitted  
   
 """* path1 = get\_file\_list(path)  
 list\_averange = []  
 index = 1  
 xx = [i for i in range(43104,43462)]  
 for i in path1:  
 name1 = os.path.basename(i)  
 if name1[0:2] == 'fi':  
 #print(name1)  
 result = calculate\_speed(i)  
 result1 = [[datas] for datas in result]  
 # result2 = []  
 # for k in range(len(result)):  
 # result2.append([43101 + k,result[k]])  
 savecsvs(os.path.join(path, 'speed', 's{}.csv'.format(str(index))),result1)  
 list\_averange.append(result)  
  
 #plt.xlim(0,600)  
 plt.plot(xx,result,label='s{}'.format(str(index)))  
 #plt.plot(xx, result)  
 plt.xlabel("date")  
 plt.ylabel("speed(meter per day)")  
 #plt.title('%s'%i)  
 index = index + 1  
  
 result2 = []  
 for m in range(len(list\_averange[0])):  
 sum = 0  
 for n in range(len(list\_averange)):  
 sum = sum + float(list\_averange[n][m])  
 #savecsv(os.path.join(path, 'speed', 'sa.csv'),[sum/len(list\_averange)])  
 result2.append(sum/len(list\_averange))  
  
  
 plt.ticklabel\_format(style='scientific', axis='y', scilimits=(0, 0))  
 plt.legend()  
 plt.savefig(os.path.join(path, 'speed', 'SpeedPerDay.png'))  
 plt.show()

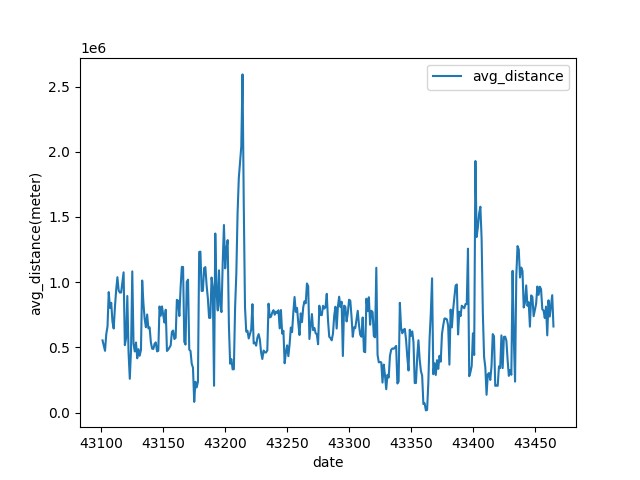
1. Offset distance calculation according to different centroids fitting model



Code:

def get\_distance\_point2line(point, line):  
 *"""  
 This function is for calculating the distance from points to a line  
  
 Args:  
 point: The observation points  
 line: A line  
  
 Returns:  
 distance: The distance from points to the line  
   
 """* line\_point1, line\_point2 = np.array(line[0:2]), np.array(line[2:])  
 vec1 = line\_point1 - point  
 vec2 = line\_point2 - point  
 distance = np.abs(np.cross(vec1, vec2)) / np.linalg.norm(line\_point1 - line\_point2)  
 return distance  
  
#Index calculation of offset distance  
def calculate\_distance(path\_list):  
 *"""  
 This function is for getting migration axis and offset distance  
  
 Args:  
 path\_list: The files in the path  
  
 Returns:  
 list11: The offset distances  
   
 """* list\_distance = []  
 for l in path\_list:  
 name1 = os.path.basename(l)  
 if name1[:2] == 'fi':  
 tmp = pd.read\_csv(l, header=None)  
 list\_distance.append(tmp)  
  
 mean\_x = []  
 mean\_y = []  
 for q in list\_distance:  
 mean\_x.append(float(q.iloc[1, 0]))  
 mean\_y.append(float(q.iloc[1, 1]))  
  
 mean\_x1 = np.mean(mean\_x)  
 mean\_y1 = np.mean(mean\_y)  
 print(mean\_x1, mean\_y1)  
  
  
 dict1 = {}  
 list1 = []  
 max\_list\_x = []  
 max\_list\_y = []  
 for r in path\_list:  
 name1 = os.path.basename(r)  
 if name1[0:2] == 'fi':  
 tmp = pd.read\_csv(r, header=None)  
 for l in range(1,359):  
 x1 = float(tmp.iloc[1, 0])  
 y1 = float(tmp.iloc[1, 1])  
  
