In [1]:

```
import pandas as pd
import numpy as np
import os
from tqdm import tqdm
from datetime import datetime
from datetime import timedelta
from datetime import date as dtdt
from pytz import timezone
import itertools
import math
# from datetime import datetime, timedelta

################################################################################
# Important, make sure to use the correct file
weather_station = 'MI-14850-TRAVERSE_CITY_CHERRY_CPTL_AP'

################################################################################
# In this part, we select the number of years used for the experiment
start_year_lib = [2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015

# start_year_lib = [2006, 2007, 2008]

hour_lib = ['00:00', '01:00', '02:00', '03:00', '04:00', '05:00', '06:00', '
            '10:00', '11:00', '12:00', '13:00', '14:00', '15:00', '16:00', '
            '18:00', '19:00', '20:00', '21:00', '22:00', '23:00']

# Initialize an empty dataframe to store the combined data
df_combined_all = pd.DataFrame()

# Initialize an empty list to store the lengths of each 'df_GOES_meteo_combi
lengths_list = []


def daterange(date1, date2):
    for n in range(int ((date2 - date1).days) + 1):
        yield date1 + timedelta(n)

################################################################################
def round_datetime_to_nearest_hour(obj_arr, STATE_ID):
    arr_len = len(obj_arr)
    Date_CST = []
    Time_CST = []
    for i in range(arr_len):
        # iterate through
        date_item = obj_arr['Date'][i]
        time_item = obj_arr['Time'][i]
        t = datetime.strptime(date_item + " " + time_item, "%Y-%m-%d %H:%M")
        # Calculate the number of minutes past the last full hour
        minutes_past_hour = t.minute + t.second / 60
        # Round up to the next whole number of hours if the time is more tha
        # or round down to the current hour if it's less than 30 minutes pas
        if minutes_past_hour >= 30:
            num_hours = math.ceil(minutes_past_hour / 60)
        else:
```

```
            num_hours = 0
        # Create a new datetime object representing the rounded time
        if STATE_ID in ['IL', 'WI']:
            # No need to dial back one hour
            rounded_time_temp = t + timedelta(hours=num_hours)
        else:
            rounded_time_temp = t + timedelta(hours=num_hours-1)
        rounded_time=datetime(year=rounded_time_temp.year,
                        month=rounded_time_temp.month,
                        day=rounded_time_temp.day,
                        hour=rounded_time_temp.hour, minute=0, second=0)
        result_stamp = rounded_time.strftime("%Y-%m-%d %H:%M")

        new_date, new_time = result_stamp.split(' ')
        Date_CST.append(new_date)
        Time_CST.append(new_time)

    obj_arr['Date_CST'] = Date_CST
    obj_arr['Time_CST'] = Time_CST
    return obj_arr

##############################################################################
def check_snow_24_120(df):
    does_snow_24_120 = []
    for i in range(len(df)):
        if i + 120 < len(df):
            if any(df['is_snow_precip'].iloc[i+24:i+120]):
                does_snow_24_120.append(True)
            else:
                does_snow_24_120.append(False)
        else:
            does_snow_24_120.append(False)
    df['does_snow_24_120'] = does_snow_24_120
    return df

##############################################################################
def lake_1D_matcher(df_temp, df_lake_1D_map):
    # left join df_temp with df_lake_1D_map based on latitude and longitude
    df_merged_temp = pd.merge(df_lake_1D_map, df_temp, on=['latitude', 'long

    # extract value column into a list
    value_temp = df_merged_temp['value'].tolist()

    return value_temp

##############################################################################
def is_valid_data(df_temp):

    crit_1 = 0
    crit_2 = 0
    crit_3 = 0

    # Pass in value
    i_temp = df_temp.value
```

```
    try:

        i_temp_max = i_temp.max().max()

        # Acquire mode in the array
        i_temp_mode = i_temp.mode()[0]
    except:
        i_temp_max = 0
        i_temp_mode = 0

    # Check Criteria #1:
