General Description:

Our model uses python, specifically a method called random forest regressor, to predict temperature in San Diego Bay based on previous ARMS data.

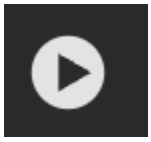
How it works (simplified):

Random forest regression, simplified, is a string of decision trees. It involves taking a dataframe and shuffling it, called "bootstrapping". For each shuffle or bootstrap, the model learns a decision tree. Our data goes through 28 of these iterations, and outputs an averaged prediction over all the decision trees.

Using established libraries, we can graphically represent existing temperatures and predictions.

How to Use (cell by cell instructions):

To run code in google collab, press the “play” button in the upper left-hand corner of each cell. **Cells NEED to be run in chronological order**.



1)

```
import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np
from google.colab import files
```

Run the first cell. This imports libraries that contain existing python functions, and allow graphing.

2)

```
uploaded = files.upload() #You should download the SGYC.xlsx file as a .csv and then upload it here
```

Choose Files No file chosen Cancel upload

Running the second cell will prompt you with a .csv file. You should upload a .csv file (by selecting “Choose Files”) containing existing ARMS data. To save a .csv from a spreadsheet, select file > download > .csv

3)

```
df = pd.read_csv("SGYC.csv") #Creating a dataframe using the name of the file pasted above


df.drop('#', axis=1,inplace=True) #Dropping the number column since we already have this by default
df.columns = ['Date-Time', 'Temp°C', 'lux'] #Renaming columns for easier legibility

temp_list = df['Temp°C'].tolist() #Converting the column to a list so that we can iterate over it


def group(temp_list,group_size):
  return [temp_list[i:i + group_size] for i in range(0, len(temp_list), group_size)] #Groups the list of temperat

"""
This (0, len(temp_list), group_size) represents what is being iterated over.
the loop is bound to the length of temp_list, but group_size represents the step size
"""



sorted = group(temp_list,60)


sorted = pd.DataFrame(sorted) #Rows are entries, Columns are groups
```

Run the third cell. This code converts the dataframe from google sheets (the .csv file) to a dataframe that python can understand.

4)

```
  df['Date-Time'] = pd.to_datetime(df['Date-Time'], format="%m/%d/%Y %H:%M:%S")

  # Take every 60th timestamp (1 per hour)
  hour_labels = df['Date-Time'][::60].dt.strftime("%m/%d/%Y %H:%M:%S").tolist()
  hour_labels.pop()
  print(len(hour_labels))
  print(hour_labels[:5])   # preview
```

Run the fourth cell. Ensures that date-time format is understandable to python. Takes every 60th timestamp (1 per hour).

5)

```
sorted.head()
```

Run the fifth cell. All this does is ensure we have altered our data correctly.

6)

```
def average_temp(data_frame):
    averages = []
    for i in range(len(data_frame)):
        data_list = data_frame.iloc[i].tolist()
        average_temp = np.mean(data_list)
        if np.isfinite(average_temp): #Accounting for NaNs
          averages.append(float(average_temp))
    return averages

averages = average_temp(sorted)



average_temper = pd.DataFrame(averages)

average_temper.columns = ["Average temperature (celsius)"]

average_temper.head()
```

Run the sixth cell. Until now, our temperature data is by every minute. To simplify graphing, we are averaging this to every hour.

7)

```
def set_depth_location(depth, location, data_frame):
  # Create lists with the same length as the DataFrame index
  locat = list([location] * len(data_frame))
  dep = list([depth] * len(data_frame))

  data_frame["Location"] = locat #adding new columns
  data_frame["Depth (ft)"] = dep
  return data_frame

averages = set_depth_location("8 ft", "SGYC", average_temper)


averages["Date"] = hour_labels
```

Run the 7th cell. This adds additional labels to our data, including location, depth, and date.

8)

```
averages.head()
```

Run cell 8 (emulates cell 5). Ensures we are doing everything correctly.

9)

```
plt_graph = averages.drop(columns=["Depth (ft)", "Location"])
```

Run cell 9. This removes columns we can't show on our graph.

10)
Preprocessing another dataset