 x = float(tmp.iloc[l, 0])  
 y = float(tmp.iloc[l, 1])  
 distance1 = ((x - x1) \*\* 2 + (y - y1) \*\* 2) \*\* 0.5  
 dict1[distance1] = l  
 list1.append(distance1)  
 max\_distance = max(list1)  
 max\_x\_1 = float(tmp.iloc[dict1[max\_distance], 0])  
 max\_y\_1 = float(tmp.iloc[dict1[max\_distance], 1])  
 max\_list\_x.append(max\_x\_1)  
 max\_list\_y.append(max\_y\_1)  
 list1 = []  
 dict1 = {}  
  
 mean\_x2 = np.mean(max\_list\_x)  
 mean\_y2 = np.mean(max\_list\_y)  
 print(mean\_x2, mean\_y2)  
  
  
 line = [mean\_x1, mean\_y1, mean\_x2, mean\_y2]  
  
 list11 = []  
 for r in path\_list:  
 list\_tmp = []  
 name1 = os.path.basename(r)  
 if name1[0:2] == 'fi':  
 tmp = pd.read\_csv(r, header=None)  
 for p in range(1,359):  
 x = float(tmp.iloc[p, 0])  
 y = float(tmp.iloc[p, 1])  
 distance2 = get\_distance\_point2line((x, y), line)  
 list\_tmp.append(distance2)  
 list11.append(list\_tmp)  
 return list11  
  
def off\_distance\_main(path):  
 *"""  
 This function is for saving and drawing the results  
  
 Args:  
 path: The file path  
  
 Returns:  
 True: Omitted  
   
 """* path1 = get\_file\_list(path)  
 result = calculate\_distance(path1)  
 count1 = 0  
 tmp = 0  
 index = 1  
 for j in result:  
 list\_averange = []  
 result2 = []  
 items = [[datas] for datas in j]  
 savecsvs(os.path.join(path, 'off\_distance', 'd{}.csv'.format(str(index))),items)  
 index = index + 1  
 for b in range(len(j)):  
 for l in result:  
 tmp += l[b]  
 averange = int(tmp/len(result))  
 tmp = 0  
 result2.append(averange)  
 items = [[datas] for datas in result2]  
 #savecsvs(os.path.join(path, 'off\_distance', 'da.csv'), items)  
  
 count = 1  
 xx = [i for i in range(43104, 43462)]  
 for l in result:  
 name = os.path.basename(path1[count])  
 #plt.xlim(0,360)  
 plt.plot(xx,l,label='d{}'.format(count))  
 #plt.plot(xx, l)  
 plt.xlabel("date")  
 plt.ylabel("offset distance(meter)")  
 #plt.title('%s'%l)  
 count += 1  
 #plt.title('%s'%name)  
  
 plt.ticklabel\_format(style='scientific', axis='y', scilimits=(0, 0))  
 plt.legend()  
 plt.savefig(os.path.join(path, 'off\_distance', 'OffsetDistancePerDay.png'))  
 plt.show()

1. Centroid average distances



Code:

def get\_avg(datalist):  
 *"""  
 This function is for obtaining the average distance  
  
 Args:  
 datalist: The data to be processed  
  
 Returns:  
 avg: The average distance  
   
 """* data1 = [datalist[0][0],datalist[0][1]]  
 avg\_datalist = []  
 for i in range(1,len(datalist)):  
 datas = [datalist[i][0],datalist[i][1]]  
 avg\_datalist.append(dist\_eclud(data1,datas))  
 if avg\_datalist:  
 avg = sum(avg\_datalist)/(len(datalist) - 1)  
 else:  
 avg = 0  
 return avg  
   
#Calculate the average Euclidean distance for daily centroids  
def avg\_distance(save\_path):  
 *"""  
 This function is for calculating the average distance for daily centroids  
  
 Args:  
 save\_path: The path of files  
  
 Returns:  
 True: Omitted  
   
 """* df=pd.read\_csv(os.path.join(save\_path, 'centroids.csv'))  
 train\_data = np.array(df)  
 excel\_list = train\_data.tolist() # list  
 # excel\_list = excel\_list[1:]  
  
 datelist = get\_date\_list(excel\_list)  
  
 avg\_list = []  
 for date in datelist:  
 same\_date = get\_same\_date\_list(date,excel\_list)  
 avg\_date = get\_avg(same\_date)  
 avg\_list.append(avg\_date)  
 savecsv(os.path.join(save\_path, 'avg.csv'),[date,avg\_date])  
  
 plt.plot(datelist, avg\_list, label="avg\_distance")  
 plt.xlabel("date")  
 plt.ylabel("avg\_distance(meter)")  
 plt.legend(loc="best")  
 plt.gca().ticklabel\_format(axis="x", useOffset=False)  
 plt.savefig(os.path.join(save\_path, 'avg.png'))  
 plt.show()