    if i_temp_max <= 0.03:
        crit_1 = 1

    # Check Criteria #2:
    if i_temp_mode <= 0.02:
        crit_2 = 1

    # Check Criteria #3
    if len(i_temp) <= 3000:
        crit_3 = 1

    crit_sum = crit_1 + crit_2 + crit_3

    if crit_sum > 0:
        cond = False
    else:
        cond = True

    return cond


def cloud_finder(value_temp):
    # Deduct 0.1 from all elements in the array
    value_temp = [x - 0.1 for x in value_temp]

    # Count the number of elements that are larger than or equal to 0
    count = sum(1 for x in value_temp if x >= 0)

    return count




for start_year in start_year_lib:
    ## It is always the case
    end_year = start_year + 1

    filename = weather_station[0:9]+str(start_year)+'Fall-'+str(end_year)+'S

    ## Extract the state indicator
    STATE_ID = filename[:2]
    STATE_2_LTR = filename[:2]
    # print(STATE_2_LTR)

    ##########################################################################
```

```
    # Check if they exist
    if os.path.exists('NSW_Weather/'+weather_station+'/'):
        print('Weather station data exists for ', weather_station)
    else:
        print('Weather station data does not exist for ', weather_station)

    if os.path.isfile('NSW_Weather/'+weather_station+'/'+filename):
        print("File exists, reading table!")
        temp_table = pd.read_csv('NSW_Weather/'+weather_station+'/'+filename
    else:
        print("File does not exist, please rewind!")

    ##########################################################################
    # Go get the satellite imagery data

    folder_name = 'zone_0_'+filename[9:13]+'Fall_'+filename[18:22]+'Spring'
    # print(folder_name)

    # Acquire the folder location for 2-D lake data based on folder_name
    folder_name_2D = folder_name[0:7]+'T_'+folder_name[7:]
    # print(folder_name_2D)

    # Add the parent folder name
    parent_path = 'GOES_Hourly_Statistics/'

    # Check if they exist
    if os.path.exists(parent_path + folder_name):
        print('1-D Lake Michigan data exists for ', folder_name[7:15], ' to
    else:
        print('1-D Lake Michigan data does not exist for ', folder_name[7:15

    if os.path.exists(parent_path + folder_name_2D):
        print('2-D Lake Michigan data exists for ', folder_name_2D[9:17], '
    else:
        print('2-D Lake Michigan data does not exist for ', folder_name[9:17

    ##########################################################################
    # Start a new part
#     print(type(start_year))
#     print(type(start_year))
#     print(type(start_year))
#     print(type(start_year))
    temp_start = int(start_year)
    temp_end = int(end_year)
    start_dt = dtdt(temp_start, 10, 1)
    end_dt = dtdt(temp_end, 3, 31)
    weather_date_theo = [dt.strftime("%Y%m%d") for dt in daterange(start_dt,
    csv_date_list = list(itertools.chain.from_iterable(itertools.repeat(x, 2
    csv_time_list = hour_lib * len(weather_date_theo)
    # Example usage
    weather_with_CST = round_datetime_to_nearest_hour(temp_table, STATE_ID)
    weather_with_CST['Precip (in)'] = weather_with_CST['Precip (in)'].replac
    weather_with_CST['Temp (F)'] = weather_with_CST['Temp (F)'].replace('M',
    is_snow_precip = ((weather_with_CST['Precip (in)'] > 0) & (weather_with_
    weather_with_CST['is_snow_precip'] = is_snow_precip
    weather_with_CST = check_snow_24_120(weather_with_CST)
```

```
# Initialize empty list to store results
# does_snow_24_120 = []

# # Iterate through each row in the 'is_snow_precip' column
# for i in range(len(weather_with_CST['is_snow_precip'])):

#     # Check if the next 24 to 120 rows contain more than 2 True values
#     if i+120 < len(weather_with_CST['is_snow_precip']):
#         rolling_sum = weather_with_CST['is_snow_precip'][i+24:i+121].r
#         count_true = np.sum(rolling_sum > 2)
#     else:
#         count_true = 0

#     # Append the result to the list
#     does_snow_24_120.append(count_true)

# # Add the list to the dataframe
# weather_with_CST['does_snow_24_120'] = does_snow_24_120
Date_CST = pd.Series(csv_date_list, name = 'Date_CST')
Time_CST = pd.Series(csv_time_list, name = 'Time_CST')

# Start to retrieve files
# Get a list of all files in the directory
file_list_1D = os.listdir(parent_path + folder_name)

# Sort the list of files (Necessary on Linux)
file_list_1D.sort()

# Get a list of all files in the directory
file_list_2D = os.listdir(parent_path + folder_name_2D)

# Sort the list of files (Necessary on Linux)
file_list_2D.sort()

intend_date_list = []

for date in csv_date_list:
    intend_date = date[:4] + '.' + date[4:6] + '.' + date[6:]
    intend_date_list.append(intend_date)

intend_time_list = [t.replace(':', '') for t in csv_time_list]
intend_timestamp_list = [f"{date}.{time}" for date, time in zip(intend_d

Date_UTC = []

for date in csv_date_list:
    intend_date = date[:4] + '-' + date[4:6] + '-' + date[6:]
    Date_UTC.append(intend_date)

Time_UTC = csv_time_list.copy()

file_1D_timestamp = []

for filename in file_list_1D:
    file_1D = filename[7:22]
    file_1D_timestamp.append(file_1D)
```

```
# Define UTC and CST time zones
utc_tz = timezone('UTC')
cst_tz = timezone('Etc/GMT+6')

# Example lists of datetime strings
date_utc_list = Date_UTC.copy()
time_utc_list = Time_UTC.copy()

# Convert datetime strings from UTC to CST
date_cst_list = []
time_cst_list = []
for date_str, time_str in zip(date_utc_list, time_utc_list):
    datetime_utc = datetime.strptime(date_str + ' ' + time_str, '%Y-%m-%
    datetime_utc = utc_tz.localize(datetime_utc)
    datetime_cst = datetime_utc.astimezone(cst_tz)
    date_cst_list.append(datetime_cst.strftime('%Y-%m-%d'))
    time_cst_list.append(datetime_cst.strftime('%H:%M'))

data_dict = {'Date_UTC': Date_UTC,
         'Time_UTC': Time_UTC,
         'Date_CST': date_cst_list,
         'Time_CST': time_cst_list,
         'intend_timestamp_list': intend_timestamp_list}

# create the DataFrame using the dictionary
df_GOES_time_lib = pd.DataFrame(data_dict)

df_GOES_file_lib = pd.DataFrame({'file_timestamp': file_1D_timestamp, 'f

df_GOES_combined = pd.merge(df_GOES_time_lib, df_GOES_file_lib, left_on=
df_GOES_combined.fillna('None', inplace=True)

output_dir = 'output/GOES_file_lib_dir/'
output_csv_name = str(start_year)+'Fall_'+str(end_year)+'Spring_GOES_lib

output_file_path = os.path.join(output_dir, output_csv_name)

df_GOES_combined.to_csv(output_file_path, index=False)

###################
df_reference_1D = pd.read_csv('02-05-2023/zone_0_sample_take_2/' + 'goes
table_1D_len = df_reference_1D.shape[0]
df_lake_1D_map = df_reference_1D[['latitude', 'longitude']].copy()
matched_1D_file_list = df_GOES_combined['file_list_1D'].tolist()
matched_2D_file_list = df_GOES_combined['file_list_2D'].tolist()
matrix = df_reference_1D.values
df_tester_again = pd.read_csv('02-05-2023/zone_0_sample_take_2/' + 'goes
temp_result_tester = lake_1D_matcher(df_tester_again, df_lake_1D_map)

###################
lake_1D_list = []
# lat_lists = []
# lon_lists = []
cond_list = []
count_list = []
```

```
    cloud_exist_list = []
    for file_name in tqdm(matched_1D_file_list):
    # for file_name in matched_1D_file_list[1769:1775]:

        try:
            temp_file_path = parent_path + folder_name + '/' + file_name
    #         print( temp_file_path)
    #         print(counter)
            df_temp = pd.read_csv(temp_file_path)
            value_temp = lake_1D_matcher(df_temp, df_lake_1D_map)
            cond = is_valid_data(df_temp)
            cond_list.append(cond)
            lake_1D_list.append(value_temp)
            num_clouds = cloud_finder(value_temp)
            count_list.append(num_clouds)
            if num_clouds < 720:
                exist_temp = False
            else:
                exist_temp = True
            # Replace all NaN values in the 'exist_temp' array with 0.0
            exist_temp = np.nan_to_num(exist_temp, nan=0.0)
            cloud_exist_list.append(exist_temp)
    #         lat_lists.append(lat_list)
    #         lon_lists.append(lon_list)
        except FileNotFoundError:
            lake_1D_list.append(np.zeros(3599))
            cond_list.append(False)
            count_list.append(0)
            cloud_exist_list.append(False)

    # list of matrices
    lake_2D_list = []

#     # loop over file names
#     for file_name in tqdm(matched_2D_file_list):
#         try:
#             # read csv file into dataframe
#             temp_file_path = parent_path + folder_name_2D + '/' + file_nam
#             df_temp = pd.read_csv(temp_file_path)
#             df_temp = df_temp.iloc[1:, 1:]
#             # Replace NaN values in 'values' with 0 in 'df_temp'
#             df_temp = df_temp.fillna(0)
#     #         print(df_temp.shape)
#             # convert dataframe to numpy array/matrix
#             mat_temp = df_temp.values
#     #         mat_temp = np.array(df_temp.values.flatten())

#             # assume df_temp is a DataFrame with multiple columns
#     #         arrays = [df_temp.iloc[:, i].to_numpy() for i in range(len(d
#     #         mat_temp = arrays
#         except FileNotFoundError:
#             # if file does not exist, save NaN
#             mat_temp = np.zeros((105, 79))

#         # append matrix to list
#         lake_2D_list.append(mat_temp)
```

```
    df_GOES_combined['lake_1D_list'] = lake_1D_list
#     df_GOES_combined['lake_2D_list'] = lake_2D_list
    df_GOES_combined['data_usable'] = cond_list
    df_GOES_combined['cloud_count'] = count_list
    df_GOES_combined['cloud_exist'] = cloud_exist_list

    df_GOES_meteo_combined = pd.merge(df_GOES_combined, weather_with_CST, or

    column_names = df_GOES_meteo_combined.columns.tolist()

    df_GOES_meteo_combined = df_GOES_meteo_combined.drop(['Date', 'Time', 'i

    df_GOES_meteo_combined.rename(columns={'file_list_1D': 'File_name_for_1D
    df_GOES_meteo_combined.rename(columns={'file_list_2D': 'File_name_for_2D
    df_GOES_meteo_combined.rename(columns={'lake_1D_list': 'Lake_data_1D'},
#     df_GOES_meteo_combined.rename(columns={'lake_2D_list': 'Lake_data_2D'}

    column_names = df_GOES_meteo_combined.columns.tolist()
    # Concatenate the temporary data to the 'df_combined_all' dataframe
    df_combined_all = pd.concat([df_combined_all, df_GOES_meteo_combined], i

    # Store the length of 'df_GOES_meteo_combined' in the 'lengths_list'
    lengths_list.append(len(df_GOES_meteo_combined))
```

```
Weather station data exists for  MI-14850-TRAVERSE_CITY_CHERRY_CPTL_AP
File exists, reading table!
1-D Lake Michigan data exists for  2006Fall  to  2007Spring
2-D Lake Michigan data exists for  2006Fall  to  2007Spring
/tmp/ipykernel_58001/1036582874.py:175: ParserWarning: Falling back to the
'python' engine because the 'c' engine does not support skipfooter; you can
avoid this warning by specifying engine='python'.
  temp_table = pd.read_csv('NSW_Weather/'+weather_station+'/'+filename, ski
prows = 8, skipfooter = 15)
100%|██████████| 4368/4368 [01:07<00:00, 64.64it/s]
/tmp/ipykernel_58001/1036582874.py:175: ParserWarning: Falling back to the
'python' engine because the 'c' engine does not support skipfooter; you can
avoid this warning by specifying engine='python'.
  temp_table = pd.read_csv('NSW_Weather/'+weather_station+'/'+filename, ski
prows = 8, skipfooter = 15)
Weather station data exists for  MI-14850-TRAVERSE_CITY_CHERRY_CPTL_AP
File exists, reading table!
1-D Lake Michigan data exists for  2007Fall  to  2008Spring
2-D Lake Michigan data exists for  2007Fall  to  2008Spring
100%|██████████| 4392/4392 [01:11<00:00, 61.48it/s]
/tmp/ipykernel_58001/1036582874.py:175: ParserWarning: Falling back to the
'python' engine because the 'c' engine does not support skipfooter; you can
avoid this warning by specifying engine='python'.
  temp_table = pd.read_csv('NSW_Weather/'+weather_station+'/'+filename, ski
prows = 8, skipfooter = 15)
Weather station data exists for  MI-14850-TRAVERSE_CITY_CHERRY_CPTL_AP
File exists, reading table!
1-D Lake Michigan data exists for  2008Fall  to  2009Spring
2-D Lake Michigan data exists for  2008Fall  to  2009Spring
```

```
100%|██████████| 4368/4368 [01:12<00:00, 60.25it/s]
/tmp/ipykernel_58001/1036582874.py:175: ParserWarning: Falling back to the
'python' engine because the 'c' engine does not support skipfooter; you can
avoid this warning by specifying engine='python'.
  temp_table = pd.read_csv('NSW_Weather/'+weather_station+'/'+filename, ski
prows = 8, skipfooter = 15)
Weather station data exists for  MI-14850-TRAVERSE_CITY_CHERRY_CPTL_AP
File exists, reading table!
1-D Lake Michigan data exists for  2009Fall  to  2010Spring
2-D Lake Michigan data exists for  2009Fall  to  2010Spring
100%|██████████| 4368/4368 [01:01<00:00, 71.25it/s]
/tmp/ipykernel_58001/1036582874.py:175: ParserWarning: Falling back to the
'python' engine because the 'c' engine does not support skipfooter; you can
avoid this warning by specifying engine='python'.
  temp_table = pd.read_csv('NSW_Weather/'+weather_station+'/'+filename, ski
prows = 8, skipfooter = 15)
Weather station data exists for  MI-14850-TRAVERSE_CITY_CHERRY_CPTL_AP
File exists, reading table!
1-D Lake Michigan data exists for  2010Fall  to  2011Spring
2-D Lake Michigan data exists for  2010Fall  to  2011Spring
100%|██████████| 4368/4368 [00:55<00:00, 79.07it/s]
/tmp/ipykernel_58001/1036582874.py:175: ParserWarning: Falling back to the
'python' engine because the 'c' engine does not support skipfooter; you can
avoid this warning by specifying engine='python'.
  temp_table = pd.read_csv('NSW_Weather/'+weather_station+'/'+filename, ski
prows = 8, skipfooter = 15)
Weather station data exists for  MI-14850-TRAVERSE_CITY_CHERRY_CPTL_AP
File exists, reading table!
1-D Lake Michigan data exists for  2011Fall  to  2012Spring
2-D Lake Michigan data exists for  2011Fall  to  2012Spring
100%|██████████| 4392/4392 [01:14<00:00, 58.82it/s]
/tmp/ipykernel_58001/1036582874.py:175: ParserWarning: Falling back to the
'python' engine because the 'c' engine does not support skipfooter; you can
avoid this warning by specifying engine='python'.
  temp_table = pd.read_csv('NSW_Weather/'+weather_station+'/'+filename, ski
prows = 8, skipfooter = 15)
Weather station data exists for  MI-14850-TRAVERSE_CITY_CHERRY_CPTL_AP
File exists, reading table!
1-D Lake Michigan data exists for  2012Fall  to  2013Spring
2-D Lake Michigan data exists for  2012Fall  to  2013Spring
100%|██████████| 4368/4368 [01:21<00:00, 53.47it/s]
/tmp/ipykernel_58001/1036582874.py:175: ParserWarning: Falling back to the
'python' engine because the 'c' engine does not support skipfooter; you can
avoid this warning by specifying engine='python'.
  temp_table = pd.read_csv('NSW_Weather/'+weather_station+'/'+filename, ski
prows = 8, skipfooter = 15)
Weather station data exists for  MI-14850-TRAVERSE_CITY_CHERRY_CPTL_AP
File exists, reading table!
1-D Lake Michigan data exists for  2013Fall  to  2014Spring
2-D Lake Michigan data exists for  2013Fall  to  2014Spring
```

```
100%|██████████| 4368/4368 [01:01<00:00, 71.22it/s]
/tmp/ipykernel_58001/1036582874.py:175: ParserWarning: Falling back to the
'python' engine because the 'c' engine does not support skipfooter; you can
avoid this warning by specifying engine='python'.
  temp_table = pd.read_csv('NSW_Weather/'+weather_station+'/'+filename, ski
prows = 8, skipfooter = 15)
Weather station data exists for  MI-14850-TRAVERSE_CITY_CHERRY_CPTL_AP
File exists, reading table!
1-D Lake Michigan data exists for  2014Fall  to  2015Spring
2-D Lake Michigan data exists for  2014Fall  to  2015Spring
100%|██████████| 4368/4368 [01:12<00:00, 60.01it/s]
/tmp/ipykernel_58001/1036582874.py:175: ParserWarning: Falling back to the
'python' engine because the 'c' engine does not support skipfooter; you can
avoid this warning by specifying engine='python'.
  temp_table = pd.read_csv('NSW_Weather/'+weather_station+'/'+filename, ski
prows = 8, skipfooter = 15)
Weather station data exists for  MI-14850-TRAVERSE_CITY_CHERRY_CPTL_AP
File exists, reading table!
1-D Lake Michigan data exists for  2015Fall  to  2016Spring
2-D Lake Michigan data exists for  2015Fall  to  2016Spring
100%|██████████| 4392/4392 [01:18<00:00, 55.67it/s]
/tmp/ipykernel_58001/1036582874.py:175: ParserWarning: Falling back to the
'python' engine because the 'c' engine does not support skipfooter; you can
avoid this warning by specifying engine='python'.
  temp_table = pd.read_csv('NSW_Weather/'+weather_station+'/'+filename, ski
prows = 8, skipfooter = 15)
Weather station data exists for  MI-14850-TRAVERSE_CITY_CHERRY_CPTL_AP
File exists, reading table!
1-D Lake Michigan data exists for  2016Fall  to  2017Spring
2-D Lake Michigan data exists for  2016Fall  to  2017Spring
100%|██████████| 4368/4368 [01:15<00:00, 58.22it/s]
```

In [2]:

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, Dense, Flatten, Dropout

X = np.stack(df_filtered['Lake_data_1D'].to_numpy())

df_filtered['does_snow_24_120'] = df_filtered['does_snow_24_120'].apply(lamb

y = df_filtered['does_snow_24_120'].values.astype(int)
# print(y)

# Fill NaN values with 0
X = np.nan_to_num(X)
y = np.nan_to_num(y)


input_data = []
output_data = []
```

```
for i in range(len(X) - 35):
    input_data.append(X[i:i+21])
    output_data.append(y[i+35])

input_data = np.stack(input_data)
output_data = np.stack(output_data)


# Scale the input data
scaler = StandardScaler()
input_data_scaled = scaler.fit_transform(input_data.reshape(input_data.shape

# Reshape the input data to match Conv1D input shape (batch_size, steps, inp
input_data_scaled = input_data_scaled.reshape(input_data_scaled.shape[0], in

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(input_data_scaled, outpu


model = Sequential()
model.add(Conv1D(64, 3, activation='relu', input_shape=(X_train.shape[1], X_
model.add(Conv1D(128, 3, activation='relu'))
model.add(Dropout(0.1))
model.add(Conv1D(128, 3, activation='relu'))
model.add(Dropout(0.1))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['mae',


# model.compile(optimizer='adam', loss='mse', metrics=['mae', 'accuracy'])


history = model.fit(X_train, y_train, epochs=200, batch_size=64, validation_
```

```
2023-05-11 08:03:18.998254: I tensorflow/core/platform/cpu_feature_guard.c
c:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network
Library (oneDNN) to use the following CPU instructions in performance-criti
cal operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 10
      7 from tensorflow.keras.models import Sequential
      8 from tensorflow.keras.layers import Conv1D, Dense, Flatten, Dropout
---> 10 X = np.stack(df_filtered['Lake_data_1D'].to_numpy())
     12 df_filtered['does_snow_24_120'] = df_filtered['does_snow_24_120'].a
pply(lambda x: int(round(x)) if isinstance(x, float) and not np.isnan(x) el
se (int(x) if not np.isnan(x) else 0))
     14 y = df_filtered['does_snow_24_120'].values.astype(int)

NameError: name 'df_filtered' is not defined
```

In [ ]: